SAMP: A Model Inference Toolkit of Post-Training Quantization for Text Processing via Self-Adaptive Mixed-Precision

Rong Tian[1]*, **Zijing Zhao**[2], **Weijie Liu**[2,3], **Haoyan Liu**[3,4], **Weiquan Mao**[3], **Zhe Zhao**[3] **and Kan Zhou**[1]

[1]Kuaishou Technology, Beijing, China
[2]Peking University, Beijing, China
[3]Tencent AI Lab, Beijing, China
[4]Institute of Dataspace, Hefei Comprehensive National Science Center, Anhui, China

Abstract

The latest industrial inference engines, such as FasterTransformer[1] and TurboTransformers (Fang et al., 2021), have verified that half-precision floating point (FP16) and 8-bit integer (INT8) quantization can greatly improve model inference speed. However, the existing INT8 quantization methods are too complicated, and improper usage will lead to model performance damage greatly. In this paper, we develop a toolkit for users to easily quantize their models for inference, in which **S**elf-**A**daptive **M**ixed-**P**recision (SAMP) is proposed to automatically control quantization rate by a mixed-precision architecture to balance model accuracy and efficiency. Experimental results show that our SAMP toolkit has a higher speedup than PyTorch (Paszke et al., 2019) and FasterTransformer while ensuring the required accuracy. In addition, SAMP is based on a modular design, decoupling the tokenizer, embedding, encoder and target layers, which allows users to handle various downstream tasks and can be seamlessly integrated into PyTorch.

1 Introduction

Text understanding is one of the basic tasks in the field of Natural Language Processing (NLP), including information retrieval, dialogue system, sentiment recognition, summarization, language model, etc. Transformer-based models (Vaswani et al., 2017) have achieved state-of-the-art in many downstream tasks, such as BERT (Devlin et al., 2018), XLNet (Yang et al., 2019), Google T5 (Raffel et al., 2020), etc. In some large industrial systems, training frameworks (e.g. TensorFlow (Abadi et al., 2016) or PyTorch (Paszke et al., 2019)) are not good options to deploy models due to the lack of high GPU occupation considerations and good memory management of them during the inference phase (Wang et al., 2021).

Conventional inference acceleration tools for deep learning models such as NVIDIA TensorRT (Vanholder, 2016), TurboTransformers (Fang et al., 2021) and LightSeq (Wang et al., 2021) are primarily designed for fixed-size or variable-length inputs. These tools' optimization concepts mainly take into account memory management, operation fusion, or other data pruning techniques in the online computing systems, mostly single-precision calculation (only floating-point is used). So the acceleration performance is limited. On this basis, FasterTransformer developed by NVIDIA performs fixed-point acceleration on Transformer models (using Fully-quantization in all transformer layers), and has achieved an excellent speedup compared with floating-point. However, this method of Fully-quantized in all transformer layers makes it difficult for INT8-quantization inference results to achieve the accuracy of floating-point calculations, resulting in a large loss of calculation accuracy in specific tasks, making it difficult to be widely used. On the other hand, we find that the kernel-fusion policy in FasterTransformer INT8-quantization implementation can still be optimized.

To solve these problems, we propose an inference toolkit SAMP, which contains a self-adaptive mixed-precision Encoder and a series of advanced fusion strategies. Objectively, The mixed-precision calculation of floating-point and fixed-point can obtain better calculation accuracy than fully-fixed-point calculation. **Self-Adaptive Mixed-Precision Encoder** can find an optimal combination of mixed-precision among a large number of General Matrix Multiplication (GEMM) operations and Transformer layers, which can align the performance of model inference most closely with user needs (calculation accuracy and inference latency). **Advanced Fusion Strategies** make fusion improvements for embedding kernels and quantization-related operations respectively, reducing CUDA kernel calls by half. Moreover, SAMP is an end-

*Corresponding author: Rong Tian.
E-mail: tianrong03@kuaishou.com

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

Inference Toolkit	Tokenizer	Mixed-Precision GEMMs			Downstream Tasks		
		Layers	MHA-FFN	Fully-quantized	Classification	NER	Text Matching
FasterTransformer	✗	✗	✗	✓	✓	✗	✗
TurboTransformers	✗	✗	✗	✗	✓	✗	✗
LightSeq	✗	✗	✗	✗	✗	✗	✗
SAMP	✓	✓	✓	✓	✓	✓	✓

Table 1: Features for FasterTransformer, TurboTransformers, LightSeq and our proposed SAMP. SAMP supports tokenizer, different combinations of mixed-precision modes and various downstream tasks.

to-end toolkit implemented by C++ programming language (from Tokenizer to Embedding, Encoder, and Downstream tasks), which has excellent inference speed and reduces the threshold of industrial application. Table 1 shows the innovative features compared with similar systems. We present the following as the key contributions of our work:

Self-Adaptive SAMP balances computational accuracy and latency performance in post-training quantization inference methods. Users can choose a mixed-precision configuration with appropriate accuracy and inference latency for different tasks. SAMP also suggests a combination of quantization modes automatically via an adaptive allocation method.

Efficiency SAMP shows better inference speedup than other inference toolkits in a wide precision range (from floating-point to fixed-point). In CLUE[2] classification task datasets, SAMP achieves up to 1.05-1.15 times speedup compared with FasterTransformer.

Flexibility SAMP covers lots of downstream tasks, such as classification, sequence labeling and text matching. And Target modules are extensible and flexible to customize. It is user-friendly and less dependent. SAMP supports both C++ and Python APIs, only requires CUDA 11.0 or above. We also provides many convenient tools for model conversion.

2 Related Work

2.1 Quantization in Neural Networks

Quantizing neural networks dates back to the 1990s (Balzer et al., 1991; Marchesi et al., 1993). In the early days, the main reason to quantize models is to make it easier for digital hardware implementation (Tang and Kwan, 1993). Recently, the research of quantizing neural networks has revived due to the success of deep learning (Guo, 2018). A slew of new quantization methods have been proposed, which are divided into two categories, post-training quantization (PTQ) and quantization-aware training (QAT), according to whether the quantization procedure is related to model training. PTQ requires no re-training and is thus a lightweight push-button approach to quantization, only calibration needed. QAT requires fine-tuning and access to labeled training data but enables lower bit quantization with competitive results (Jacob et al., 2018). However, this method is difficult to popularize in the industry, especially for many existing models that need to be retrained, and the training process is also very long. Both FasterTransformer and our SAMP use PTQ to achieve fixed-point quantization acceleration.

2.2 Kernel Fusion

Kernel fusion can improve computational efficiency by reducing the number of memory accesses, increasing cache locality and reducing kernel launch overhead (Fang et al., 2021). Especially in inference, because of no back-propagation procedure, some small adjacent operators can be fused into a larger kernel. Previous fusion methods mainly include Tensor-fusion and Layer-fusion (Vanholder, 2016). Tensor-fusion is mainly to concatenate tensors of the same shape into one tensor. Layer-fusion is to fuse operators of adjacent layers into one operator layer. Our fusion improvement of embedding kernels in SAMP is Tensor-fusion, and the operation fusion of quantization operators belongs to Layer-fusion.

3 Architecture

3.1 Overview of SAMP

In this section, we mainly introduce four modules of SAMP as shown in Figure 1: Tokenizer, Embedding, Encoder and Downstream Target, and some

[2] https://github.com/CLUEbenchmark/CLUE

innovative features make it stand out from other similar works.

Tokenizer: SAMP is a task-oriented and end-to-end inference library, which has a complete word segmentation module for Chinese and English that supports multi-granularity tokenization, such as character-based tokenization, wordpiece tokenization and general BertTokenizer. This module is implemented by C++ programming language and multi-thread processing methodology. So, its processing is faster than some Python programming language implementations in similar inference libraries.

Embedding: Current embedding method proposed by BERT (Devlin et al., 2018) is constructed by summing the corresponding token, segment, and position embeddings, which is implemented by previous work (e.g. FasterTransformer) through three independent operation kernels. We fuse these three operators into one kernel (Embedding Kernel) to reduce CUDA kernel calls, as shown in Figure 1.

Encoder: SAMP selects Transformer (Vaswani et al., 2017) as the basic component in Encoder module. Our innovative features about how to quantitatively balance the accuracy and latency performance of FP16 and INT8 are mainly realized in this module, and we propose an **Accuracy-Decay-Aware allocation** algorithm to obtain best speedup of mixed-precision while ensuring the required accuracy automatically. At the same time, the fusion improvements of quantization operators are also implemented in this module.

Downstream Target: SAMP supports a lot of models in NLP downstream tasks, including classification, multi-label, named entity recognition, text matching and so on. These capabilities and customization are implemented in Target module.

3.2 SAMP Transformer-based Encoder

In order to solve the problem of serious loss of accuracy, which exists in Fully-quantization method of FasterTransformer, SAMP divides the GEMMs in Transformer Encoder into two categories by multi-head attention (MHA) and feed-forward network (FFN) to generate two mixed-precision modes: Fully-Quant and Quant-FFN-Only. Fully-Quant means GEMMs in MHA and FFN are both quantized. Quant-FFN-Only means GEMMs only in FFN are quantized, and MHA reserves FP32/FP16 accuracy. Figure 2 illustrates the kernel details of the two mixed-precision modes in SAMP. For a

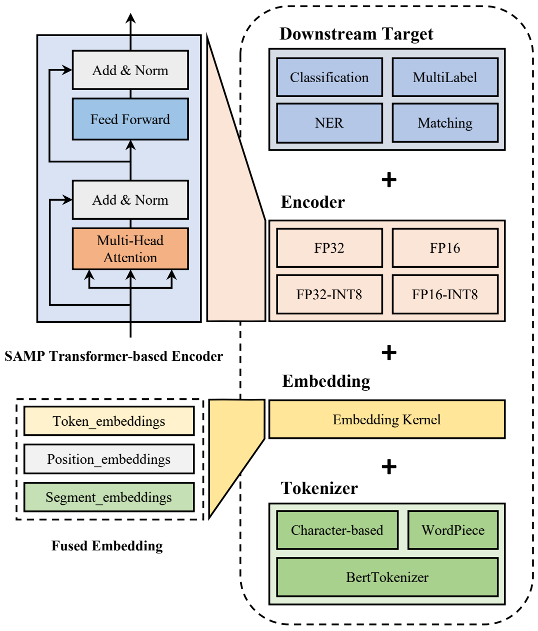


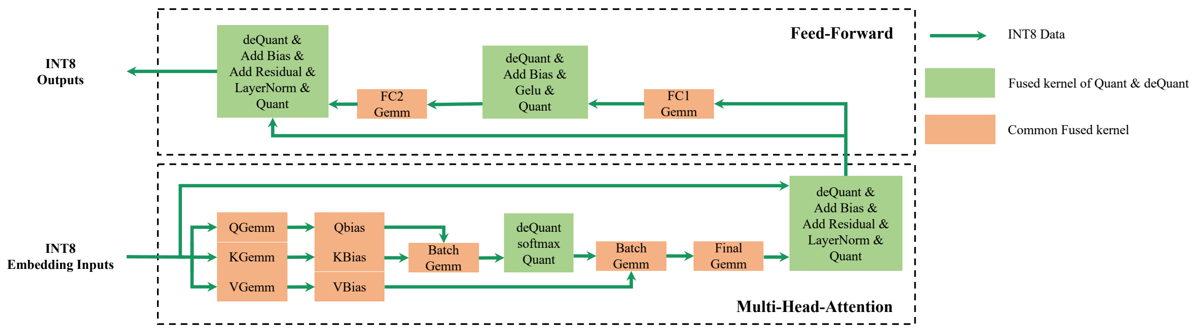
Figure 1: The architecture of SAMP.

Transformer-based model, the encoder module is usually composed of lots of Transformer layers. Assuming that the number of layers is L, there are 2L combinations of mixed-precision.

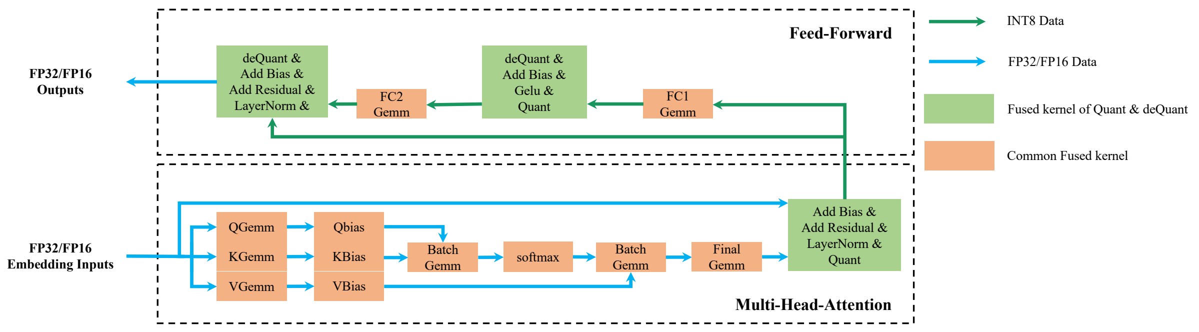
Fully-Quant mode shown in Figure 2(a) quantizes the FP32/FP16 inputs of the Encoder in Embedding module, so the data bit width between Embedding and Encoder is INT8 directly, which reduces the cost of separate quantization kernel call. In addition, we also make a big kernel fusion with Quant/deQuant operations, such as AddResidual, AddBias, and LayerNorm, so that in the whole forward calculation in Encoder, the data transmission between kernels is always 8-bit integer (all green arrows). This fusion reduces the bit width of memory I/O and the number of kernels, making the speed of SAMP INT8-quantization exceed FasterTransformer 5% $\sim$ 10%, as shown in Section 4.3.

Quant-FFN-Only mode shown in Figure 2(b) only quantized the GEMM operations in FFN. As stated above, we reserve the FP32/FP16 GEMM algorithms in MHA, and quantize the floating-point result after LayerNorm operation at the end of MHA. The INT8 GEMM algorithm in FFN is the same as that in Fully-Quant mode, and the only difference is that quantization is not used in the last big kernel to support floating-point outputs.

We now illustrate how SAMP works effectively. More details of installation and usage are described in Appendix A. For a specific task, SAMP will automatically calculate the accuracy and latency



(a) Fully-Quant mode.


Feed-Forward
INT8 Data
FP32/FP16 Data
Fused kernel of Quant & deQuant
Common Fused kernel
FP32/FP16 Outputs
deQuant & Add Bias & Add Residual & LayerNorm &
FC2 Gemm
deQuant & Add Bias & Gelu & Quant
FC1 Gemm
FP32/FP16 Embedding Inputs
QGemm
KGemm
VGemm
Qbias
KBias
VBias
Batch Gemm
softmax
Batch Gemm
Final Gemm
Add Bias & Add Residual & LayerNorm & Quant
Multi-Head-Attention

(b) Quant-FFN-Only mode.

Figure 2: Two modes of Transformer models in SAMP method. Each square above represents a CUDA kernel, including one or more function operations. Arrows indicate dataflow. For Fully-Quant mode, both Multi-Head Attention and Feed-Forward Network are INT8-quantized in Transformer, while in Quant-FFN-Only mode, only Feed-Forward Network is quantized.

Algorithm 1 Accuracy-Decay-Aware allocator

```
Input: $Array\ A$ , $L$ of Accuracy and Latency, the number
    of Transformer-layers $N$
Output: $Lq$, number of quantized Transformer-layers.
 1: $dr_{min} \leftarrow$ MAX_FLOAT
 2: $A_{fp16} \leftarrow A_0, L_{fp16} \leftarrow L_0$
 3: for $i = 0$ to $N$ do
 4:     if $i == 0$ then
 5:         $A_{rec} \leftarrow A_{fp16}$
 6:         $L_{rec} \leftarrow L_{fp16}$
 7:     else
 8:         $dr \leftarrow (A_i - A_{rec})/(L_i - L_{rec})$
 9:         if $dr < 0$ or $dr < dr_{min}$ then
10:             $dr_{min} \leftarrow dr$
11:             $A_{rec} \leftarrow A_i$
12:             $L_{rec} \leftarrow L_i$
13:             $Lq \leftarrow i$
14:         end if
15:     end if
16: end for
17: return $Lq$
```

of these mixed-precision combinations of different modes, using Fully-FP16 implementations of SAMP as baseline. Users can input specific latency and accuracy requirements before the calculation. SAMP will find the mixed-precision combination that mostly meets the requirements, and configure the mixed-precision parameters to inference toolkit automatically. When users cannot give clear requirements, SAMP will generate a set of recommended configuration parameters of mixed-precision by the **Accuracy-Decay-Aware allocation** algorithm. Specifically, in the two modes we proposed above, the speedup increases linearly with the number of quantization layers (each layer of Quant-FFN-Only mode brings $2 \sim 3\%$ speedup compared with Fully-FP16 in BERT-base binary classification), while the accuracy drops significantly after more layers of quantization. This algorithm will recommend a balance between accuracy and speedup of mixed-precision combination, as shown in Algorithm 1.

4 Experiments

4.1 Experiment Settings

In this section, we show the experimental results of SAMP from two aspects: SAMP trade-off test on text classification tasks and latency speedup. All the evaluation experiments are conducted on GPU NVIDIA Tesla T4, CUDA 11.0. Moreover, we use INT8-quantization calibration tool pytorch-

PTQ Libraries	Quantized Layer		AFQMC		IFLYTEK		TNEWS	
	MHA	FFN	Accuracy	Speedup	Accuracy	Speedup	Accuracy	Speedup
PyTorch-FP16	0/12	0/12	0.7337	1.0000	0.6048	1.0000	0.5633	1.0000
FasterTransformer-FP16	0/12	0/12	0.7340	2.9319	0.6052	1.6524	0.5634	3.1351
FasterTransformer-INT8	12/12	12/12	0.5773	3.4990	0.4540	2.4539	0.5058	3.5551
SAMP-FP16	0/12	0/12	0.7338	3.3741	0.6056	1.4870	0.5632	3.5022
SAMP-Fully-Quant	2/12	2/12	0.6671	3.5790	0.5572	1.5550	0.0930	3.6790
	4/12	4/12	0.3167	3.7689	0.2957	1.6144	0.0856	3.9083
	6/12	6/12	0.3188	4.0486	0.1454	1.7305	0.0952	4.2274
	8/12	8/12	0.6435	4.3882	0.1493	1.8645	0.0851	4.5985
	10/12	10/12	0.6874	4.7751	0.1149	2.0162	0.0900	4.9869
	12/12	12/12	0.4409	5.1817	0.0150	2.1978	0.0884	5.3271
SAMP-Quant-FFN-Only	0/12	2/12	0.7340	3.4799	0.6007	1.5073	0.5654	3.6659
	0/12	4/12	0.7318	3.6162	0.5932	1.5532	0.5640	3.7465
	0/12	6/12	0.7088	3.7725	0.5840	1.6269	0.5610	3.9527
	0/12	8/12	0.6872	4.0059	0.5786	1.7095	0.5523	4.1440
	0/12	10/12	0.5588	4.2262	0.5663	1.7863	0.5208	4.3917
	0/12	12/12	0.5279	4.4574	0.5641	1.8821	0.5077	4.6195

Table 2: SAMP test for Fine-tuned BERT-base(L12_H768) model on CLUE tasks AFQMC, IFLYTEK and TNEWS. We all use min-max calibrator of pytorch-quantization[3] to generate scales for INT8-quantization. We only show partial experimental data here due to space constraints. Compared with SAMP-FP16, Underlined scores represent a mixed-precision combination recommended by the accuracy-decay-aware allocation method in each mode.

quantization[3] of NVIDIA TensorRT , which provides four calibration methods for post-training quantization (PTQ). Users can choose an appropriate calibration method to generate scale values, which convert model weights from floating-point to fixed-point, for mixed-precision calculations.

First, we test three groups of experiments for SAMP trade-off (between accuracy and latency speedup) in text classification tasks AFQMC(Ant Financial Question Matching Corpus), IFLYTEK(Long Text classification) and TNEWS(Short Text Classification for News) in Chinese Language Understanding Evaluation Benchmark (Xu et al., 2020). We use BERT-base (12-Layer, HiddenSize-768) released by Google (Devlin et al., 2018) as the pre-trained model, and train the FP32 baseline models by the paradigm of "Pre-training and Fine-tuning" in each task. And we also use TencentPretrain (Zhao et al., 2022) as training toolkit. Finally, SAMP self-adaptively obtains the best trade-off between accuracy and latency speedup of mixed-precision on the Dev set.

Secondly, we also test SAMP latency speedup separately, choosing the popular PyTorch and the latest version of FasterTransformer for comparison. Due to the difference of Tokenizer(shown in Table 1) and programming languages in Target modules (SAMP's targets are developed by C++ programming language, and FasterTransformer uses Python targets (Fang et al., 2021)), we only make speedup comparison with Encoder.

4.2 Text Classication on CLUE

Table 2 shows the changes of accuracy and speedup with the increase of the number of quantized Transformer-layers in two modes, Fully-Quant and Quant-FFN-Only. The upper bound of speedup is All-layers Fully-Quant and lower bound is Fully-FP16. We choose PyTorch-FP16 implementation as baseline for speedup comparison. In each mode, with the increase of the number of quantized Transformer-layers, the speedup of three tasks increased steadily, while accuracy decreases faster and faster.

SAMP has three modes: SAMP-FP16, SAMP-Fully-Quant and SAMP-Quant-FFN-Only. It automatically recommends the appropriate mixed-precision combination (underlined scores in Table 2) for each task by using the accuracy-decay-aware allocation method. For example, compared with Fully-FP16, in SAMP Quant-FFN-Only mode, the AFQMC task achieves a speedup of 18.7% (4.0059 vs. 3.3741) through 8-layer FFN quantization, and the accuracy decreases by only 4.7% (0.6872 vs. 0.7338) . IFLYTEK task achieves a speedup of 26.6% and accuracy of it decreases by only 4.15%. In TNEWS task, the accuracy de-

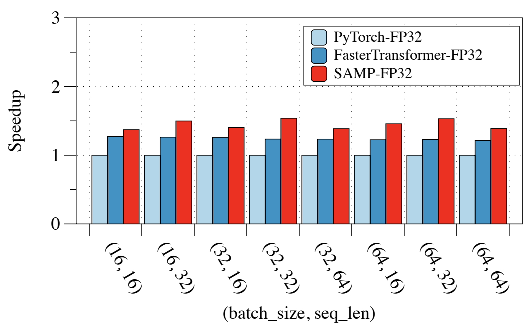
[3]https://github.com/NVIDIA/TensorRT/tree/main/tools/pytorch-quantization

creases slightly by only 0.22% in 6-layer FFN quantization, and achieves a speedup of 12.9%. These recommended results of SAMP have significantly higher accuracy than All-layers Fully-quantization in FasterTransformer, and even most of them have achieved better speedups. Finally, the balance idea between accuracy and latency of SAMP is proved to be effective significantly.

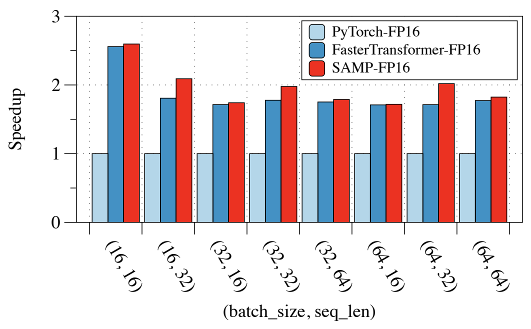
We also find an interesting phenomenon that accuracy decreases heavily in Fully-Quant mode compared with Quant-FFN-Only. The main reason for the severe accuracy loss of quantizing MHA is caused by quantizing the output of Softmax in MHA. For general neural network layers, the distribution of positive and negative outputs are almost balanced, so that the precision range of 8-bit fixed-point (-2^7 to $2^7 - 1$) can be fully used. But the output value of Softmax is between 0 and 1, so the part of -128 to 0 is unused (default in symmetric quantization, refer to Appendix B). Experimental results shows most Softmax output quantized values are distributed between 0 and 64, rather than -128 to 127. The accuracy loss of quantizing Softmax output accumulates when Transformer layer gets deeper, resulting in the overall severe accuracy loss of SAMP Fully-Quant and FasterTransformer. So, **Quant-FFN-Only is the preferred mode** recommended by SAMP.

4.3 Speedup of SAMP

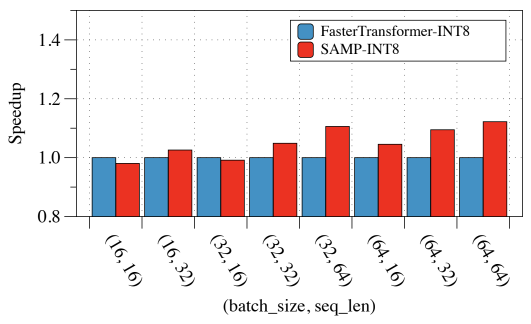
For our kernel-fusion improvements, we also make a latency benchmark comparison for fully floating-point and fixed-point, including Fully-FP32, Fully-FP16 and Fully-INT8. As shown in Figure 3, SAMP floating-point Encoder achieves higher speedups than PyTorch and FasterTransformer in common batch size and length of sequence respectively. These histogram tables show that SAMP-FP32 achieves up to 1.5x speedup compared with PyTorch and 1.1x compared with FasterTransformer, and SAMP-FP16 achieves up to 2x speedup compared with PyTorch and 1.15x compared with FasterTransformer-FP16. Meanwhile, SAMP-Fully-INT8 achieves up to 1.1x speedup compared with FasterTransformer-INT8 in common application scenarios. These comparisons demonstrate that SAMP has been optimized as a new inference tool with faster floating-point and fixed-point computations for Transformer-based Encoder.



(a) Speedup in Fully-FP32.



(b) Speedup in Fully-FP16.



(c) Speedup in Fully-INT8.

Figure 3: Encoder speedup on GPU Tesla T4 compared with FasterTransformer and PyTorch.

5 Conclusion

In this paper, we introduce a new inference toolkit SAMP for NLP models. The main contribution of SAMP is to solve the problem of serious performance loss of the existing quantization inference tools in the industrial application of text understanding. And it also pioneers the application of quantization inference to various downstream tasks through a wide variety of task-type coverage. SAMP is light-weight, flexible, and user-friendly. At present, it has been widely used in our business, which greatly saves the deployment cost of industrial applications.

In the future work, we will focus on optimizing the quantization effect of GEMMs in MHA, and explore fixed-point acceleration methods with lower bit width than 8-bit integer, and introduce SAMP to more models.

Limitations

We propose a high-performance quantization inference toolkit SAMP, but it inevitably contains some limitations as:

- SAMP is an end-to-end inference toolkit implemented by C++ programming language. Compared with most toolkits of Python programming language, the flexibility of it is limited, and users are required to have some basic knowledge or experience in C++ language development. Based on that, we have provided a lot of convenient Python APIs for ordinary users.
- In different series of GPU architectures, the library files of SAMP need to be re-compiled. Users familiar with Nvidia architectures of Data Center know that the construction and compilation of these basic environments are essential.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pages 265–283. Savannah, GA, USA.

Wolfgang Balzer, Masanobu Takahashi, Jun Ohta, and Kazuo Kyuma. 1991. Weight quantization in boltzmann machines. *Neural Networks*, 4(3):405–409.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jiarui Fang, Yang Yu, Chengduo Zhao, and Jie Zhou. 2021. Turbotransformers: an efficient gpu serving system for transformer models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 389–402.

Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Young Jin Kim and Hany Hassan Awadalla. 2020. Fastformers: Highly efficient transformer models for natural language understanding. *arXiv preprint arXiv:2010.13382*.

Michele Marchesi, Gianni Orlandi, Francesco Piazza, and Aurelio Uncini. 1993. Fast neural networks without multipliers. *IEEE transactions on Neural Networks*, 4(1):53–62.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Chuan Zhang Tang and Hon Keung Kwan. 1993. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8):2724–2727.

Han Vanholder. 2016. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, page 2.

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.

Xiaohui Wang, Ying Xiong, Yang Wei, Mingxuan Wang, and Lei Li. 2021. LightSeq: A high performance inference library for transformers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers (NAACL-HLT)*, pages 113–120. Association for Computational Linguistics.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, et al. 2020. Clue: A chinese language understanding evaluation benchmark. *arXiv preprint arXiv:2004.05986*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019.

Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Zhe Zhao, Yudong Li, Cheng Hou, Jing Zhao, Rong Tian, Weijie Liu, Yiren Chen, Ningyuan Sun, Haoyan Liu, Weiquan Mao, et al. 2022. Tencentpretrain: A scalable and flexible toolkit for pre-training models of different modalities. *arXiv preprint arXiv:2212.06385*.

A Installation and Usage

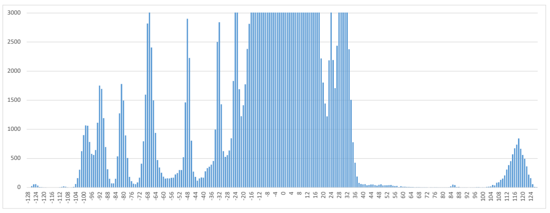
SAMP is a high performance inference toolkit with less dependencies and high compatibility. Prerequest of SAMP only includes CMake >= 3.13, GCC >= 8.3 and CUDA >= 11.0. To run calibration, SAMP provides calibration tools depending on PyTorch >= 1.7.0 and NVIDIA pytorch-quantization. To install SAMP, users only need to sepcify the GPU compute capability, make a 'build' directory and run CMake and Make. The executable files will be generated under 'build/bin' directory.

SAMP provide a user-friendly end-to-end inference usage. First, we can use calibration tools to load the pre-trained language model weights of HuggingFace style and run the calibration process, and dump the weights into a format required by CUDA. Second, to run self-adaptive mix-precision, SAMP provides some scripts that calibrate the model of different quantization layer settings and recommands the high accuracy and low latency ones. Under normal conditions, with the number of quantized Transformer-layers gets higher, the model tends to have lower latency but suffers higher accuracy loss. Users can set required highest time cost threshold or lowest accuracy threshold. If highest time cost threshold is set, SAMP will recommand the setting with the highest accuracy whose time cost is lower than the threshold. If the lowest accuracy threshold is set, SAMP will recommand the setting with the lowest time cost whose accuracy is higher than the threshold. If neither is set, SAMP will recommand top-5 appropriate settings based on the ratio of speedup / accuracy-loss.

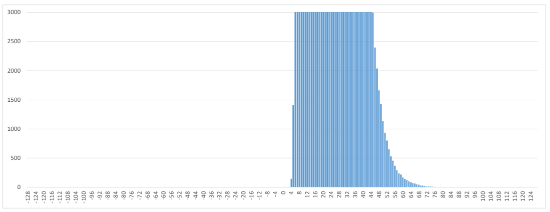
B Loss in quantization after Softmax

Softmax operation is computed during self-attention in transformer models. The output of such operation is quite different from output of other layers. Common quantization methods usually multiply the floating point numbers by a scale and round them into an integer. The rounding operation makes some different floating point numbers rounded into a same integer, which causes the accuracy loss. To reduce such loss, calibration methods try to find the appropriate scales that makes the quantized integer in -128 to 127 as well-distributed as possible. The output values of Softmax operation are between 0 and 1. On the premise of symmetric quantization, the part of -128 to 0 is unused. In addition, in the output matrix of Softmax, the sum of the elements in the same row is 1, so the attention-softmax output matrix in short sequences tend to have larger element values. Since the scale in the same layer is pre-computed in calibration process and is fixed in inference process, it is unable to reconcile the distribution of Softmax output in short sequences and long sequences. We counted the distribution of Attention-Softmax outputs, and take the distribution of MHA output as a comparison. Figure-4 shows that the distribution of quantized Attention-Softmax outputs are squeezed in a narrow space of 0-64, while the quantized output of MHA is distributed in -128 to 127. The accuracy loss of quantized Attention-Softmax outputs accumulate when Transformer layers get deeper, resulting in the overall severe accuracy loss of Fully-quantized SAMP and FasterTransformer.

In the quantized Attention-Softmax outputs, the unused number of INT8-integer is 67.58%, while in the quantized outputs of MHA, the number is only 4.30%.



(a) Distribution of quantized MHA outputs



(b) Distribution of quantized Attention-Softmax outputs.

Figure 4: Distribution of quantized MHA output and quantized Attention-Softmax outputs. We count the distribution on 64 sequences of TNEWS classification data. X-axis represents the quantized INT-8 integer values, and Y-axis represents the number of output elements that use the corresponding INT-8 integer values.