Efficient Model Configuration for Transformers: Improving Translation Quality with Reduced Parameters and Comparable Inference Speed

Anonymous EACL submission

Abstract

In recent developments, Transformers have emerged as the leading performers in a range of natural language processing tasks, including the challenging domain of machine translation. Nonetheless, traditional Transformers have encountered a significant obstacle in the form of high inference costs. This paper addresses this issue by investigating the influence of various model hyperparameters on the architecture of Transformers, focusing on their impact on both translation quality and inference speed. Our research findings lead us to propose an optimized model configuration, which surpasses standard efficient vanilla Transformers by achieving a 1-point increase in BLEU score, while utilizing fewer parameters and maintaining identical inference speed when running on a CPU.

1 Introduction

Transformers (Vaswani et al., 2017) have been widely used in Natural Language Processing (NLP). The architecture was first introduced for Machine Translation (MT) and has been adopted for other sequence-to-sequence, classification, and generation tasks. Compared to the previous state-of-the-art, i.e., LSTM (Hochreiter and Schmidhuber, 1997), Transformers are faster to train as they are parallelizable in training time, but their main drawback is their increased inference time. Unlike Recurrent Neural Networks (RNN), decoding in transformers has time complexity of $O(n^2)$. This problem has led many researchers to find efficient transformer alternatives. Tay et al. (2020) provides a comprehensive survey on different approaches for increasing the efficiency of Transformer architecture.

The most focused part of research on Transformer efficiency has been on alternative types of self-attention in order to reduce its time-complexity, which has shown great speed-up in longer texts (Beltagy et al., 2020; Wang et al., 2020b). There also have been proposed methods to reduce latency and increase speed for tasks with shorter sequences. For instance, *Average Attention Network* (Zhang et al., 2018) is an alternative attention mechanism that increases speed up to 1.30x without quality loss, with the caveat of increased parameter count.

The aforementioned methods have a common problem which is the lack of ecosystem support. Vanilla Transformers have had many optimizations over the years since it was introduced (Gschwind et al., 2022). There are also efficient tools for their efficient inference which do not support alternative architectures (Klein et al., 2020; Junczys-Dowmunt et al., 2018). This creates an incentive to explore various network parameters to find better vanilla Transformer configurations.

Neural Architecture Search (NAS), initially popularized within the context of Convolutional Neural Networks (CNNs), has been extended to Transformers and gained popularity for the exploration of optimal architecture configurations. Literature on Transformer NAS has shown promising results for finding smaller models with comparable quality. This paper primarily focuses on finding the effect of hyperparameters on model latency and its quality for MT. Diverging from earlier investigations, this study adopts an analytical approach, opting to train models from the ground up rather than employing a super-network as a proxy method, which has been the primary mode of exploration in prior work (Pham et al., 2018). While earlier architecture search efforts have tended to prioritize larger models, this research uniquely emphasizes the significance of smaller models that can be effectively deployed on edge computing devices.

In this work, first, we discuss the related work in section 2, then, we show that in order to compare the effect of different hyperparameters, it is not needed to train until convergence, as the first epoch can be a good proxy for relative performance. Next, we train a number of models which we believe are good representatives of the entire search space. Section 3 discusses the search space and experiment setup in-depth. Section 4 shows our findings and the effect of parameter size and hyperparameter selection on model latency and quality. We train a model on WMT'14 En-De dataset to verify our results. Finally, Section 5 and Section 6, concludes the paper and presents future work, respectively.

To the best of our knowledge, no analytical work has been done on the effect of hyperparameters for NMT. Our contributions are as follows:

- Analyzed the effect of hyperparameters on quality, latency, and throughput.
- Analyzed the effect of hyperparameters on quantization degradation.
- Presented design insight for creating fast and deployable models.
- Presented a Pareto frontier for different hardware and needs (CPU/GPU).
- Presented models trained on WMT'14 En-De dataset to verify our findings.

2 Related Work

Because Transformers (Vaswani et al., 2017) has gained state-of-the-art at many NLP tasks, researchers have tried to find smaller models while retaining the same quality. The Evolved Transformer (So et al., 2019) was one of the first endeavors to find better architecture configurations for Transformers. The paper uses differentiable NAS with evolution search for finding the best model given certain constraints. The focus was to find models with the best performance rather than search or inference efficiency. They have found a range of models for WMT'14 En-De dataset from 7M to 221M parameters with slightly better performance compared to vanilla Transformer models of the same size.

Wang et al., 2020a has used an evolutionary algorithm with direct feedback from different hardware and found different hardware prioritize different models for increased latency. The paper uses a supernetwork in order to reduce training time, as one single supernetwork is created to include all search space and different configurations are created by weight sharing. The homogeneity convention of different layers has been left out, and models can have different feed-forward network sizes and the number of heads in every layer. The search space for this paper was [512, 640] for embedding dim, [1024, 2048, 3072] for hidden dim, [4, 8] for the head number in all attention modules, and [1, 2, 3, 4, 5, 6] for the number of decoder layers. They found GPUs prefer wide and shallow networks over deep ones, while CPUs have lower latency on narrow and rather deep networks.

Kasai et al., 2020, in the context of comparing Auto-Regressive (AR) and Non-Auto-Regressive (NAR) Architectures, has shown decreasing the number of decoder layers has minimal impact on translation quality while this degradation can be remedied by increasing the size of encoder layers. Bérard et al., 2021 have shown this phenomenon also happens in the multilingual setting, implying that decoders are the speed bottleneck of transformer inference.

Javaheripi et al., 2022 applies NAS on generative transformers. It uses parameter size as a proxy for model quality, and training many models verified a good correlation between the proxy and the ground truth. The paper uses evolutionary search to find more efficient models. It also presents a Pareto frontier for different hardware, finding model configurations that are faster while having better performance than the baseline.

Chitty-Venkata et al., 2022 is a comprehensive survey on NAS for transformers, interested readers can refer to this paper for an in-depth survey of NAS for different tasks.

WMT's translation efficiency shared task (Heafield et al., 2020; Heafield et al., 2021) had submissions that used decreasing the number of decoder layers as the main component of increasing speed while retaining the same quality, although other methods of optimization are widely used in these submissions. Klein et al., 2020 uses vanilla Transformers showing that using a larger feed-forward network (FFN) can have an impact on translation quality while having little effect on speed.

Our work is mainly focused on gaining insight into the model design rather than finding optimized model(s) while analyzing the effect of beam search size and quantization on different architectures and finding different models for different needs (e.g., servers have different priorities than edge devices). To our knowledge, this kind of analysis has not been done on combinations of these parameters.

3 Experiment Setup

This section discusses the experiment setup and the reason behind it. To begin, we demonstrate the viability of training a neural machine translation (NMT) model for a single epoch as a reliable indicator of its quality when fully trained. Next, we delve into the details of the explored search space for the models under investigation. Finally, we provide an overview of the experimental setup, including the environment and tools employed for conducting the experiments.

3.1 Dataset and Proxy

We randomly selected 10 million samples out of a total of 87 million parallel data from the WMT'22 dataset for En-De translation efficiency shared task[1] as our training dataset with the provided SentencePiece (Kudo and Richardson, 2018) model as our tokenizer. We opt for a partial training dataset to efficiently use computational resources and direct them towards space exploration and to mitigate potential overfitting concerns. Furthermore, we do not pursue training to convergence based on preliminary experiments, which indicated that such a strategy is unnecessary for model comparison. Figure 1 shows the training trend for 18 different Transformer configuration representing our complete search space. The results demonstrated that the ranking of models in terms of BLEU score remains stable as training progresses, with fluctuations below 0.5 BLEU deemed statistically insignificant. Employing t-statistics, we have 97.5% CI [3.12, 3.49] the BLEU score increase, suggesting that it is improbable that we would get any significant difference change (> 0.5 BLEU) between models after training for four more epochs. Consequently, we conduct remaining experiments with only one epoch according to these findings.

3.2 Search Space

The search space is [128, 192, 256, 384, 512] for embedding dimensions, and [512, 1024, 2048, 3072, 4096] for hidden dimensions. Number of heads is relative to the embedding dimension and is kept at $h = embedding/64$. We also use [1, 3, 6] for the number of decoder layers. Literature shows that time spent during the decoding phase is 20–30 times that of the encoding phase (Zhang et al., 2018; Wang et al., 2020a; Bérard et al., 2021).

[1] https://data.statmt.org/wmt22/efficiency-task/data/clean/

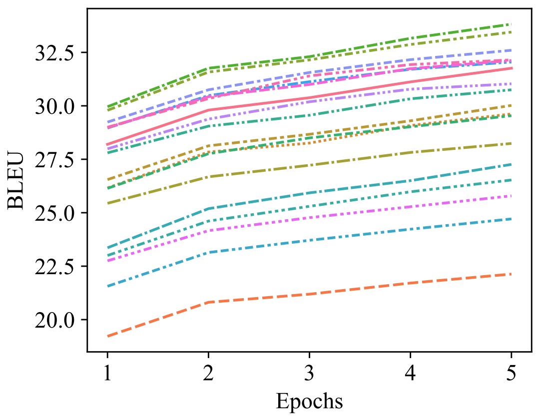

Figure 1: BLEU score on combined En-De validation datasets taken from WMT'14 to WMT'18 for 18 Transformer configurations. configurations.

Hence, following Wang et al., 2020a, we focus on the decoder configuration and consider 6 layers for encoder layers for all our experiments.

3.3 Training and Inference

Fairseq (Ott et al., 2019) is used to train our models. Training parameters are kept the same for all models. We use a batch size of 4096 tokens, the optimizer is Adam ($\beta_1 = 0.9$ and $\beta_2 = 0.98$) with a learning rate, dropout, and weight decay of $5\text{e}{-4}$, 0.1, and $1\text{e}{-4}$, respectively. We use CTranslate2[2] (Klein et al., 2020) to perform NMT inference. CTranslate2 is an open-source Transformer inference engine, which supports various hardware platforms while using appropriate instructions to maximize the speed. Our initial experiments show that Fairseq does not fully utilize the CPU during the inference time. During inference time, we use batch and beam sizes of [1, 16, 32, 64] and [1, 2, 3, 4, 5], respectively, while the length penalty is set to 0.6. Moreover, NVIDIA GTX 1080 and Intel Xeon E5-2620 (limited to a single core) are employed as our GPU and CPU for inference testing. The CPU in use supports `AVX2` instruction set which, unlike `AVX512`, is more consistently supported and is available in all of newer Intel CPUs. For CPU inference, we also test the effect of post-training quantization on speed and quality. More concretely, [`int8`, `int16`, `fp32`] are considered as quantization precision.

[2] https://github.com/OpenNMT/CTranslate2

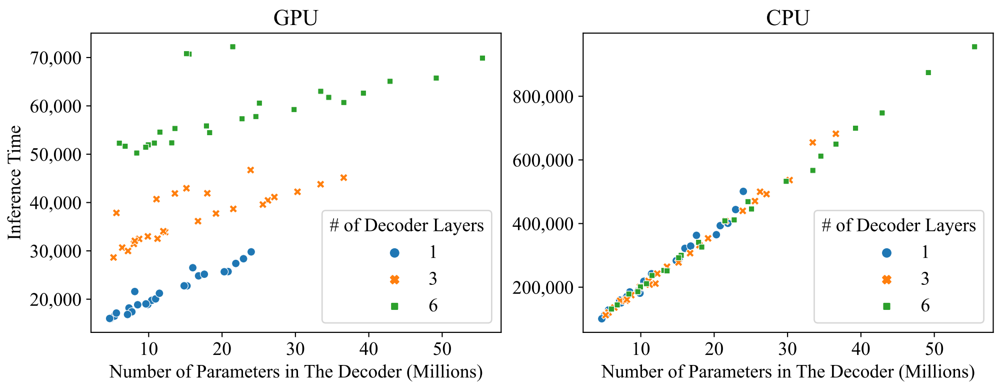

Figure 2: Relation of the number of parameters in the decoder (including embedding layer) on inference time of translation. WMT'22 test set (1k sentences) is employed.

4 Findings

This section includes our findings on the experiments, discussing the effect of different parameters on inference time, quality, and model size.

4.1 Parameters and Inference Speed

CPUs and GPUs have different behaviors regarding the effect of the number of parameters on inference time. Figure 2 shows inference time vs. the number of parameters in the decoder in the latency setting (i.e., batch size of 1). It is seen that inference time on the CPU scales linearly with the number of parameters, while on the GPU, the number of layers is a more important feature. On The GPU, the number of parameters has a less pronounced effect on inference time compared to the CPU. This phenomenon is caused by the parallelization of the entire layer in GPUs, while CPUs have limited parallelization. Regardless of the configuration, the number of parameters determines the inference speed on the CPU.

Figure 3 shows the effect of the number of parameters in the decoder relative to the BLEU score. As can be seen, the Pareto frontier is dominated mostly by models with 3 decoder layers.

Klein et al., 2020 demonstrated that the size of FFN does not affect latency in GPU inference. However, this observation holds true exclusively for GPU inference, as in the case of CPU inference, FFN size exhibits a direct correlation with the number of parameters, which in turn influences latency. In line with the conclusions of another study by Kasai et al., 2020, which suggests that shallow decoders are optimal for fast inference, this assertion predominantly holds for GPU inference, while CPU-based inference stands to gain advantages from deeper decoder architectures.

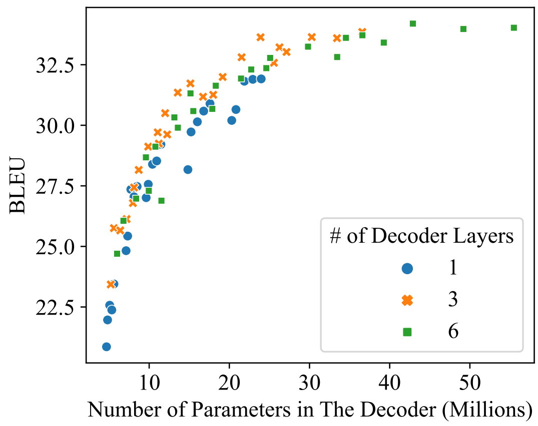

Figure 3: Number of parameters vs. BLEU score.

4.2 Quantization and Batch Size

Running all models with different quantization methods shows that int8 quantization increases the speed 95% CI (1.97, 2.12) times. while having an average BLEU difference of 95% CI (-0.2, -0.1). These numbers are lower for 16-bit integer quantization, being 95% CI (1.19, 1.26) and 95% CI (-0.03, 0.00), respectively. The experiments also show that the embedding size has a negative relation with the quality degradation effect of quantization. `int8` quantization also has a more pronounced effect when the decoder is shallow, with an average speedup of 95% CI (2.07, 2.39) for decoders with 1 layer vs. 95% CI (1.80, 1.96) for decoders with 6 layers. Based on the above discussion, the Pareto frontier is dominated by models with `int8` quantization.

Batching on the CPU can have up to 8x speedup on a single core, with a mean of 7x, but this speedup

Model Name	Encoder	Decoder	FFN Size	Emb. Size	Heads	# of Parameters
A	6	1	2048	512	8	58.4M
B	6	3	2048	384	6	46.6M
C	6	6	3072	256	4	42.9M

Table 1: Models trained on WMT'14 En-De task.

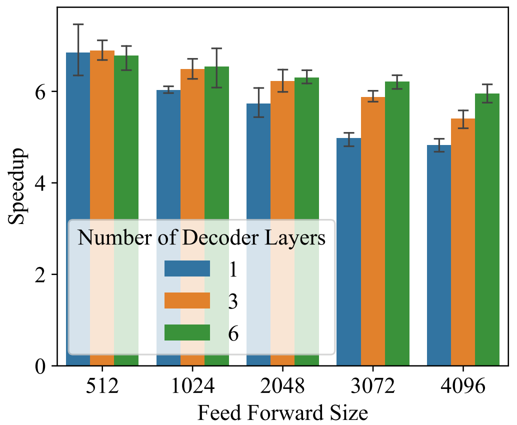

Figure 4: Speedup of using a batch size of 64 compared to 1 on the CPU. Bars are 95% CI.

is bottlenecked by feed-forward network (FFN) size. As Figure 4 shows, increasing the size of the FFN has a decreasing impact on speed gain from batching, it can be seen that fewer decoder layers lead to a steeper decrease in speedup.

The relationship between quantization and batching speed on CPU performance is not independent.

Remarkably, it is noteworthy that when the batch size is configured to 64, both int8 and int16 quantization precision yield identical speeds to the default `fp32` precision. For visualization of this phenomenon, refer to Figure 8 in Appendix A. This intriguing finding suggests that the advantages of `int8` quantization may be most pronounced when the batch size is restricted to just one.

The utilization of batching can further enhance the speedup achieved by GPUs, with a maximum improvement of up to 32x and an average improvement of 19.35x. The number of decoder layers significantly influences the speedup gains. For models employing a 6-layer decoder, the average speedup reaches 23.0x. In contrast, models with fewer decoder layers exhibit lower speedup gains, with an average speedup of 14.5x observed for models utilizing a single decoder layer.

4.3 Beam Search

Our experiments show that there is no significant impact on the BLEU score by quantization.

However, it has been observed that `int8` quantization mitigates the abrupt decrease in speed encountered when increasing the beam size from 1 to 2. It is worth mentioning that models utilizing `int8` quantization experience a continued slowdown as the beam size increases, whereas this phenomenon is not observed with fp32. As illustrated in Figure 5, the utilization of `int8` quantization also counteracts the impact of embedding size on the slowdown effect.

4.4 Verification

We use WMT'14 En-De dataset (4.5 million parallel sentences) provided by Hugging Face[3] to verify our findings and explore cross-dataset applications of our best models.

We take three models with similar number of parameters in the decoder (i.e., 21 million) and train them for 50 epochs (i.e., 30k steps). Model configurations can be seen in Table 1. These models use SentencePiece (32k vocab size) without prior tokenization. We evaluate the performance of three models on the WMT'14 test set, considering both `int8` quantization and non-quantized scenarios. Beam sizes of 1 and 5 are utilized, with a length penalty of 0.6 maintained across all experiments. Results are presented in Table 2. Notably, Model A demonstrates the highest speed on the GPU, while all models exhibit similar performance on the CPU. Model B also exhibits a slightly lower total parameter count. These findings align with our earlier observations, which position models with 3 decoder layers at the Pareto frontier. Surprisingly, Model C, with even fewer total parameters than that of Model B, outperforms Model A.

It is important to mention that knowledge distillation, a technique shown to enhance quality and eliminate the need for beam search (Kim and Rush, 2016), was not applied to these models. All experiments were conducted under the same inference

[3] `https://huggingface.co/datasets/wmt14`

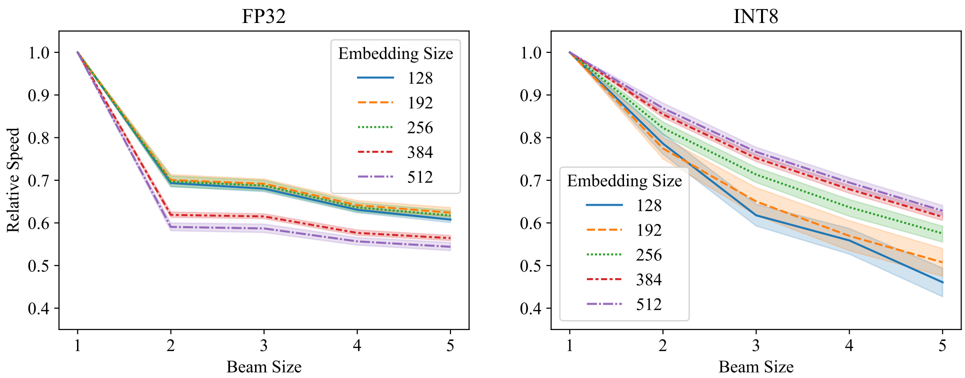

Figure 5: Effect of beam size on speed, relative to greedy search in the latency setting. The area around the lines indicate 95% CI.

conditions as described in Section 3.

Model	Time: CPU	GPU	BLEU
A + `fp32`	652.51	**45.84**	23.72
+ `int8`	315.27	-	23.65
+ beam	1197.16	63.52	25.13
+ both	520.19	-	25.17
B + `fp32`	616.35	60.24	24.84
+ `int8`	**309.37**	-	24.69
+ beam	1087.58	91.52	**26.00**
+ both	541.59	-	25.80
C + `fp32`	640.65	89.64	24.76
+ `int8`	328.92	-	24.62
+ beam	1060.34	130.42	25.77
+ both	585.85	-	25.48

Table 2: Inference results for WMT'14 En-De test set, time is in seconds for 3k test sentences in latency mode. BLEU scores are computed with sacrebleu. Beam search size used is 5.

5 Conclusion

In this paper, we analyzed the effect of different architectural parameters on model quality and inference speed on both CPU and GPU. We also examined the effect of quantization on latency and batched settings. In the end, we trained three models to verify our results and showed that with the same inference speed and slightly fewer parameters, it is possible to reach better translation quality using a better architecture configuration than widely used ones.

6 Future Work

This paper is mostly focused on the effect of decoder configuration on speed and quality. Increasing the number of layers in the encoder has been shown to increase model quality with minimal impact on inference speed. Finding the effect of this technique on models with different decoder configurations can be an extension of this research. Exploring the effect of knowledge distillation on these architectures is also another avenue to explore.

Limitations

Experiments in this paper were done on a single English-to-German translation direction. Findings of this paper may not be attributed to other language pairs, which are not from the same family or are distant pairs. More concretely, we expect English-to-German to have similar behavior to other close language pairs such as English-to-French, but these findings may not be extended to language pairs like English-Arabic or English-Hindi. The aim of this research was to compare different architectures, for this reason, techniques which have an orthogonal effect on quality (e.g., using Moses tokenizer before SentencePiece) were not explored.

Ethics Statement

All experiments in this work were conducted using public datasets. In the course of our experiments, CO_2 emissions were a part of our concerns and we tried our best to keep it as low as possible. ChatGPT was used to post-edit part of the paper.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *ArXiv*, abs/2004.05150.

Alexandre Bérard, Dain Lee, Stéphane Clinchant, Kweonwoo Jung, and Vassilina Nikoulina. 2021. Efficient inference for multilingual neural machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8563–8583.

Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. 2022. Neural architecture search for transformers: A survey. *IEEE Access*, 10:108374–108412.

Michael Gschwind, Eric Han, Scott Wolchok, Rui Zhu, and Christian Puhrsch. 2022. A BetterTransformer for Fast Transformer Inference.

Kenneth Heafield, Hiroaki Hayashi, Yusuke Oda, Ioannis Konstas, Andrew M. Finch, Graham Neubig, Xian Li, and Alexandra Birch. 2020. Findings of the fourth workshop on neural generation and translation. In *Workshop on Neural Generation and Translation*.

Kenneth Heafield, Qianqian Zhu, and Roman Grundkiewicz. 2021. Findings of the wmt 2021 shared task on efficient translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 639–651.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

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

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André FT Martins, et al. 2018. Marian: Fast neural machine translation in c+. *ACL 2018*, page 116.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *CoRR*, abs/2006.10369.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327.

Guillaume Klein, Dakun Zhang, Clément Chouteau, Josep Maria Crego, and Jean Senellart. 2020. Efficient and high-quality neural machine translation with opennmt. In *Workshop on Neural Generation and Translation*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR*, abs/1808.06226.

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. *CoRR*, abs/1904.01038.

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

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International conference on machine learning*, pages 5877–5886. PMLR.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *ACM Computing Surveys*, 55:1 – 28.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating neural transformer via an average attention network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1789–1798.

A Appendix: Charts and Figures

Figure 6 Shows the relation between the embedding size of models and BLEU score difference from `int8` quantization, lower embedding sizes are affected more negatively than models with higher embedding size.

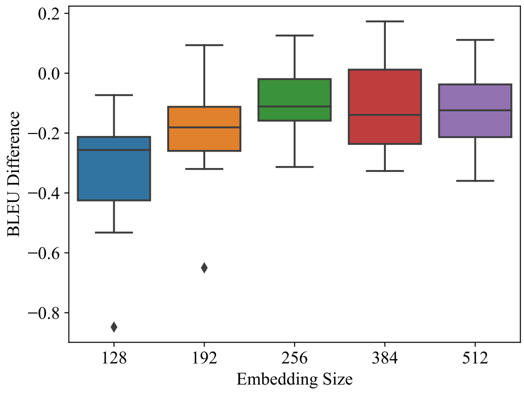

Figure 6: Effect of quantization on quality loss for different embedding sizes.

Figure 7 shows that in very small models, `int8` quantization has a more pronounced effect on speed.

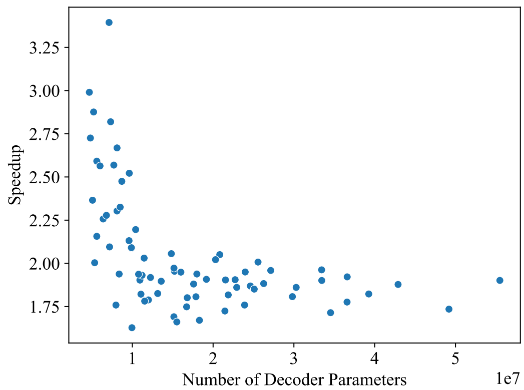

Figure 7: Relation of number of parameters in the decoder (including embedding layer) on speedup gained from `int8` quantization.

Figure 8 Shows that the effect of speedup gained from quantization diminishes with an increase in batch size.

Figure 9 Shows the relation between the number of parameters in the decoder and its effect on speedup gained from batching. Although a lower number of parameters leads to higher speedup gain, the number of decoder layers is another factor affecting speedup gain.

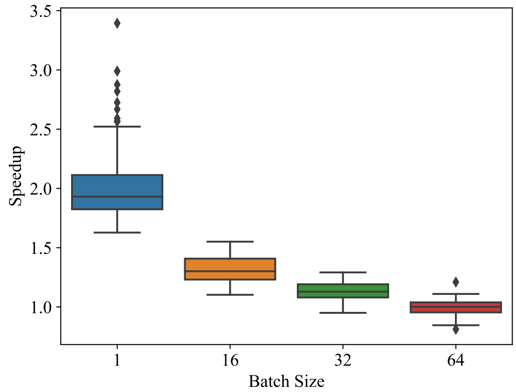

Figure 8: Speedup gained from `int8` quantization for each batch size compared to `fp32` inference.

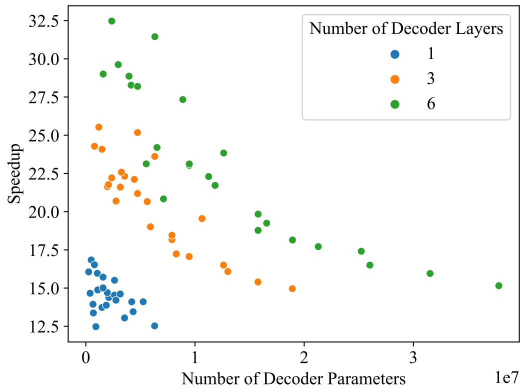

Figure 9: Speedup gained from batch size of 64 compared to 1 on GPU.

Figure 10 shows the effect of using batch size on speed for both the CPU and the GPU, the CPU having more variance for models with the same number of decoder layers, while the GPU has a smaller variance in models with 1-layer decoder, and more variance between different layer configurations.

Figure 11 Shows the difference between effect of parameter size on inference speed, both for the CPU and the GPU. The CPU inference is mostly affected by the number of parameters, while on the GPU number of layers plays an important role in determining inference speed.

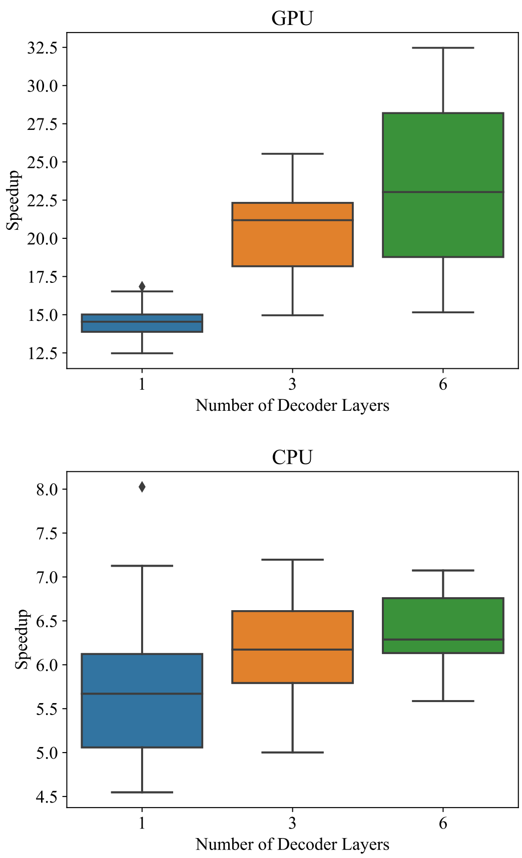

Figure 10: Speedup gained from usning batch size of 64 compared to 1 on GPU (top) and CPU (bottom) for different number of decoder layers.

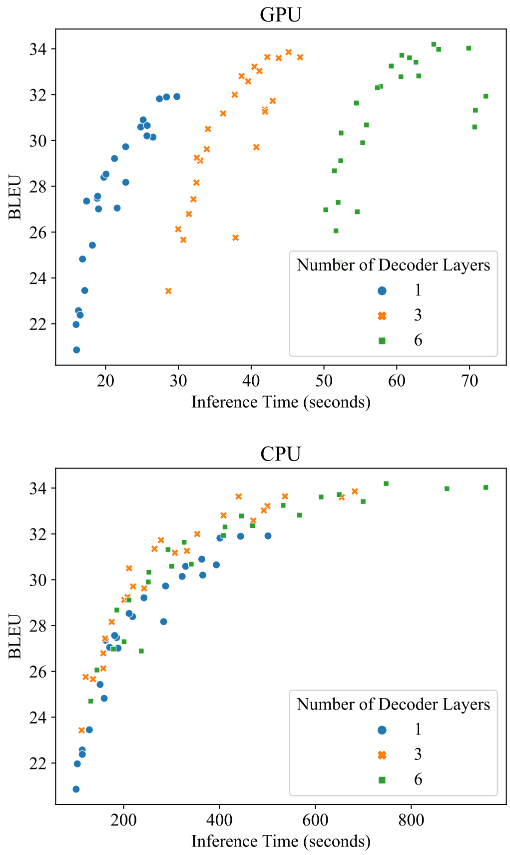

Figure 11: Time vs. BLEU score on GPU (top) and CPU (bottom).