LightSeq: A High Performance Inference Library for Transformers

Xiaohui Wang, Ying Xiong, Yang Wei, Mingxuan Wang, Lei Li
ByteDance AI Lab
{wangxiaohui.neo, xiongying.taka, weiyang.god}@bytedance.com
{wangmingxuan.89, lileilab}@bytedance.com

Abstract

Transformer, BERT and their variants have achieved great success in natural language processing. Since Transformer models are huge in size, serving these models is a challenge for real industrial applications. In this paper, we propose LightSeq, a highly efficient inference library for models in the Transformer family. LightSeq includes a series of GPU optimization techniques to to streamline the computation of neural layers and to reduce memory footprint. LightSeq can easily import models trained using PyTorch and Tensorflow. Experimental results on machine translation benchmarks show that LightSeq achieves up to 14x speedup compared with TensorFlow and 1.4x compared with FasterTransformer, a concurrent CUDA implementation. The code is available at https://github.com/bytedance/lightseq.

1 Introduction

Sequence processing and generation have been fundamental capabilities for many natural language processing tasks, including machine translation, summarization, language modeling, etc (Luong et al., 2015; Qi et al., 2020; Dai et al., 2019). In recent years, with the introduction of Transformer model (Vaswani et al., 2017b), many pre-trained language models such as BERT, GPT, and mRASP have also been widely used in these tasks (Devlin et al., 2019; Radford et al., 2019; Yang et al., 2020; Lin et al., 2020).

However, the parameters of these models become increasingly large, which causes the high latency of inference and brings great challenges to the deployment (Kim and Hassan, 2020). The current popular inference systems are not necessarily the best choice for the online service of sequence processing problems. First, training frameworks, such as TensorFlow and PyTorch, require accommodating flexible model architectures and backward propagation, which introduce additional memory allocation and extra overhead of using fine-grain kernel functions. Therefore, the direct deployment of the training framework is not able to make full use of the hardware resource. Taking an example of machine translation, the Transformer big model currently takes roughly 2 seconds to translate a sentence, which is unacceptable in both academia and industry (Edunov et al., 2018; Hsu et al., 2020). Second, current optimizing compilers for deep learning such as TensorFlow XLA (Abadi et al., 2017), TVM (Chen et al., 2018) and Tensor RT (Vanholder, 2016) are mainly designed for fixed-size inputs. However, most NLP problems enjoy variable-length inputs, which are much more complex and require dynamic memory allocation. Therefore, a high-performance sequence inference library for variable-length inputs is required. There are several concurrent CUDA libraries which share a similar idea with our project, such as FasterTransformer [1] and TurboTransformers (Fang et al., 2021).

We will highlight three innovative features that make LightSeq outperforms similar projects. First, we replace a straightforward combination of fine-grained GPU kernel functions in TensorFlow or PyTorch implementations with coarse-grain fused ones, which avoid high time cost introduced by a mass of kernel function launches and GPU memory I/O for intermediate results. As a result, LightSeq reduces the atomic kernel functions by four times compared with Tensorflow approaches. Second, we specially design a hierarchical auto regressive search method to speed up the auto-regressive search. Third, we propose a dynamic GPU memory reuse strategy. Different from fixed-length inputs, sequence processing tackles the variable-length inputs, which bring difficulty for memory allocation. LightSeq proposes to pre-define the maximal memory for each kernel function and shares the GPU

[1] https://github.com/NVIDIA/FasterTransformer

Proceedings of NAACL HLT 2021: IndustryTrack Papers, pages 113–120
June 6–11, 2021.

Inference Libraries	Models					Decoding Methods		
	Transformer	GPT	VAE	BERT	Multilingual	Beam Search	Diverse Beam Search	Sampling
FasterTransformer	✓	✓	✗	✓	✗	✓	✓	✓
TurboTransformers	✓	✗	✗	✓	✗	✗	✗	✗
LightSeq	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Features for FasterTransformer, TurboTransformers and our proposed LightSeq. LightSeq supports the most features for a comprehensive set of Transformer models.

memory across non-dependent ones. As a result, LightSeq reduces eight times memory allocation without loss of inference speed. As a benefit, LightSeq enjoys several advantages:

Efficient LightSeq shows better inference performance for generation tasks. For example, in machine translation benchmarks, LightSeq achieves up to 14 times speedup compared with TensorFlow and 1.4 times speedup compared with FasterTransformer.

Functional LightSeq supports more architecture variants, such as BERT, GPT, Transformer, and Variational Autoencoders (VAEs). Further, LightSeq provides different search algorithms, such as beam search, diverse beam search and probabilistic sampling (Vijayakumar et al., 2018). Table 1 shows the functional comparison between FasterTransformer[2], TurboTransformers[3], and LightSeq in text generation tasks.

Convenient LightSeq is easy to use, which contains a serving system and efficient CUDA implementations. The popular models, such as BERT, Roberta, GPT, VAEs, MT Transformer, and Speech Transformer can be directly deployed online without code modification. For user-specific architectures, LightSeq supports multiple model reuse, which can be easily adapted with only a few lines of code modification.

2 LightSeq Approach

Transformer-based NLP models mainly consist of two components during inference: the feature calculation layer and the output layer, as shown in Figure 1.

The feature calculation layer is mainly based on self-attention mechanism and feature transformation, which is actually implemented by matrix multiplication and a series of I/O-intensive operations such as element-wise (e.g., reshape) and reduce (e.g., layer normalization).

The output layer slightly changes in different tasks, such as classification in NLU tasks or search (e.g., beam search) in NLG tasks. This layer is usually composed of the `Softmax` over vocabulary, probability sorting, cache refreshing, etc., which are essentially I/O-intensive.

These two components pose challenges for efficient inference:

- The fine-grained call of I/O-intensive GPU kernel function brings a huge amount of GPU memory I/O, which becomes the performance bottleneck of feature calculation.
- Redundant calculations exist due to the fact that we only need a few tokens/labels with the highest probability instead of all in classification or search for the output layer.
- Dynamic shape in variable sequence length and auto-regressive search makes it difficult to achieve memory reuse within or between requests, which leads to a large number of GPU memory allocation during model service.

LightSeq employs a series of innovative methods to address these challenges to accelerate model development, such as fusion of multiple kernel functions to reduce I/O overhead, hierarchical optimization of search algorithms to erase redundant calculations, and reuse of dynamic GPU memory to avoid run-time allocation. The following is a detailed introduction to these methods.

2.1 Operation Fusion

Transformer feature calculation layer needs to be highly optimized since it is ubiquitous in various

[2]As of this writing, we use FasterTransformer v2.1 for comparison.

[3]we use TurboTransformers for comparison at commit 0eae02ebadc8b816cd9bb71f8955a7e620861cd8

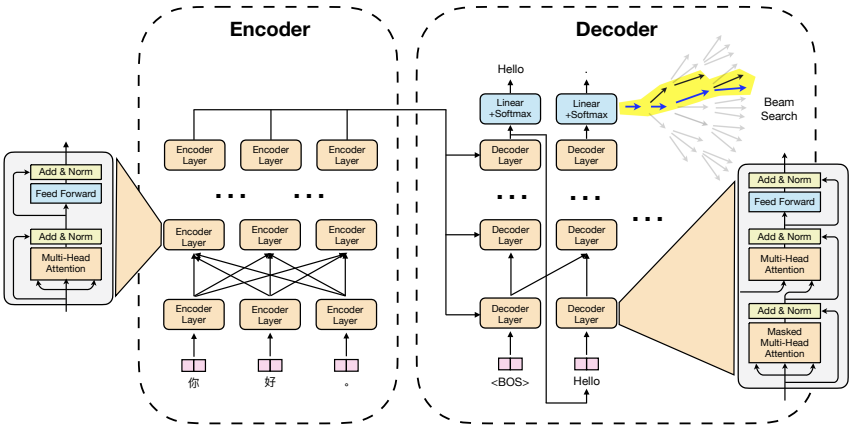


Figure 1: The process of sequence to sequence generation using Transformer model with beam search.

NLP tasks today. In most deep learning frameworks, such as TensorFlow and PyTorch, it is implemented by a straightforward combination of fine-grained kernel functions from standard libraries provided by hardware manufacturers, which introduces high time cost due to a mass of kernel function launches and GPU memory I/O for intermediate results.

Taking layer normalization implemented by TensorFlow as an example, there are still three kernel launches[4] and two intermediate results (mean and variance) even with the help of optimizing compilers like TensorFlow XLA (Abadi et al., 2017). As a comparison, we can write a custom kernel function dedicated to layer normalization based on the CUDA toolkit, which produces only one kernel launch without intermediate results.

LightSeq implements the Transformer feature calculation layer with general matrix multiply (GEMM) provided by cuBLAS[5] and custom kernel functions. The detailed structure is shown in Figure 2. Combination of fine-grained operations between GEMM operations is fused into one custom kernel function. In consequence, there are only six custom kernel functions and six GEMM in a Transformer encoder layer, which is usually more than four times less than its corresponding implementation in common deep learning frameworks like TensorFlow or PyTorch.

2.2 Hierarchical Auto Regressive Search

LightSeq supports a comprehensive set of output layers, such as sentence-level and token-level classification, perplexity calculation for language models, and auto-regressive search like beam search, diverse beam search and top-k/top-p sampling (Holtzman et al., 2020). Redundant calculations often exist in these output layers since we only need a few labels/tokens with the highest probability instead of all of them. Auto-regressive search is relatively complicated, and we will discuss it in the next paragraph. For the other types of output layers, we can simply replace `Softmax` with the probability calculation of token/label with the highest `logits`, which brings more obvious benefit when the size of vocabulary or labels is large.

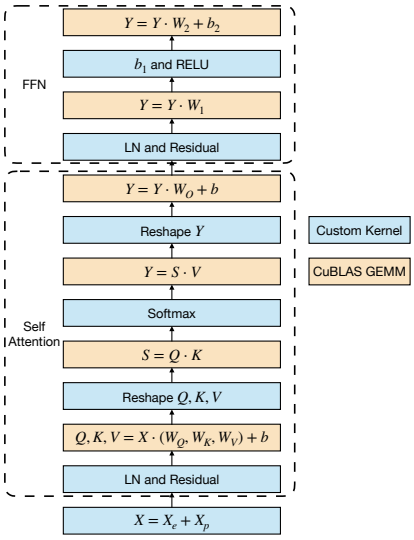


Figure 2: The structure of optimized Transformer encoder layers in LightSeq.

Auto-regressive search is widely used in machine translation and text generation. LightSeq proposes Hierarchical Auto Regressive Search (HARS) method to erase redundant calculations and parallel computing. Here we take the most used beam search method as an example to intro-

[4]Two for `reduce_mean` operations and one for calculation of the final result.

[5]https://developer.nvidia.com/cublas

duce the proposed HARS method.

In one step of the beam search process, given the `logits`, we need to perform two calculations over the whole vocabulary:

1. Compute the conditional probability using `Softmax` and write the intermediate result into GPU memory.

2. Read the intermediate result from GPU memory and select the top-k beams and tokens by sequential probability.

These two calculations are highly time-consuming since the vocabulary size is usually in tens of thousands of scales. For example, they account for a latency proportion of 30% in Transformer base models.

In order to reduce the input size of these two calculations, LightSeq introduces a two-stage strategy that is widely employed in the recommended system: retrieve and re-rank.

Before the probability computation and top-k selection, the retrieve is carried out first. For each beam, we calculate as follows:

1. Randomly divide `logits` into k groups.

2. Calculate the maximum of group i, denoted as m_i

3. Calculate the minimum of m_i, denoted as $\mathcal{R}$, which can be regarded as a rough top-k value of `logits`.

4. Select `logits` larger than $\mathcal{R}$ and write them into GPU memory.

The retrieve is co-designed based on GPU characteristics and `logits` distribution. Hence it is efficient and effective:

- Efficient. The retrieve is implemented by one kernel function and can be executed within a dozen instruction cycles.

- Effective. After the retrieve, only dozens of candidates were selected.

After the retrieve, the original two calculations of beam search will be carried out on the small set of candidates, named as Hierarchical Auto Regressive Search.

Figure 3 is a detailed illustration of the proposed hierarchical strategy. In the original beam search method, we need to compute the probability and select the top-k over the whole vocabulary. However, by hierarchical method, we only need to pick a small set of candidates from each beam and then perform probability computation and top-k selection.

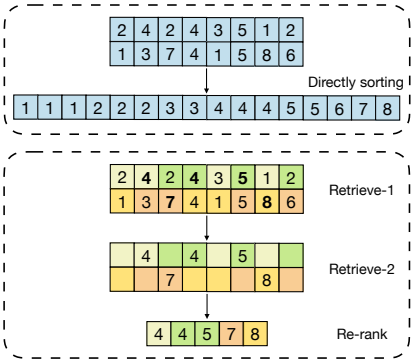


Figure 3: An illustration of the proposed hierarchical strategy. In this case, beam size is 2 and vocabulary size is 8. Each row represents `logits` in a beam.

2.3 Dynamic GPU Memory Reuse

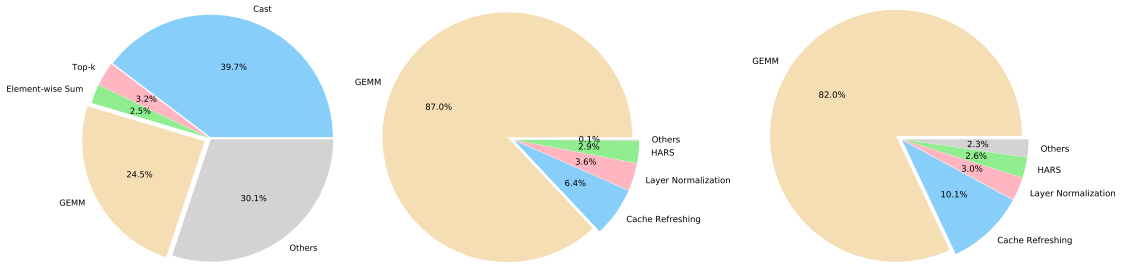
In order to save GPU memory occupancy and avoid allocation of GPU memory during the model serving, LightSeq pre-defines the maximum of dynamic shapes, such as the maximal sequence length. At the start of the service, each intermediate result in the calculation process is allocated GPU memory to its maximum. Besides, GPU memory is shared for non-dependent intermediate results.

Through this memory reuse strategy, on a T4 graphics card, we can deploy up to 8 Transformer big models[6] at the same time, so as to improve graphics card utilization in low frequency or peak-shifting scenarios.

3 Experiments

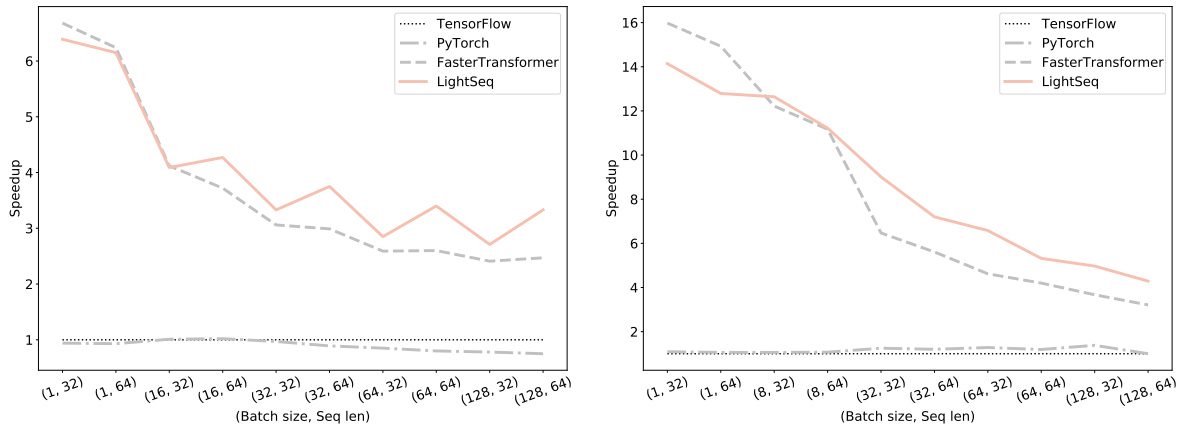
In this section, we will show the improvements of LightSeq with different GPU hardware and precisions. We first analyze the GPU occupation of LightSeq during inference to investigate if LightSeq can make full use of GPU resources. Then, we make a fair comparison with TensorFlow, PyTorch, FasterTransformer, and TurboTransformers on machine translation and text generation to show the efficiency of LightSeq.

[6]Under the configuration of 8 batch size, 256 sequence length, 4 beam size and 30000 vocabulary size.



(a) TensorFlow with Float16. (b) LightSeq with Float16. (c) LightSeq with Float32.

Figure 4: Proportion of computation occupation. GEMM is the main indicator and the larger number indicates the higher computation efficiency.



(a) P4 speedup in Float32. (b) T4 speedup in Float16.

Figure 5: Speedup on Transformer with beam search compared with FasterTransformer, TurboTransformers and PyTorch implementation. The baseline is TensorFlow implementation.

3.1 Experiment Settings

We test the generation performance of LightSeq on two latest NVIDIA inference GPU Tesla P4 and T4, choosing TensorFlow, PyTorch, and FasterTransformer implementations as a comparison. Another related library, TurboTransformers, mainly focuses on the Transformer encoder and is not powerful enough for text generation. Its speedup for sequence generation compared to TensorFlow is only about 15%, and it only supports Float32 on GPU. Therefore we do not compare with it.

The experiments on machine translation are conducted on the popular WMT14 English to German translation tasks. The hyper-parameters setting resembles transformer base model (Vaswani et al., 2017a). Specifically, we reduce the vocabulary size of both the source language and target language to 50K symbols using the sub-word technique (Bojanowski et al., 2017).

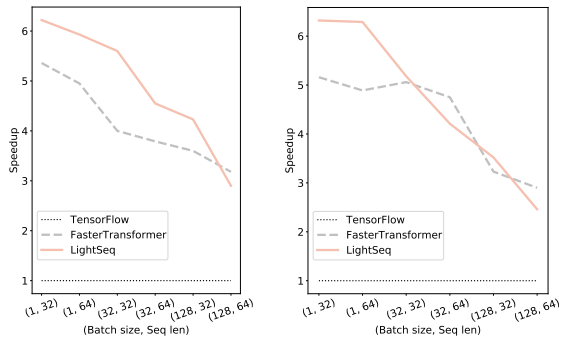
The experiments on text generation are conducted on a randomly initialized Transformer model and test dataset. Results of Tensorflow and FasterTransformer are obtained from the scripts in the source code of FasterTransformer. The sequence length is used for limiting the total size in the generation test, and the values for top-k and top-p are the most selected settings in our deployments.

3.2 GPU Occupation of LightSeq

We first analyze the GPU occupation to verify the efficiency of LightSeq. The experiments are conducted on Tesla T4 card with the GPU profiling toolkit. The latency of each module is shown in Figure 4 with both Float16 and Float32 precision. We classify the operation into three categories: GEMM, cache refreshing, and others. GEMM latency is the most important indicator, which shows the proportion of matrix calculations occupying the GPU calculation.

After optimization, we can find that:

- GEMM operation in LightSeq accounts for



(a) Top-$p = 0.75$. (b) Top-$k = 32$.

Figure 6: T4 speedup on Transformer with sampling compared with FasterTransformer in Float16. LightSeq outperforms FasterTransformer in most cases.

87% and 82% respectively for Float16 and Float32, accounting for most of the inference time. However, in the original TensorFlow model, GEMM operations account for only 25%. This shows that beam search optimization has achieved good results.

- Cast and other operations in TensorFlow are expensive, which launches over 80 different GPU kernels. In LightSeq, we fuse cast operations into weight loading, and other operations into more efficient implementations.

- The latency of cache refreshing in LightSeq accounts for 6% and 10% respectively, which are not negligible but hard to be optimized further. Possible solutions include reducing the amount of cache, such as reducing the number of decoder layers, reducing cache precision, etc.

The results demonstrate that LightSeq has been optimized to a disabling extent and greatly increases the speed of inference. Another interesting finding is that Float16 is more efficient than Float32. A possible explanation is that Float16 occupies less memory. Therefore the cache refreshing and memory I/O operations potentially take less time.

3.3 Comparison on Machine Translation

The comparison between LightSeq, TensorFlow, PyTorch and FasterTransformer are shown in Figure 5. We group the test set into different buckets according to the sequence length and batch size. For example, the x-axis (a, b) indicates that the batch size is a and the sequence length is b. The y-axis is the speedup compared with TensorFlow baseline. The results provide several interesting findings:

- For both LightSeq and FasterTransformer, the speedup gap for smaller batch size or shorter sequence length is much larger.

- The speedup for T4 is larger than P4. The main reason is that T4 is more powerful than P4 and has much room for improvement.

- In most cases, LightSeq performs better than FasterTransformer. For larger batch size and longer sequences, the gap increases. While for smaller batch size, FasterTransformer performs better.

- PyTorch is slightly slower than TensorFlow in P4 and faster in T4, which indicates that LightSeq also greatly outperforms PyTorch in all cases.

The findings provide some guidance for optimization work in the future. There is almost no space to accelerate the inference by fusion of non-computationally intensive operators, especially for small batch size. Future work is recommended to focus on optimizing GEMM operations which account for 80% to 90% of the total computation time.

Finally, we compare TurboTransformers with PyTorch by the translation demo[7]. As of this writing, only decoder layers of MT Transformer in float32 precision is supported, so we only compare the latencies of decoder layers without beam search and cache refreshing. In the final results, TurboTransformers only achieves about 2x speedup for different batch sizes and sequence lengths. So TurboTransformers has no comparability with LightSeq in machine translation tasks (As TurboTransformer repo says, "TurboTransformer will bring 15.9% performance improvements on RTX 2060 GPU. We are still working on decoder model optimization.").

3.4 Comparison on Text Generation

In the text generation scenario, the sampling strategy is applied to improve the diversity of generation. Among which, top-k and top-p sampling strategies are more popular.

[7] https://github.com/TurboNLP/Translate-Demo/tree/443e6a46fefbdf64282842b6233a8bd0a22d6aeb

Figure 6 shows the performance comparison of Transformer base with top-k/top-p sampling. The values of top-k and top-p are added in the x-axis. The results provide following findings:

- In most cases, LightSeq achieves greater speedup than FasterTransformer. Unlike results in machine translation, LightSeq performs better for smaller batch size and shorter sequence, while FasterTransformer performs better for larger batch size and longer sequence.

- The speedup in generation tasks are not as large as machine translation. It is mainly because of the lower complexity of sampling methods than beam search, reducing the benefits obtained from operation fusion and HARS.

4 Conclusion

In this paper, we address the deployment problem of expensive sequence models and present an efficient inference library LightSeq for sequence processing and generation, reducing the gap between the performance of big models and the requirement of online services. Comparisons with FasterTransformer show that we perform better in both machine translation and text generation. In future work, we will focus on exploring more techniques to achieve a more significant speedup, including efficient integer-arithmetic-only inference and sparse GEMM computations.

Acknowledgments

We would like to thank the colleagues in machine translation service and advertisement service to support our experiments in online environments and apply LightSeq into real-time systems.

References

Martín Abadi, Michael Isard, and Derek Gordon Murray. 2017. A computational model for tensorflow: an introduction. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2017, Barcelona, Spain, June 18, 2017*, pages 1–7. ACM.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, Carlsbad, CA. USENIX Association.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2978–2988. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 489–500. Association for Computational Linguistics.

Jiarui Fang, Yang Yu, Chengduo Zhao, and Jie Zhou. 2021. Turbotransformers: an efficient GPU serving system for transformer models. In *PPoPP '21: 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Virtual Event, Republic of Korea, February 27- March 3, 2021*, pages 389–402. ACM.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yi-Te Hsu, Sarthak Garg, Yi-Hsiu Liao, and Ilya Chatsviorkin. 2020. Efficient inference for neural machine translation. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 48–53, Online. Association for Computational Linguistics.

Young Jin Kim and Hany Hassan. 2020. FastFormers: Highly efficient transformer models for natural language understanding. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 149–158, Online. Association for Computational Linguistics.

Zehui Lin, Xiao Pan, Mingxuan Wang, Xipeng Qiu, Jiangtao Feng, Hao Zhou, and Lei Li. 2020. Pre-training multilingual neural machine translation by leveraging alignment information. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 2649–2663. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421. The Association for Computational Linguistics.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 2401–2410. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Han Vanholder. 2016. Efficient inference with tensorrt.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017a. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017b. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010.

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. 2018. Diverse beam search for improved description of complex scenes. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 7371–7379. AAAI Press.

Jiacheng Yang, Mingxuan Wang, Hao Zhou, Chengqi Zhao, Weinan Zhang, Yong Yu, and Lei Li. 2020. Towards making the most of BERT in neural machine translation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9378–9385. AAAI Press.