TenTrans High-Performance Inference Toolkit for WMT2021 Efficiency Task

Kaixin Wu, Bojie Hu, Qi Ju*
TencentMT Oteam
{danielkxwu, bojiehu, damonju}@tencent.com

Abstract

The paper describes the TenTrans's submissions to the WMT 2021 Efficiency Shared Task. We explore training a variety of smaller compact transformer models using the teacher-student setup. Our model is trained by our self-developed open-source multilingual training platform TenTrans-Py[1]. We also release an open-source high-performance inference toolkit[2] for transformer models and the code is written in C++ completely. All additional optimizations are built on top of the inference engine including attention caching, kernel fusion, early-stop, and several other optimizations. In our submissions, the fastest system can translate more than 22,000 tokens per second with a single Tesla P4 while maintaining 38.36 BLEU on En-De *newstest2019*. Our trained models and more details are available in TenTrans-Decoding competition examples[3].

1 Introduction

We participate in the GPU throughput track of the Workshop on Machine Translation (WMT) 2021 Efficiency Shared Task. The efficiency task aims at exploring the different techniques for training and optimizing GPU models for high throughput while preserving the highest possible accuracy. While we do not pay more attention to training techniques, we apply a variety of optimizations to improve the computation efficiency of our GPU models in the inference phase.

In terms of the training phase, we trained a variety of smaller compact student models using the common teacher-student training approach (Hinton et al., 2015; Kim and Rush, 2016) on our open-source multilingual training platform TenTrans-Py. All of them are based on the deep transformer which has proven more effective and has lower training costs than the wide transformer models (Wang et al., 2019). For the inference phase, our strategy for the shared task includes attention caching, kernel fusion, early-stop, and several other optimizations. All of these optimizations are employed in a high-optimized and C++-based inference engine TenTrans-Decoding.

The paper is structured as follows: Section 2 describes the data preparation and the training details, then Section 3 presents the variety of ours optimizations to improve decoding efficiency. The detailed accuracy and efficiency results are shown in Section 4. Finally, we conclude our work in Section 5.

2 Teacher-student Training

To train smaller compact student models, the teacher-student training approach (Hinton et al., 2015; Kim and Rush, 2016) is adopted. First, a large model (the teacher) is trained on all available bilingual data, included synthetic data generated by the back-translation (Sennrich et al., 2015a) method. Multiple model ensembles are also typically used to build stronger teacher systems. Then, all our small optimized models (the student) are created using sequence-level knowledge distillation (Kim and Rush, 2016) and trained on data generated from the teacher model. The sequence-level knowledge distillation is a common technique that has proven successful for reducing the size of neural models, especially in NMT tasks.

2.1 Deep Transformer

Transformer networks (Vaswani et al., 2017) are the current state-of-the-art in many machine translation tasks, and the deep transformer (Wang et al., 2019) which simply stacks more encoder layers has been proved to further enhance the accuracy of the model. To stabilize the training of the deep

* Corresponding author.

[1] https://github.com/TenTrans/TenTrans

[2] https://github.com/TenTrans/TenTrans-Decoding

[3] https://github.com/TenTrans/TenTrans-Decoding/blob/master/examples/WMT21-Efficiency.md

Proceedings of the Sixth Conference on Machine Translation (WMT), pages 800–803

Transformer	N_{enc}	N_{dec}	h	d_{model}	d_{ff}	param.	BLEU
Teacher-base-20_6 (2xFFN)	20	6	8	512	4096	160M	39.97
Student-base-20_1	20	1	8	512	2048	88M	39.93
Student-base-10_1	10	1	8	512	2048	58M	39.30
Teacher-tiny-20_1	20	1	8	256	1024	28M	38.36

Table 1: Transformer model configurations and SacreBLEU (Post, 2018) scores on *newstest2019*.

model, we use the Pre-Norm strategy (Wang et al., 2019). The layer normalization (Ba et al., 2016) is applied to the input of every sub-layer which the computation sequence could be expressed as: layer normalization $\longrightarrow$ multi-head attention / feed-forward $\longrightarrow$ residual-add. All of our models are based on deep transformer architecture.

2.2 Teacher & Student Models

The different model configurations for both teacher and student models are presented in Table 1. We train a teacher model and three student model variant with a different number of encoder layers N_{enc}, decoder layers N_{dec}, hidden size d_{model}, and feed-forward network size d_{ff}. We adopt a deep encoder and a shallow decoder architecture of all student models, and the number of decoder layers is set to 1 by default. All of our models tie source embedding, target embedding, and softmax weights.

2.3 Data and Training Details

Dataset Following the shared task setup, we limit our training data to the WMT 2021 English-German translation task. The bilingual data used in the English-German task includes all the available corpora provided by WMT 2021: Europarl v10, ParaCrawl v7.1, News Commentary, Wiki Titles v3, Tilde Rapid corpus and WikiMatrix. For monolingual data, we only use NewsCrawl2020, Europarl v10, and News Commentary for back-translation.

Data preprocessing Then, we normalize punctuation and tokenize all data with the Moses tokenizer (Koehn et al., 2007). For the bitext datasets, we remain sentences no longer than 200 words as well as sentence pairs with a source / target length ratio between 0.3 and 2.0. The fast-align tools (Dyer et al., 2013) are applied to further obtain a cleaned and high-quality parallel corpus. For the monolingual dataset, the sentences with words between 4 and 200 are remained. See Table 2 for details on the bitext and monolingual dataset sizes. After that, we use joint byte pair encodings (BPE) with 32K split operations for subword segmentation (Sennrich et al., 2015b).

	En-De	De (**mono.**)
No filter	49.2M	57.0M
+ length filter	46.9M	55.2M
+ fast-align	41.2M	-

Table 2: Number of sentences in bitext and monolingual datasets for different filtering schemes.

Student training First, we train the teacher model on all available bilingual data, including synthetic data through the back-translation method, and we use English-German *newstest2019* as the development set. We ensemble four best models for building a stronger teacher. Then, the English part of the bilingual data is translated by the teacher model and the resulting synthesized parallel data is used to train the student models. Table 1 shows their evaluation scores on *newstest2019* of different models. The results correlate well with the expectation that more model parameters lead to better performance. Our distillation student models show strong competitiveness even when the number of parameters is greatly reduced.

3 GPU Inference Optimizations

3.1 Implementation: TenTrans-Decoding

TenTrans-Decoding is an open-source high-optimized inference engine for transformer models and the code is written in C++. TenTrans-Decoding's goal is to offer a lightweight and rapid deployment of high-performance service solutions for executing models. All additional optimizations are built on top of the inference engine.

3.2 Attention Caching

We apply the common technique of caching linear projections in Transformer decoder layers. More specifically, we cache the linear transformations for keys and values before cross-attention layers and each step of decoder self-attention layers.

3.3 Kernel Fusion

To reduce kernel launching overhead and enhance the GPU computation efficiency, we implement many kernel fusion techniques for our Transformer models.

- ***Add_bias_residual_layerNormalization*** For the layer normalization between two General Matrix Multiplications (GEMMs), we reorganize the *AddBias* kernel, residual network, and *LayerNormalization* kernel into a single one.
- ***Add_bias_ReLU*** In the Feed-Forward network layers of the Transformer model, the *AddBias* kernel and *ReLU* kernel are fused into one.
- ***Add_bias_residual*** For the output of every encoder or decoder layer, we fuse the *AddBias* kernel and residual network.
- ***Fused_multihead_attention*** In addition to the fusion techniques above, we also fuse the attention layer by packing GEMMs and bias to further improve the computation efficiency.

Figure 1 details the kernel fusion techniques of a transformer decoder layer. The computation graph of a transformer can be reorganized into a more compact graph by fusing all the kernels between two GEMMs into a single one.

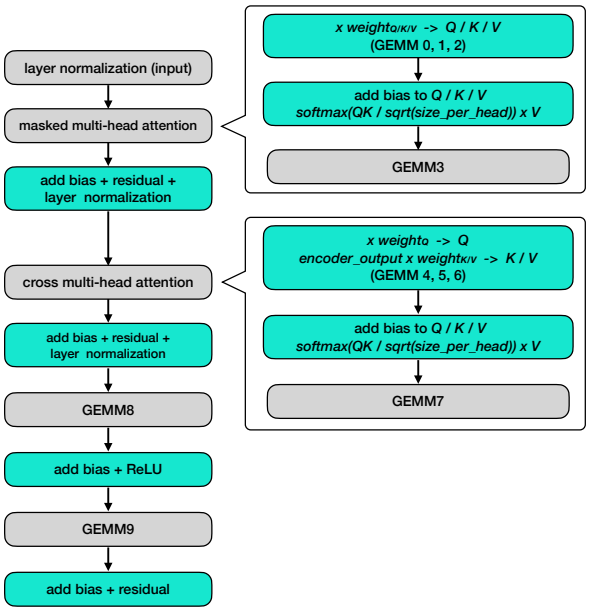

Figure 1: Kernel fusion of a transformer decoder layer. The part in darker color represent using the kernel fusion technique.

	Speed	Ratio	BLEU
TenTrans-Py	696.5	1.00x	38.91
TenTrans-Decoding	1822.4	2.62x	38.91
+ kernel fusion	2565.4	3.68x	38.82
+ early-stop	2682.5	3.85x	38.82
+ sorted batch	5034.8	7.23x	38.98

Table 3: The decoding speed (source tokens per second) and SacreBLEU scores on *newstest2019* for student-tiny-20_1. The speed is measured by a single Tesla P4 GPU and the beam size is 4.

3.4 Early-stop

In batch decoding, the number of decoding ending steps between sentences is different. The early-stop strategy which optimizes kernel function is adopted to avoid redundant computation. For sentences that have been decoded in batch, there is no additional computation for these sentences until the whole batch has been decoded.

3.5 Sorted Batch & Greedy Search

In addition to the methods above, we sort all input sentences from shortest to longest, and the batch size is 128 in our settings. The sorting makes the batches contain sentences of similar sizes which reduces the amount of padding and increases the computation efficiency. During decoding, we use greedy search instead of beam search since we find the distillation model are insensitive to the beam size. We skip the final softmax layer and simply get the maximum from the output logits.

4 Optimization Results

Table 3 shows the impact of different inference optimizations when decoding the Student-tiny-20_1 student transformer model. TenTrans-Decoding leads to a 2.62x speedup than the TenTrans-Py baseline without any inference optimizations. Combine all the inference optimizations mentioned above, it can achieve a 7.23x speedup with no accuracy loss over the baseline.

Table 4 presents all of our submissions and we only participate in the GPU-throughput track. As details in Table 4, we report our model configuration, model size, and metric for translation, including SacreBLEU scores on *newstest2019* and the real translation time cost. All of our systems are tested on a single Tesla P4 GPU. All student models follow a deep encoder and a shallow decoder architecture, the number of decoder lay-

transformer	Model size	Speed (tokens/s)	Ratio	Time Cost (s)	BLEU
Teacher-base-20_6 (2xFFN)	642MB	6274.0	1.00x	9.80	39.97
Student-base-20_1	354MB	12128.1	1.93x	5.07	39.93
Student-base-10_1	234MB	15900.3	2.53x	3.87	39.30
Student-tiny-20_1	113MB	22481.8	3.58x	2.74	38.36

Table 4: Results of all submissions. Time Cost in seconds to translate *newstest2019* and BLEU scores are reported using SacreBLEU. The *newstest2019* contains 1997 sentences. All systems were executed on a single Tesla P4 GPU with greedy search.

ers is 1 by default. All student models training with sequence-level distillation show a competitive performance. The Student-base-20_1 transformer achieves a 1.93x speedup over the teacher baseline with almost no accuracy loss, and the amount of parameters is greatly reduced. Compared with the teacher baseline, the Student-base-10_1 transformer has a speedup of 2.53x times and a slight decrease of only 0.67 BLEU. The Student-tiny-20_1 transformer, our fastest system, which has one-sixth parameters of the teacher model, achieves 38.36 BLEU on *newstest2019* and speeds up the teacher baseline by 3.58x.

In this version, we do not pay more attention to the model size, memory footprint, and low precision inference (e.g., FP16). All operations on the model are based on FP32 floating-point numbers. In the future version, we plan to optimize these points mentioned above.

5 Conclusion

This work presents the TenTrans's submissions to the 2021 Efficiency Shared Task of WMT. We show the deep encoder and shallow decoder student models that training with sequence-level distillation can achieve a competitive performance both in speed and accuracy compared with the teacher baseline. To further improve computation efficiency, we combine several optimizations including attention caching, kernel fusion, early-stop and sorted batch. Finally, our fastest student model achieves a speedup of 3.58x times, while only has one-sixth parameters of the teacher baseline.

In the future, we will apply low-precision inference (e.g., FP16) and more kernel fusion techniques to improve the computation efficiency of our GPU systems. Furthermore, we will continue to explore a more efficient teacher-student training approach to obtain compact student models with competitive performance both in quality and speed.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Chris Dyer, Victor Chahuneau, and Noah A Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015a. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015b. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

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 *Advances in neural information processing systems*, pages 5998–6008.

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. *arXiv preprint arXiv:1906.01787*.