HAT: Hardware-Aware Transformers for Efficient Natural Language Processing

Hanrui Wang[1], **Zhanghao Wu**[1], **Zhijian Liu**[1], **Han Cai**[1], **Ligeng Zhu**[1], **Chuang Gan**[2], **Song Han**[1]
[1]Massachusetts Institute of Technology, [2]MIT-IBM Watson AI Lab
{hanrui,zhwu,zhijian,hancai,ligeng,chuangg,songhan}@mit.edu

Abstract

Transformers are ubiquitous in Natural Language Processing (NLP) tasks, but they are difficult to be deployed on hardware due to the intensive computation. To enable low-latency inference on resource-constrained hardware platforms, we propose to design Hardware-Aware Transformers (HAT) with neural architecture search. We first construct a large design space with *arbitrary encoder-decoder attention* and *heterogeneous layers*. Then we train a *SuperTransformer* that covers all candidates in the design space, and efficiently produces many *SubTransformers* with weight sharing. Finally, we perform an evolutionary search with a hardware latency constraint to find a specialized *SubTransformer* dedicated to run fast on the target hardware. Extensive experiments on four machine translation tasks demonstrate that HAT can discover efficient models for different hardware (CPU, GPU, IoT device). When running WMT'14 translation task on Raspberry Pi-4, HAT can achieve **3**× speedup, **3.7**× smaller size over baseline Transformer; **2.7**× speedup, **3.6**× smaller size over Evolved Transformer with **12,041**× less search cost and no performance loss. HAT is open-sourced.

1 Introduction

Transformer (Vaswani et al., 2017) has been widely used in natural language processing tasks. By stacking multiple identical encoder/decoder layers with attention modules, it provides a significant performance improvement over previous convolutional or recurrent neural network models (Kim, 2014).

Nevertheless, it is challenging to deploy Transformers on mobile devices due to the high computation cost. For instance, in order to translate a sentence with only 30 words, a Transformer-Big model needs to execute 13G FLOPs and takes 20 seconds on a Raspberry Pi. Such long latency will hurt the user experience on edge devices. Thus we need hardware-efficient Transformers (Figure 1).

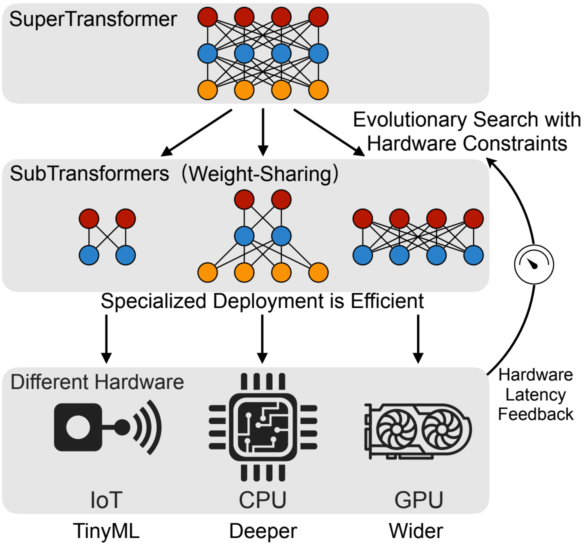

SuperTransformer
Evolutionary Search with Hardware Constraints
SubTransformers (Weight-Sharing)
Specialized Deployment is Efficient
Hardware Latency Feedback
Different Hardware
IoT
TinyML
CPU
Deeper
GPU
Wider

Figure 1: Framework for searching Hardware-Aware Transformers. We first train a SuperTransformer that contains numerous sub-networks, then conduct an evolutionary search with hardware latency feedback to find one **specialized** SubTransformer for each hardware.

There are two common pitfalls when evaluating the efficiency of a Transformer. *(1) FLOPs does not reflect the measured latency*. Although FLOPs is used as an metric for efficiency in prior arts (Howard et al., 2017; Wu et al., 2020), it is not a good latency proxy. As in Figure 2 (*Right*), models with the *same* FLOPs can result in very *different* measured latencies; *(2) different hardware prefers different Transformer architecture.* As in Table 1, the Transformer model optimized on one hardware is *sub-optimal* for another because latency is influenced by different factors on different hardware platforms. For example, the embedding size has significant impact on the Raspberry Pi latency but hardly influences the GPU latency (Figure 2).

Inspired by the success of Neural Architecture Search (NAS) (Bender et al., 2018; Guo et al., 2019; Pham et al., 2018; Cai et al., 2019a), we propose to search for **H**ardware-**A**ware **T**ransformers (HAT)

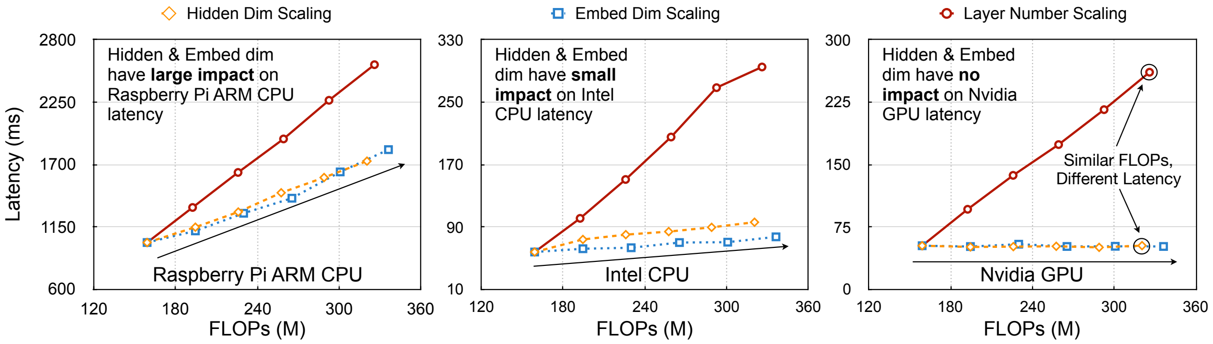


Figure 2: Latency of different Transformer models on different hardware. We find (1) FLOPs does not reflect the real measured latency; (2) Latency influencing factors of different hardware are contrasting. Thus we need to consider hardware latency feedback to design specialized models for different hardware.

Measured On → Specialized For ↓	BLEU	GPU Latency	ARM CPU Latency
HAT (GPU)	28.10	**147 ms**	6491 ms
HAT (ARM CPU)	28.15	184 ms	**6042 ms**

Table 1: BLEU score and measured inference latency of HAT on WMT'14 En-De task. The efficient model for GPU is not efficient for ARM CPU and vice versa.

by *directly* involving the latency feedback into the design loop. In this way, we do not need FLOPs as the latency proxy and can search specialized models for various hardware.

We first construct a large search space with *arbitrary encoder-decoder attention* and *heterogeneous Transformer layers*. Traditional Transformer has an information bottleneck between the encoder and decoder. Arbitrary encoder-decoder attention breaks the bottleneck, allowing all decoder layers to attend to multiple and different encoder layers instead of only the last one. Thus low-level information from the encoder can also be used by the decoder. Motivated by Figure 2, we introduce heterogeneous Transformer layers to allow different layers to have different architecture adapting various hardware.

To perform a low-cost search in such a large design space, we first train a Transformer supernet – SuperTransformer, which contains many SubTransformers sharing the weights. We train all SubTransformers simultaneously by optimizing the uniformly sampled SubTransformers from the SuperTransformer. The performance of a SubTransformer with inherited weights from the SuperTransformer can provide a good relative performance approximation for different architectures trained from-scratch. Unlike conventional NAS, we only need to pay the SuperTransformer training cost for *once* and can evaluate *all* the models in the design space with it. Finally, we conduct an evolutionary search to find the best SubTransformer under the hardware latency constraint. Experiments show that HAT can be naturally incorporated with model compression techniques such as quantization and knowledge distillation.

We evaluate HAT with WMT'14 En-De, WMT'14 En-Fr, WMT'19 En-De, and IWSLT'14 De-En tasks on Raspberry Pi ARM CPU, Intel Xeon CPU, and Nvidia TITAN Xp GPU. Compared with previous work (Vaswani et al., 2017; So et al., 2019; Gu et al., 2019; Wu et al., 2020), HAT achieves up to **3**× speedup, **3.7**× smaller size over Transformer-Big without loss of accuracy. With **12,041**× less search cost, HAT outperforms the Evolved Transformer with **2.7**× speedup and **3.6**× smaller size. It also achieves up to **1.9**× speedup over Levenshtein and Lite Transformers with no BLEU score loss. With 4-bit quantization, HAT can further reach **25**× model size reduction.

HAT has three contributions: (1) **Hardware-Aware and Specialization.** To our best knowledge, we are the first to directly involve the hardware feedback in the model design, to reduce NLP model latency for target hardware, instead of relying on proxy signals (FLOPs). For different hardware platforms, specialized models for low-latency inference are explored. (2) **Low-cost Neural Architecture Search with a Large Design Space.** We propose *arbitrary encoder-decoder attention* to break the information bottleneck; and *heterogeneous layer* to let different layers alter its capacity. A weight-shared SuperTransformer is trained to search for efficient models at a low cost. (3) **Design Insights.** Based on the search results, we reveal some design insights: Attending to multiple encoder layers is beneficial for the decoder; GPU prefers shallow and wide models while ARM CPU prefers deep and thin ones.

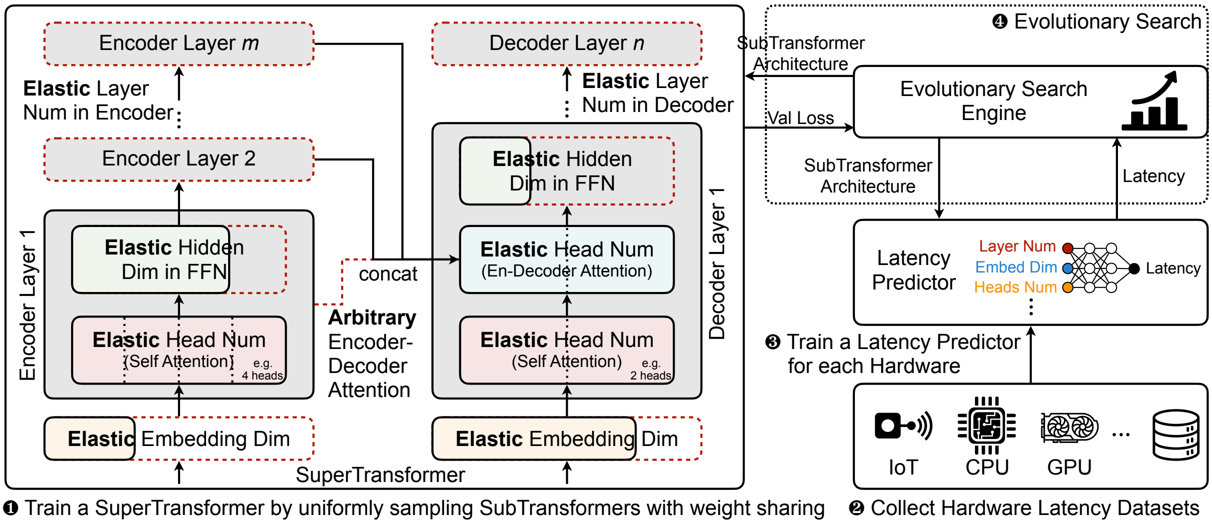

Encoder Layer *m*
Elastic Layer Num in Encoder
Encoder Layer 2
Elastic Hidden Dim in FFN
Elastic Head Num (Self Attention) e.g. 4 heads
Encoder Layer 1
Elastic Embedding Dim
concat
Arbitrary Encoder-Decoder Attention
Decoder Layer *n*
Elastic Layer Num in Decoder
Elastic Hidden Dim in FFN
Elastic Head Num (En-Decoder Attention)
Elastic Head Num (Self Attention) e.g. 2 heads
Decoder Layer 1
Elastic Embedding Dim
SuperTransformer
❶ Train a SuperTransformer by uniformly sampling SubTransformers with weight sharing
❹ Evolutionary Search
SubTransformer Architecture
Val Loss
Evolutionary Search Engine
SubTransformer Architecture
Latency
Latency Predictor
Layer Num
Embed Dim
Heads Num
Latency
❸ Train a Latency Predictor for each Hardware
IoT
CPU
GPU
...
❷ Collect Hardware Latency Datasets

Figure 3: HAT Overview. A large design space is constructed with Arbitrary Encoder-Decoder Attention and Heterogeneous Layers. (1) Train a weight-shared SuperTransformer by iteratively optimizing randomly sampled SubTransformers. It can provide a performance proxy for SubTransformers. (2) Collect *(SubTransformer architecture, latency)* data pairs on the target hardware. (3) Train a latency predictor for each hardware to provide fast and accurate latency feedback. (4) Perform an evolutionary search with hardware latency constraint to find the model with the *lowest validation loss*. (5) Finally, the searched model is trained *from scratch* to get the final performance.

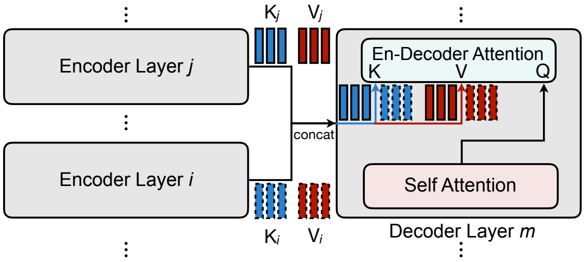


Figure 4: Arbitrary Encoder-Decoder Attention. Each encoder-decoder attention in one decoder layer can attend to the outputs from multiple encoder layers, fully leveraging the features extracted by the encoder.

2 Proposed Approaches

An overview of the HAT framework is shown in Figure 3. We firstly train a SuperTransformer with a large design space. Then, for a given hardware platform, we collect a dataset of *(SubTransformer architecture, measured latency)* pairs for different models, and train a latency predictor. Finally, we conduct an evolutionary search with a latency constraint to find an efficient model specialized for the target hardware.

2.1 Design Space

We construct a large design space by breaking two conventions in the Transformer design: (1) All decoder layers only attend to the last encoder layer; (2) All the layers are identical.

Arbitrary Encoder-Decoder Attention. Different encoder layers extract features on different abstraction levels. Conventionally, all the decoder layers only attend to the last encoder layer. It forms an *information bottleneck* that forces all the decoder layers to learn solely from the high abstraction level and ignore the low-level information. To break the bottleneck, we propose Arbitrary Encoder-Decoder Attention to learn the most suitable connections between the encoder and the decoder. Each decoder layer can choose *multiple* encoder layers to attend. The *key* and *value* vectors from encoder layers are concatenated in the *sentence length dimension* (Figure 4) and fed to the encoder-decoder cross attention module. The mechanism is efficient because it introduces no additional parameters. The latency overhead is also negligible. For example, with each decoder layer attending to two encoder layers, the latency of Transformer-Base on Nvidia TITAN Xp GPU barely increases by 0.4%. It improves the model capacity by allowing attention to different abstraction levels.

Heterogeneous Transformer Layers. Previous Transformers repeat one architecture for all layers. In HAT, instead, different layers are *heterogeneous*, with different numbers of heads, hidden dim, and embedding dim. In attention layers, different heads are used to capture various dependencies. However, Voita et al. (2019) shows that many heads are redundant. We thereby make attention head number *elastic* so that each attention module can decide its necessary number of heads.

In the FFN layer, the input features are cast to a higher dimension (hidden dim), followed by an

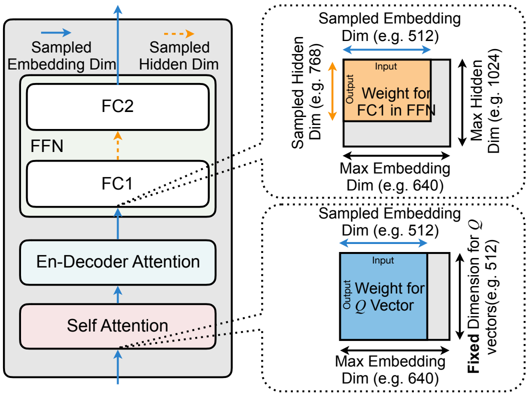


Figure 5: Weight Sharing of the SuperTransformer. All SubTransformers share the front portion of word embeddings, and weights in the fully-connected layers.

activation layer. Traditionally, the hidden dim is set as 2× or 4× of the embedding dim, but this is sub-optimal since different layers need different capacities depending on the feature extraction difficulty. We hence make the hidden dim *elastic*.

Moreover, we also support *elastic* embedding dim of encoder and decoder, but it is consistent inside encoder/decoder. The number of encoder & decoder layers are also *elastic* to learn the proper level of feature encoding and decoding. Other design choices such as the length of Q, K, V vectors in attention modules can be naturally incorporated in our framework, which we leave for future work.

2.2 SuperTransformer

It is critical to have a large design space in order to find high-performance models. However, training all the models and comparing their BLEU scores is infeasible. We thus propose SuperTransformer, a supernet for *performance approximation*, which can judge the performance of a model without fully training it. The SuperTransformer is the largest model in the search space with *weight sharing* (Pham et al., 2018; Liu et al., 2019; Cai et al., 2019a). Every model in the search space (a SubTransformer) is a part of the SuperTransformer. All SubTransformers share the weights of their common parts. For elastic embedding dim, all SubTransformers share the front portion of the longest word embedding and corresponding FC layer weights. As in Figure 5, for elastic FFN hidden dim, the front part of the FC weights is shared. For elastic head number in attention modules, the whole Q, K, V vectors (the lengths are fixed in our design space) are shared by dividing into $head\_number$ parts. Elastic layer numbers let all SubTransformers share the first several layers.

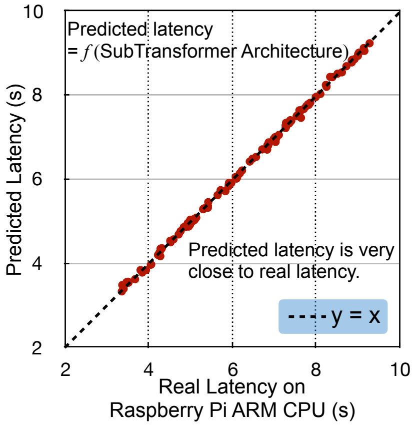


Figure 6: The latency predictor is very accurate, with an average prediction error (RMSE) of 0.1s.

In the SuperTransformer training, all possible SubTransformers are *uniformly sampled*, and the corresponding weights are updated. In practice, the SuperTransformer only needs to be trained for the same steps as a baseline Transformer model, which is fast and low-cost. After training, we can get the performance proxy of sampled models in the design space by evaluating the corresponding SubTransformers on the validation set without training.

2.3 Evolutionary Search for SubTransformer

Given a latency requirement, we perform an evolutionary search to find a satisfactory SubTransformer. There are two ways to evaluate the hardware latency of a SubTransformer: (1) Online measurement in which we measure the models during the search process. (2) Offline, where we train a *latency predictor* to provide the latency. We apply the offline method here because it is *fast and accurate*. For the online method, a single sampled SubTransformer requires hundreds of inferences to get an accurate latency, which lasts for minutes and slows down the searching. For the offline method, we encode the architecture of a SubTransformer into a feature vector, and predict its latency instantly with a multi-layer perceptron (MLP). Trained with thousands of real latency data points, the predictor yields high accuracy (Figure 6). Note that the predicted latency is only used in the search process, and we report *real measured latency* in the experiment section. Compared with deducing a closed-form latency model for each hardware, the latency predictor method is more general and faster.

We use an evolutionary algorithm to conduct the search process. As in Figure 3, the search engine queries the latency predictor for SubTransformer latency, and validates the loss on the validation set. The engine only adds SubTransformers with latency *smaller than* the hardware constraint to the population. We then train the searched models *from scratch* to obtain the final performance.

3 Experiments

3.1 Datasets

We conduct experiments on four machine translation tasks: WMT'14 En-De, WMT'14 En-Fr, WMT'19 En-De, and IWSLT'14 De-En, consisting of 4.5M, 36.3M, 43.0M, and 160K pairs of training sentences, respectively. For WMT'14 En-De, we apply 32K source-target BPE vocabulary, validate on newstest2013 and test on newstest2014, replicating Wu et al. (2019b); For WMT'14 En-Fr, we use 40K source-target BPE vocabulary, validate on newstest2012&2013, and test on newstest2014, replicating Gehring et al. (2017). For WMT'19 En-De, we use 49.6K source-target BPE vocabulary, validate on newstest2017, and test on newstest2018, the same as Junczys-Dowmunt (2019). We adopt 10K joint BPE vocabulary in lower case for IWSLT'14 De-En (Grave et al., 2017).

3.2 Experiment Setups

Baselines. Our baseline models are Transformer (Vaswani et al., 2017), Levenshtein Transformer (Gu et al., 2019), both with the Ott et al. (2019) implementation, Evolved Transformer (So et al., 2019) and Lite Transformer (Wu et al., 2020).

Evaluation Metrics. For evaluation, we use beam four and length penalty 0.6 for WMT, and beam five for IWSLT (Vaswani et al., 2017). All BLEUs are calculated with case-sensitive tokenization[1], but we also apply the compound splitting BLEU[2] for WMT, the same as Vaswani et al. (2017). We test the model with the lowest validation set loss for WMT and the last ten checkpoints averaged for IWSLT.

We test the latency of the models by measuring translation from a source sentence to a target sentence with the same length. The length is the average output length on the test set – 30 for WMT and 23 for IWSLT. For each model, we measure the latency for 300 times, remove the fastest and slowest 10% and then take the average of the rest 80%. We conduct experiments on three representative hardware platforms: Raspberry Pi-4 with an ARM Cortex-A72 CPU, Intel Xeon E5-2640 CPU, and Nvidia TITAN Xp GPU.

3.3 Implementation Details

SuperTransformer Setups. The SuperTransformer for WMT has the following design space: [512, 640] for embedding dim, [1024, 2048, 3072] for hidden dim, [4, 8] for the head number in all attention modules, [1, 2, 3, 4, 5, 6] for decoder layer number. Due to decoder auto-regression, encoder only accounts for less than 5% of the measured latency; thereby, we set the encoder layer number fixed as 6. For arbitrary encoder-decoder attention, each decoder can choose to attend to the last one, two, or three encoder layers. The SuperTransformer design space for IWSLT is the same as WMT except for [2048, 1024, 512] for hidden dim and [4, 2] for head number. We set the Q, K, V vector dim fixed as 512. The design space contains around 10^{15} possible SubTransformers and covers a wide range of model size and latency (largest = 6×smallest). We train the SuperTransformers of WMT for 40K steps and 50K steps for IWSLT.

Hardware-Aware Evolutionary Search Setups. The input of the latency predictor is a feature vector of SubTransformer architecture with ten elements: layer number, embed dim, average hidden dim, average self-attention heads, of both encoder and decoder; plus average encoder-decoder attention heads, and the average number of encoder layers each decoder layer attends. A dataset of 2000 (SubTransformer architecture, measured latency) samples for each hardware is collected, and split into train:valid:test=8:1:1. We normalize the features and latency, and train a three-layer MLP with 400 hidden dim and ReLU activation. We choose three-layer because it is more accurate than the one-layer model, and over three layers do not improve accuracy anymore. With the predictor, we conduct an evolutionary search for 30 iterations in the SuperTransformer, with population 125, parents population 25, mutation population 50 with 0.3 probability and crossover population 50.

Training Settings. Our training settings are in line with Wu et al. (2019b) and Wu et al. (2020). For WMT, we train for 40K steps with Adam optimizer and a cosine learning rate (LR) scheduler (Kingma and Ba, 2015; Loshchilov and Hutter, 2017), where the LR is linearly warmed up from 10^{-7} to 10^{-3}, and then cosine annealed. For IWSLT, we train for 50K steps with inverse square root LR scheduler. The baseline Transformers are trained with the *same settings* as the searched SubTransformers for fair comparisons.

[1] https://github.com/moses-smt/mosesdecoder

[2] https://github.com/tensorflow/tensor2tensor

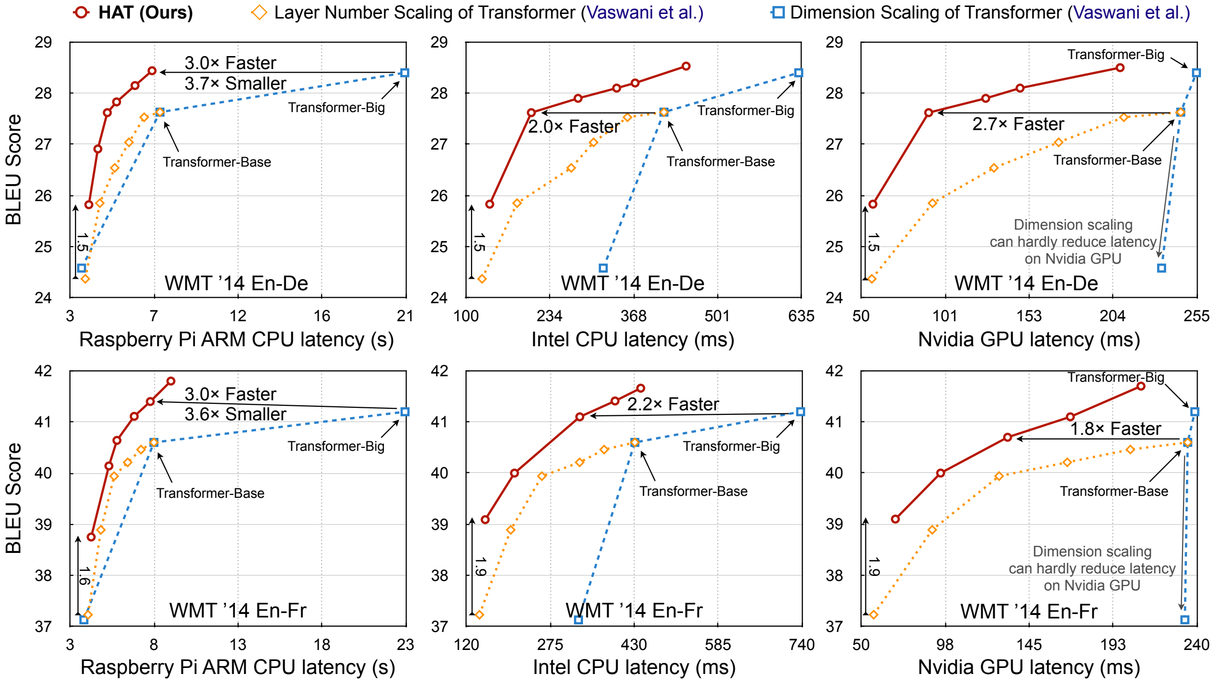


Figure 7: Inference latency and BLEU trade-offs of WMT'14 En-De and En-Fr on three hardware platforms. HAT consistently outperforms the baseline Transformers and achieves up to 3× faster inference speed and 3.7× smaller size over Transformer-Big. Specific latency, BLEU and SacreBLEU numbers are available in Appendix Table 8.

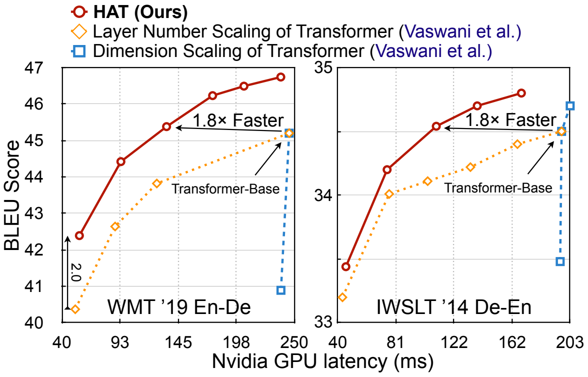


Figure 8: Inference latency and BLEU trade-offs of WMT'19 and IWSLT'14 tasks on Nvidia GPU.

4 Results

4.1 HAT Performance Comparisons

In Figure 7 and Figure 8, we compare HAT with Transformer baselines on four tasks. The embedding dims are 512 and 1024 for the Transformer-Base and Big, respectively. The hidden dims are 4× and 2× of the embedding dim for WMT and IWSLT. The IWSLT models are smaller to prevent overfitting (Wu et al., 2019b). We obtain a series of baseline models with layer number scaling (yellow) and dimension scaling (blue). We set different latency constraints on three hardware to get a series of HAT models. HAT consistently outperforms baselines with a large gap under different latency constraints. On ARM CPU, HAT is 3× faster and 3.7× smaller than Transformer-Big with the same BLEU. On Intel CPU, HAT achieves over 2× speedup. On Nvidia GPU, the blue dash line is nearly *vertical*, indicating that dimension scaling can hardly reduce the latency. In this case, HAT can still find models with low latency and high performance. We show specific latency, BLEU and SacreBLEU (Post, 2018) in Appendix Table 8.

We further compare various aspects of HAT with Transformer (Vaswani et al., 2017) and Evolved Transformer (So et al., 2019) in Table 2. HAT achieves up to 1.6×, 3×, and 3.4× speedup with up to 1.4×, 3.7×, and 4× smaller size than baselines. We report FLOPs for translating a 23-token sentence for IWSLT and 30 for WMT. We show the overall GPU hours for training the SuperTransformer and the searched SubTransformer. We also calculate the cloud computing costs with different modes: "preemptable" is cheaper ($0.74/h) than "on-demand" ($2.48/h) (Strubell et al., 2019). HAT is highly affordable since the total GPU-hour is over 12000× smaller than the Evolved Transformer, and is even smaller than Transformer-Big by virtue of the compact model size.

In Table 3, we compare HAT with other latest models. We scale down all models to have similar BLEU scores with Levenshtein for fair comparisons. We adopt the average iteration time of 2.88 for decoding (Gu et al., 2019), without limiting

		Hardware-Aware	Hetero. Layers	Latency	#Params	GFLOPs	BLEU	GPU Hours	CO_2e (lbs)	Cloud Computing Cost
IWSLT'14 De-En	Transformer	✗	✗	3.3s	32M	1.5	34.5	2	5	\$12 - \$40
	HAT (Ours)	✓	✓	**2.1s**	**23M**	**1.1**	**34.5**	4	9	\$24 - \$80
WMT'14 En-Fr	Transformer	✗	✗	23.2s	176M	10.6	41.2	240	68	\$178 - \$595
	Evolved Trans.	✗	✗	20.9s	175M	10.8	41.3	2,192,000	626,000	\$1.6M - \$5.5M
	HAT (Ours)	✓	✓	**7.8s**	**48M**	**3.4**	**41.4**	216	61	\$159 - \$534
	HAT (Ours)	✓	✓	9.1s	57M	3.9	**41.8**	224	64	\$166 - \$555
WMT'14 En-De	Transformer	✗	✗	20.5s	176M	10.6	28.4	184	52	\$136 - \$456
	Evolved Trans.	✗	✗	7.6s	47M	2.9	28.2	2,192,000	626,000	\$1.6M - \$5.5M
	HAT (Ours)	✓	✓	**6.0s**	**44M**	**2.7**	28.2	184	52	\$136 - \$456
	HAT (Ours)	✓	✓	6.9s	48M	3.0	**28.4**	200	57	\$147 - \$495

Table 2: Comparisons of latency, model size, FLOPs, BLEU and training cost in terms of CO_2 emissions (lbs) and cloud computing cost (USD) for Transformer (Vaswani et al., 2017), Evolved Transformer (So et al., 2019) and HAT. The training cost estimation is adapted from Strubell et al. (2019). The training time is for one NVIDIA V100 GPU, and the latency is measured on the Raspberry Pi ARM CPU. The cloud computing cost is based on AWS. HAT can find low-latency high-BLEU models with low cost.

	Latency	BLEU
Transformer (Vaswani et al., 2017)	4.3s	25.85
Levenshtein (Gu et al., 2019)	6.5s	25.20
Evolved Transformer (So et al., 2019)	3.7s	25.40
Lite Transformer (Wu et al., 2020)	3.4s	25.79
HAT (Ours)	**3.4s**	**25.92**

Table 3: Raspberry Pi ARM CPU latency and BLEU comparisons with different models on WMT'14 En-De. HAT has the lowest latency with the highest BLEU.

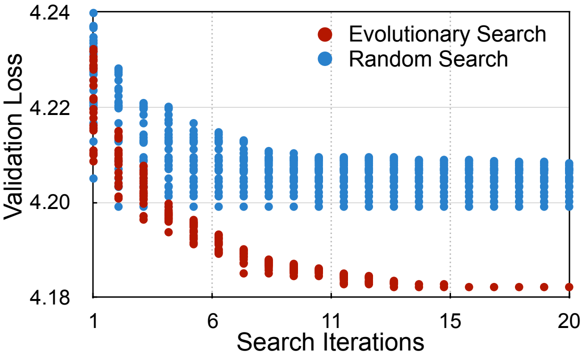


Figure 9: Evolutionary search can find better SubTransformers in the SuperTransformer than random search.

the length of the output sentence (12 tokens after decoding). HAT runs 1.3× faster than Transformer with higher BLEU; 1.9× faster than Levenshtein with 0.7 higher BLEU. Under similar latency, HAT also outperforms Lite Transformer. These results demonstrate HAT's effectiveness in lower latency scenarios. Our framework can also be adopted to speedup those models.

4.2 Analysis

Design Insights. For all HAT WMT models in Figure 7, 10% of all decoder layers attend to three encoder layers, 40% attend to two encoder layers. That demonstrates the necessity of arbitrary encoder-decoder attentions.

	SubTransformer	Latency	#Params	BLEU
WMT'14 En-De	Largest	10.1s	71M	28.1
	Searched HAT	**6.9s**	**48M**	**28.4**
WMT'14 En-Fr	Largest	10.1s	71M	41.4
	Searched HAT	**9.1s**	**57M**	**41.8**

Table 4: The searched HAT compared with the largest SubTransformer in the design space. Larger models do not necessarily have better performance. Our framework can find SubTransformers with lower latency, smaller size, and higher BLEU.

In Appendix Figure 12, we visualize the models specialized for different hardware mentioned in Table 1. We find that the GPU model is *wide but shallow*; the Raspberry Pi model is *deep but thin*. The phenomenon echos with our latency profiling (Figure 2) as GPU latency is insensitive to embedding and hidden dim, but Raspberry Pi is highly sensitive. It guides manual designs: on GPU, we can reduce the layer number and increase dimension to reduce latency and keep high performance.

Ablation Study. HAT achieves higher BLEU with 1.5× lower latency and 1.5× smaller size compared with the largest SubTransformer (Table 4). This suggests that larger models do not always provide better performance, and demonstrates the effectiveness of HAT. We also compare the evolutionary search with random search (Figure 9). Evolutionary search can find models with lower losses than random search.

WMT'14 En-De			WMT'14 En-Fr		
Inherited Val Loss	Inherited BLEU	From-Scratch BLEU	Inherited Val Loss	Inherited BLEU	From-Scratch BLEU
4.71	24.9	25.8	3.92	37.4	38.8
4.40	25.8	27.6	3.71	38.0	40.0
4.07	26.3	28.1	3.48	39.5	41.1
4.02	26.7	28.2	3.46	39.6	41.4
4.01	26.9	28.4	3.45	39.7	41.7

Table 5: The performance of SubTransformers with inherited weights are close to those trained from-scratch, and have the same relative performance order.

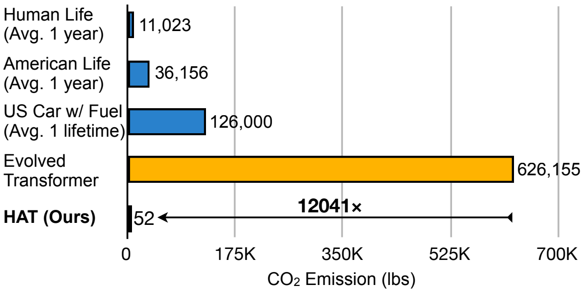


Figure 10: The search cost measured in pounds of CO_2 emission. Our framework for searching HAT reduces the cost by four orders of magnitude than the Evolved Transformer (So et al., 2019).

SubTransformer Performance Proxy. All SubTransformers inside the SuperTransformer are *uniformly sampled* and thus *equally trained*, so the performance order is well-preserved during training. We conduct experiments to show the effectiveness of the SubTransformer performance proxy as in Table 5 and Appendix Figure 11. The BLEUs of SubTransformers with inherited weights and weights trained from-scratch are very close. More importantly, they also have the *same relative performance order*. Therefore, we can rely on the proxy to search high-performance model architecture, significantly reducing the search cost.

Low Search Cost. As shown in Table 2 and Figure 10, the search cost of HAT is 12,041× lower than the Evolved Transformer. Although both are using Evolutionary Search, the key difference is that Evolved Transformer needs to train all *individual models* and sort their *final performance* to pick top ones; on the contrary, HAT trains *all models together* inside SuperTransformer and sorts their *performance proxy* to pick top ones. The superior performance of HAT proves that the performance proxy is accurate enough to find good models.

Finetuning Inherited SubTransformers In section 4.1, we trained each searched SubTransformer from-scratch in order to conduct fair comparisons with baselines. In practice, we can also directly finetune the SubTransformers with the inherited weights from the SuperTransformer to further reduce the training cost. With 10K finetuning steps (1/4 of from-scratch training), the inherited SubTransformers can achieve similar or better performance than trained from-scratch ones (Table 6). In this way, the training cost for a model under a new hardware constraint can be further reduced by 4×, since the SuperTransformer training cost is amortizable among all searched models.

Task	From-Scratch 40K	Inherit-Finetune 10K
WMT'14 En-Fr	41.5 40.0	41.7 40.2
WMT'14 En-De	28.0 27.5	28.0 27.4

Table 6: The SubTransformer inherited from the SuperTransformer can achieve similar or better performance than the same SubTransformer trained from-scratch. Training steps are saved by 4×.

Quantization Friendly. HAT is orthogonal to other model compression techniques such as quantization. We apply K-means quantization to HAT and further reduce the model size. We initialize centroids uniformly in the range of [min, max] of each weight matrix and run at most 300 iterations for each of them. Even without any finetuning, 4-bit quantization can reduce the model size by 25× with negligible BLEU loss compared to the Transformer-Big baseline (Table 7). Interestingly, the 8-bit model even has 0.1 higher BLEU than the full precision model, indicating the robustness of searched HAT. Compared with the Transformer-Base 4-bit quantization baseline, which has 24MB model size and 38.9 BLEU score, HAT has 2.2 higher BLEU with similar model size.

Knowledge Distillation Friendly. HAT is also orthogonal to knowledge distillation (KD) because HAT focuses on searching for an efficient architecture while KD focuses on better training a given architecture. We combine KD with HAT by distilling token-level knowledge (top-5 soft labels) from a high-performance SubTransformer to a low-performance SubTransformer on WMT'14 En-De task. The teacher model has a BLEU of 28.5 and 49M parameters; the student model has 30M parameters. KD can improve the BLEU of the student model from 25.8 to 26.1.

	BLEU	Model Size	Reduction
Transformer Float32	41.2	705MB	–
HAT Float32	41.8	227MB	3×
HAT 8 bits	**41.9**	**57MB**	**12×**
HAT 4 bits	41.1	**28MB**	**25×**

Table 7: K-means quantization of HAT models on WMT'14 En-Fr. 4-bit quantization reduces the model size by 25× with only 0.1 BLEU loss compared with the transformer baseline. 8-bit quantization even has 0.1 higher BLEU than its full precision version.

5 Related Work

Transformer. Transformer (Vaswani et al., 2017) has prevailed in sequence modeling (Ng et al., 2019; Junczys-Dowmunt, 2018). By stacking identical blocks, the model obtains a large capacity but incurs high latency. Recently, a research trend is to modify the Transformer to improve the performance (Chen et al., 2018; Wu et al., 2019b; Sukhbaatar et al., 2019; Wang et al., 2019). Among them, Wu et al. (2019b) introduced a convolution-based module to replace the attention; Wang et al. (2019) proposed to train deep Transformers by propagating multiple layers together in the encoder. Zhang et al. (2018) and Kim et al. (2019) also proposed AAN and SSRU to replace the attention mechanism. HAT is orthogonal to them and can be combined to search for efficient architecture with those new modules. Another trend is to apply non- or partially-autoregressive models to cut down the iteration number for decoding (Gu et al., 2019; Akoury et al., 2019; Wei et al., 2019; Gu et al., 2018). Although reducing latency, they sometimes suffer from low performance. Bapna et al. (2018) explored using learned linear combinations of encoder outputs as decoder inputs, while HAT concatenates the outputs without linear combinations, thus better preserving the low-level information. Wu et al. (2020) investigated mobile settings for NLP tasks and proposed a multi-branch Lite Transformer. However, it relied on FLOPs for efficient model design, which is an inaccurate proxy for hardware latency (Figure 2). There are also works (Kim and Rush, 2016; Junczys-Dowmunt et al., 2018; Kim et al., 2019; Yan et al., 2020) using Knowledge Distillation (KD) to obtain small student models. Our method is orthogonal to KD and can be combined with it to improve the efficiency further. There are also hardware accelerators (Ham et al., 2020; Zhang et al., 2020) for attention and fully-connected layers in the Transformer to achieve efficient processing.

Neural Architecture Search. In the computer vision community, there has been an increasing interest in automating efficient model design with Neural Architecture Search (NAS) (Zoph and Le, 2017; Zoph et al., 2018; Pham et al., 2018; He et al., 2018). Some of them applied black-box optimization methods such as evolutionary search (Wang et al., 2020b) and reinforcement learning (Cai et al., 2019b; He et al., 2018; Wang et al., 2018, 2020a; Mao et al., 2019); Some leveraged backpropagation with differentiable architecture search (Liu et al., 2019). Some of them also involved hardware constraints into optimizations such as MNasNet (Tan et al., 2019), ProxylessNAS (Cai et al., 2019b), FBNet (Wu et al., 2019a) and APQ (Wang et al., 2020b). To reduce the high design cost of NAS, supernet based methods (Pham et al., 2018; Bender et al., 2018; Guo et al., 2019) apply a proxy for sub-network performance and adopt search algorithms to find good sub-networks. For NLP tasks, the benefits of the architecture search have not been fully investigated. Recently, So et al. (2019) proposed the Evolved Transformer to search for architectures under model size constraints and surpassed the original Transformer baselines. However, it suffered from very high search costs (*250 GPU years*), making it unaffordable to search specialized models for various hardware and tasks. In addition, hardware latency feedback was not taken into account for better case-by-case specializations. Since different hardware has distinct architecture and features (Cong et al., 2018), feedback from hardware is critical for efficient NLP.

6 Conclusion

We propose Hardware-Aware Transformers (HAT) framework to solve the challenge of efficient deployments of Transformer models on various hardware platforms. We conduct hardware-aware neural architecture search in an ample design space with an efficient weight-shared SuperTransformer, consuming four orders of magnitude less cost than the prior Evolved Transformer, and discover high-performance low-latency models. We hope HAT can open up an avenue towards efficient Transformer deployments for real-world applications.

Acknowledgment

We thank NSF Career Award #1943349, MIT-IBM Watson AI Lab, Semi-conductor Research Corporation (SRC), Intel, and Facebook for supporting this research.

References

Nader Akoury, Kalpesh Krishna, and Mohit Iyyer. 2019. Syntactically supervised transformers for faster neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1269–1281, Florence, Italy. Association for Computational Linguistics.

Ankur Bapna, Mia Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3028–3033, Brussels, Belgium. Association for Computational Linguistics.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmssan, Stockholm Sweden. PMLR.

Han Cai, Chuang Gan, and Song Han. 2019a. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Han Cai, Ligeng Zhu, and Song Han. 2019b. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86, Melbourne, Australia. Association for Computational Linguistics.

Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. 2018. Understanding performance differences of fpgas and gpus. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 93–96. IEEE.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252, International Convention Centre, Sydney, Australia. PMLR.

Édouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for GPUs. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1302–1310, International Convention Centre, Sydney, Australia. PMLR.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein Transformer. *arXiv*.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2019. Single path one-shot neural architecture search with uniform sampling.

Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W Lee, et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 328–341. IEEE.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications.

Marcin Junczys-Dowmunt. 2018. Microsoft's submission to the wmt2018 news translation task: How i learned to stop worrying and love the data. *arXiv preprint arXiv:1809.00196*.

Marcin Junczys-Dowmunt. 2019. Microsoft translator at wmt 2019: Towards large-scale document-level neural machine translation. *arXiv preprint arXiv:1907.06170*.

Marcin Junczys-Dowmunt, Kenneth Heafield, Hieu Hoang, Roman Grundkiewicz, and Anthony Aue. 2018. Marian: Cost-effective high-quality neural machine translation in C++. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 129–135, Melbourne, Australia. Association for Computational Linguistics.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong. Association for Computational Linguistics.

Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*.

Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, et al. 2019. Park: An open platform for learning-augmented computer systems. In *Advances in Neural Information Processing Systems*, pages 2490–2502.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 news translation task submission. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 314–319, Florence, Italy. Association for Computational Linguistics.

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.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmssan, Stockholm Sweden. PMLR.

Matt Post. 2018. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*.

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, Long Beach, California, USA. PMLR.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 331–335, Florence, Italy. Association for Computational Linguistics.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Conference on Neural Information Processing Systems*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020a. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *ACM/IEEE 57th Design Automation Conference (DAC)*.

Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. 2018. Learning to design circuits. In *NeurIPS 2018 Machine Learning for Systems Workshop*.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020b. Apq: Joint search for network architecture, pruning and quantization policy. In *Conference on Computer Vision and Pattern Recognition*.

Bingzhen Wei, Mingxuan Wang, Hao Zhou, Junyang Lin, and Xu Sun. 2019. Imitation learning for non-autoregressive neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1304–1312, Florence, Italy. Association for Computational Linguistics.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019a. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019b. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention. In *International Conference on Learning Representations*.

Zhongxia Yan, Hanrui Wang, Demi Guo, and Song Han. 2020. Micronet for efficient language modeling. *Journal of Machine Learning Research*.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating neural transformer via an average attention network. *arXiv preprint arXiv:1805.00631*.

Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. 2020. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 261–274. IEEE.

Barret Zoph and Quoc V Le. 2017. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

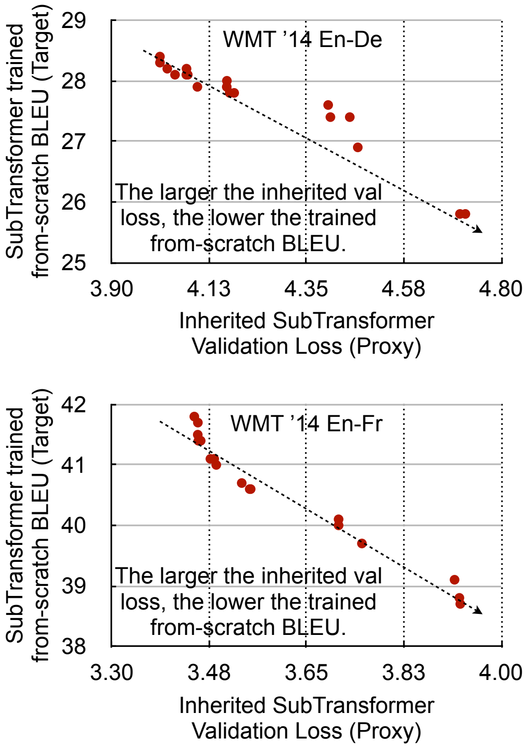


Figure 11: The validation loss of SubTransformers is a good performance proxy for BLEU of from-scratch trained SubTransformers. The larger the validation loss, the lower the BLEU score.

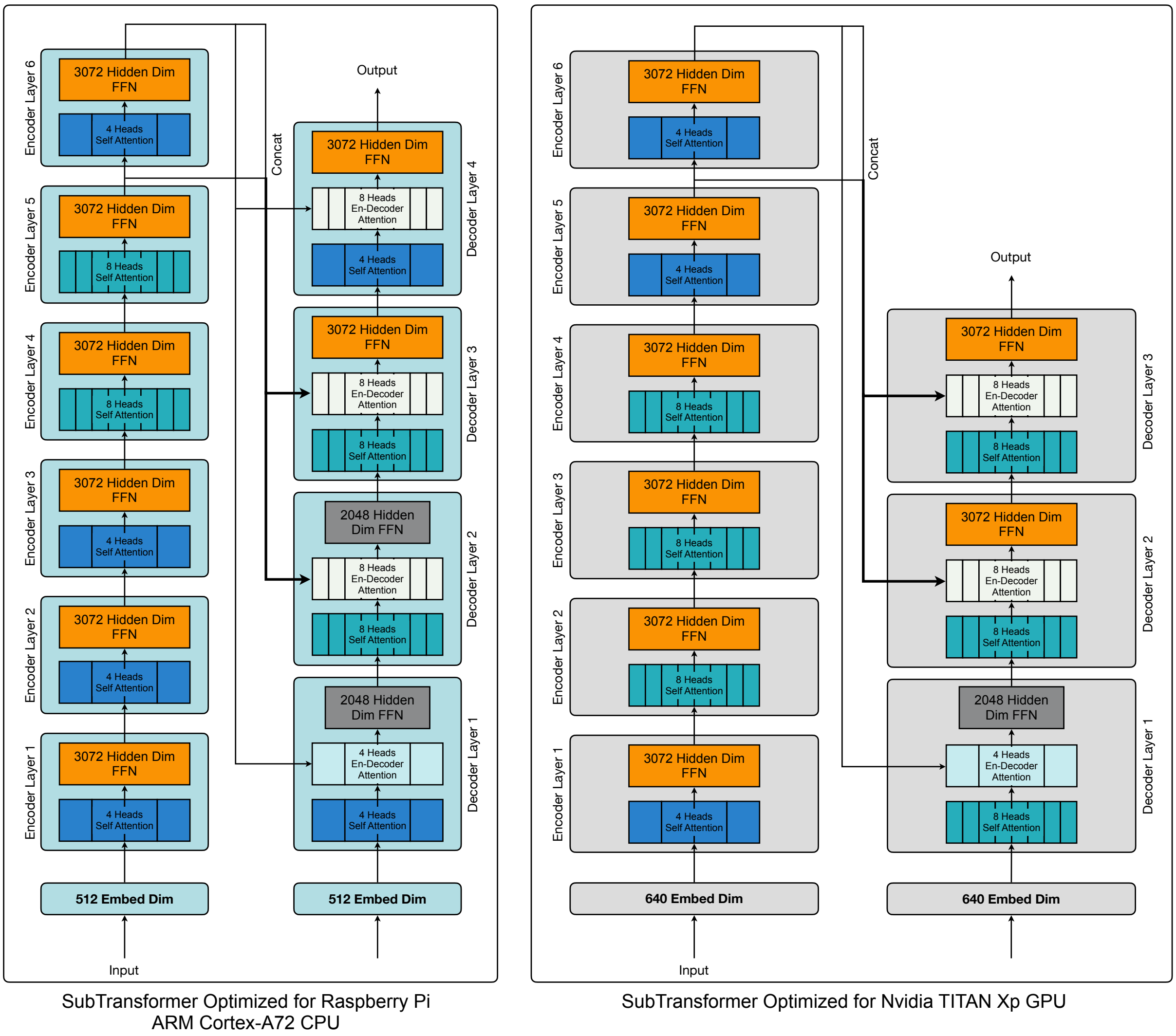
7 Appendix for "HAT: Hardware-Aware Transformers for Efficient Natural Language Processing"

7.1 SubTransformer Performance Proxy

In Figure 11, we show the relationship between the validation loss of SubTransformers directly inherited from the SuperTransformer, and the BLEU score of the SubTransformers trained from-scratch. We can observe that the larger the validation loss, the lower the BLEU score. Therefore the validation loss can be a good performance proxy.

7.2 Visualizations of Searched Models on WMT'14 En-De Task

We show the HAT models searched for Raspberry Pi ARM Cortex-A72 CPU and Nvidia TITAN Xp GPU in Figure 12. The searched model for Raspberry Pi is deep and thin, while that for GPU is shallow and wide. The BLEU scores of the two models are similar: 28.10 for Raspberry Pi CPU, and 28.15 for Nvidia GPU.



SubTransformer Optimized for Raspberry Pi
ARM Cortex-A72 CPU

SubTransformer Optimized for Nvidia TITAN Xp GPU

Figure 12: SubTransformers optimized for Raspberry Pi ARM CPU and Nvidia GPU on WMT'14 En-De task are different. The CPU model has BLEU 28.10, and GPU model has BLEU 28.15.

7.3 Latency, BLEU and SacreBLEU of searched HAT models.

In Table 8, we show the specific latency numbers, BLEU and SacreBLEU (Post, 2018) scores for searched HAT models in Figure 7 and Figure 8.

Task	Hardware	Latency	BLEU	SacreBLEU
WMT'14 En-De	Raspberry Pi ARM Cortex-A72 CPU	3.5s	25.8	25.6
		4.0s	26.9	26.6
		4.5s	27.6	27.1
		5.0s	27.8	27.2
		6.0s	28.2	27.6
		6.9s	28.4	27.8
	Intel Xeon E5-2640 CPU	137.9ms	25.8	25.6
		204.2ms	27.6	27.1
		278.7ms	27.9	27.3
		340.2ms	28.1	27.5
		369.6ms	28.2	27.6
		450.9ms	28.5	27.9
	Nvidia TITAN Xp GPU	57.1ms	25.8	25.6
		91.2ms	27.6	27.1
		126.0ms	27.9	27.3
		146.7ms	28.1	27.5
		208.1ms	28.5	27.8
WMT'14 En-Fr	Raspberry Pi ARM Cortex-A72 CPU	4.3s	38.8	36.0
		5.3s	40.1	37.3
		5.8s	40.6	37.8
		6.9s	41.1	38.3
		7.8s	41.4	38.5
		9.1s	41.8	38.9
	Intel Xeon E5-2640 CPU	154.7ms	39.1	36.3
		208.8ms	40.0	37.2
		329.4ms	41.1	38.2
		394.5ms	41.4	38.5
		442.0ms	41.7	38.8
	Nvidia TITAN Xp GPU	69.3ms	39.1	36.3
		94.9ms	40.0	37.2
		132.9ms	40.7	37.8
		168.3ms	41.1	38.3
		208.3ms	41.7	38.8
IWSLT'14 En-De	Nvidia TITAN Xp GPU	45.6ms	33.4	32.5
		74.5ms	34.2	33.3
		109.0ms	34.5	33.6
		137.8ms	34.7	33.8
		168.8ms	34.8	33.9
WMT'19 En-De	Nvidia TITAN Xp GPU	55.7ms	42.4	41.9
		93.2ms	44.4	43.9
		134.5ms	45.4	44.7
		176.1ms	46.2	45.6
		204.5ms	46.5	45.7
		237.8ms	46.7	46.0

Table 8: Specific latency numbers, BLEU and SacreBLEU scores for searched HAT models in Figure 7 and Figure 8.