QUANTPIPE: APPLYING ADAPTIVE POST-TRAINING QUANTIZATION FOR DISTRIBUTED TRANSFORMER PIPELINES IN DYNAMIC EDGE ENVIRONMENTS

Haonan Wang\textsuperscript{1 2}, Connor Imes\textsuperscript{2}, Souvik Kundu\textsuperscript{1 3}, Peter A. Beerel\textsuperscript{1 2}, Stephen P. Crago\textsuperscript{1 2}, John Paul Walters\textsuperscript{2}

\textsuperscript{1}Department of Electrical and Computer Engineering, University of Southern California, USA\textsuperscript{2}Information Sciences Institute, University of Southern California, USA\textsuperscript{3}Intel Labs, USA

ABSTRACT
Pipeline parallelism has achieved great success in deploying large-scale transformer models in cloud environments, but has received less attention in edge environments. Unlike in cloud scenarios with high-speed and stable network interconnects, dynamic bandwidth in edge systems can degrade distributed pipeline performance. We address this issue with QuantPipe, a communication-efficient distributed edge system that introduces post-training quantization (PTQ) to compress the communicated tensors. QuantPipe uses adaptive PTQ to change bitwidths in response to bandwidth dynamics, maintaining transformer pipeline performance while incurring limited inference accuracy loss. We further improve the accuracy with a directed-search analytical clipping for integer quantization method (DS-ACIQ), which bridges the gap between estimated and real data distributions. Experimental results show that QuantPipe adapts to dynamic bandwidth to maintain pipeline performance while achieving a practical model accuracy using a wide range of quantization bitwidths, e.g., improving accuracy under 2-bit quantization by 15.85% on ImageNet compared to naive quantization.

Index Terms—Distributed, Edge System, Pipeline Parallelism, Post-training Quantization, Adaptive

1. INTRODUCTION

Recently, transformer models \cite{1–3} have achieved high accuracy in many natural language processing \cite{4} and computer vision tasks \cite{5, 6}. However, this high accuracy come at the cost of extremely large size. For example, Megatron-LM has 8.3 billion parameters \cite{7} and GPT-3 has 175 billion parameters \cite{8}. To alleviate this issue, many distributed parallelism training strategies have been proposed to use decentralized cloud compute resources and accelerate the training process. For instance, the Parameter Server \cite{9} approach is a typical data-parallel method that trains multiple duplicated models with different data sets in parallel. Another model-parallel method \cite{10} slices a large-scale model into several sub-models that are small enough to fit on single nodes. The data that flows across the slicing boundary now must be transmitted through the communication interface between devices. A special case of model parallelism is pipeline parallelism, which partitions the model into consecutive shards and pipelines the execution of these shards across devices. Pipeline parallelism is straightforward, but efficient, and thus widely adopted by many applications \cite{11–13}.

Although pipeline parallelism has made considerable progress in training large models on the cloud, few of works focus on empowering inference at the edge. It is vital for many applications, since they may not be able to leverage the computation power on the cloud due to constraints on latency, privacy, or an unreliable (or non-existent) link to the cloud \cite{14}. For example, to enable a real-time detection task on a drone formation, it may be more reliable to process the task using processors of multiple drones than offloading the task to the cloud with long round-trip latency. Distributed edge inference systems have therefore been proposed \cite{14, 15}. However, when high-speed and stable bandwidths are not guaranteed, these systems may fail to achieve the expected performance. Fig. 1 demonstrates how the overall throughput suffers when communication bandwidth between individual stages is reduced. System performance can no longer be improved solely by further refinements to the partition strategy — communication between nodes must be optimized.

We address this need with QuantPipe\textsuperscript{1}, a communication-efficient distributed edge pipeline system using post-training quantization (PTQ). To the best of our knowledge, this is the first work that leverages PTQ in a distributed pipeline system specifically to compress the communication of activations between pipeline stages. QuantPipe prevents bottlenecks caused by stages suffering from performance degradation resulting

\textsuperscript{1}Code available at https://github.com/usc-isi/PipeEdge.
from intermittent bandwidth reductions in dynamic edge environments. We leverage the analytical clipping for integer quantization (ACIQ) method [16] to improve inference accuracy over a naive PTQ approach. However, we find that ACIQ still results in considerable accuracy loss under 2-bit PTQ. We find that ACIQ fails to identify the gap between the estimated and real distributions. Thus, we propose a directed-search ACIQ method (DS-ACIQ) to bridge this gap by searching for a better estimation closer to the real data distribution. Our PTQ with DS-ACIQ method (PDA) yields an accuracy improvement of 15.85% on ImageNet, and for the first time puts 2-bit PTQ into practical implementation. We also design an adaptive PDA module that responds to network dynamics. When bandwidth fluctuation is detected, QuantPipe applies PDA using the highest quantization bitwidth that can achieve the target performance under the current bandwidth.

2. BACKGROUND AND MOTIVATION

Pipeline parallelism is a straightforward but efficient parallel computing paradigm that partitions a computing task into subtasks and pipelines their execution across devices. Several works [11–13] already prove it can accelerate training of large models on the cloud. However, pipeline parallelism is still poorly developed for edge environments, and its downside — the overall performance is bounded by the slowest stage — is amplified by the intrinsic property of unstable connections in the edge environment. If any stage in the pipeline is blocked by communication due to network fluctuation, it will become the bottleneck and dominate the overall performance.

Compressing communication in a pipeline architecture can help limit or avoid communication bottlenecks. Most existing approaches [17, 18] focus on accelerating model training on the cloud, where connections are relatively fast and stable compared to edge environments. Furthermore, the pruning and knowledge distillation techniques used in these solutions require additional training prior to deployment, which is too compute-intensive for edge applications. PTQ [16] is one of the most common model compression methods that can be deployed at runtime without any training efforts. However, current works [16, 19] only use PTQ to compress the model size to fit on a single device.

These challenges motivate applying PTQ to the distributed pipeline edge system. The insight is that when communication becomes the new bottleneck due to network fluctuation, applying PTQ for communication compression can prevent performance degradation. We will use the popular ViT model [2] for performance evaluation of the pipeline system, since it has a layer-wise concatenated structure without inter-layer connections, making it suitable to be partitioned at layer boundaries in a pipeline architecture.

3. SYSTEM DESIGN AND IMPLEMENTATION

As shown in Fig. 2, QuantPipe introduces an adaptive PDA module in a pipeline system. If network fluctuation is detected at any stage, the adaptive PDA module compresses the activations out of the previous stage to reduce communication by representing data with fewer bits. For example, the communication will be compressed by $4 \times$ using 8-bit quantization. The adaptive PDA process consists of three steps. It first performs the directed search method to get an accurate estimation of the real data distribution and then calculates the optimal ACIQ clipping range based on the estimated distribution parameter. After applying ACIQ clipping to the activation data, the adaptive PDA module applies PTQ using the bitwidth that can achieve the predefined target output rate $R$. The required bitwidth is determined by the current measured bandwidth and some other support information collected by the runtime monitor. The model is evenly partitioned by an optimal partition algorithm [15]. For efficient implementation, each model shard will be assigned to only one device to avoid the overhead of collective synchronization.

**Naive PTQ and ACIQ.** QuantPipe uses a PTQ compression technique to achieve expected performance under limited bandwidth. We apply uniform quantization, which divides the data range into equal intervals for rounding. However, we find that the naive PTQ method that determines the quantization range based on the minimum and maximum tensor values can result in poor model accuracy. As shown in Fig. 3, the

---

**Fig. 2:** Overview of the QuantPipe system. An adaptive PDA module is introduced for communication compression.
naive quantization interval causes significant precision loss for relatively small values. To quantify, the mean squared error (MSE) between the original tensor and the quantized tensor shows that PTQ is large and heavily influenced by outliers. We therefore adopt the ACIQ clipping method [16] which provides a theoretical optimal clipping range that minimizes the MSE after quantization, thus limiting the impact of outliers [19]. Fig. 3 visualizes the difference between naive PTQ and ACIQ. Precision loss is more severe for those layers with extremely large variance (e.g., at the 6th layer) where the naive PTQ derives a larger quantization interval and loses almost all information for relatively small values, resulting in worse quantization accuracy. ACIQ assumes the activation follows a Laplace distribution \( L(\mu, b) \), and the optimal clipping range \( \alpha \) is determined by \( \alpha = F(q) \cdot b \), where \( b \) is the scale parameter of Laplace estimated from the real data, and \( F(q) \) is a lookup function based on the quantization bitwidth \( q \). With ACIQ clipping, most of the outliers are clamped and the quantization range is confined such that the quantization interval is still accurate enough to represent the distribution without rounding most of the data to zero.

**Directed Search.** However, ACIQ still results in significant accuracy loss under small bitwidths, like 2-bit quantization. Empirically, it results from the gap between the distribution estimated by ACIQ and the original distribution, as shown in Fig. 4. Therefore, we propose a directed-search ACIQ (DS-ACIQ) approach to bridge this gap by searching for a better-estimated scale factor \( b^* \) that minimizes MSE along the direction toward the real data. DS-ACIQ collects the histogram information of the original data \( D_R \) and compares it with the estimated distribution \( D_E \), which is derived from the estimated scale factor \( b_E = \sum_i \frac{\text{abs}(X)}{N} \) for tensor \( X \) with size \( N \). If \( \max(D_R) < \max(D_E) \), then it will solve the optimization problem

\[
\text{argmin}_{b^* \in [b_E, b_R]} \text{MSE}(D_R, D_E)
\]

by numerically searching \( b^* \) in the increasing direction with \( t \) steps, vice versa, where \( b_R = \left[ 2 \times \max(D_R) \right]^{-1} \) is the search boundary determined by the peak of the real distribution. \( t \) is heuristically set as 100 in experiments. It either finds the parameter \( b^* \) that gives a lower MSE or otherwise uses \( b_E \).

Fig. 4 shows that DS-ACIQ decreases the MSE by around 50%. DS-ACIQ compute overhead averages less than 1% in deployment.

Table 1 reports the average accuracy for all ViT-Base model partitions. We implement naive PTQ and ACIQ [16] over our pipeline framework for a fair comparison. By leveraging DS-ACIQ, PDA limits accuracy loss to an acceptable level compared to naive PTQ. The slight accuracy decrease under 6, 8, and 16 bits when compared with the ACIQ method results from the inconsistency between the intermediate minimal-MSE representation and the final output. Although the difference is minor, the DS-ACIQ approach is only activated under 4- and 2-bit quantization. For the first time, our PDA achieves a practical level of accuracy under 2 bits, outperforming that of the ACIQ method by 15.85%.

**Adaptive PDA.** QuantPipe’s adaptive PDA module monitors the output bandwidth \( B_i \) on each stage \( i \) (\( i = 1 \ldots n \)). If a significant change in bandwidth is detected at any stage \( i \), it will estimate the bitwidth \( q_{k,t+1} \) required to achieve the pre-defined target sending rate \( R \) under current bandwidth \( B_{k,t} \) and bitwidth \( q_{k,t} \) at the \( t \)-th iteration:

\[
q_{k,t+1} = \frac{32}{2^{\left(\log\left(\frac{V_{k,t} \times 32}{B_{k,t}}\right)\right)}}
\]

\( V_{k,t} \) represents the size of quantized data using bitwidth of \( q_{k,t} \) out from stage \( i \), and \( S \) denotes the microbatch size. Thus, QuantPipe adaptsively changes the quantization bitwidth to prevent performance degradation under dynamic network conditions.

**Fig. 3:** Distribution of the original data (top), after naive PTQ (middle), and after PTQ with ACIQ (bottom) using the ViT-Base model partitioned after 4th (left) and 6th (right) block.

**Fig. 4:** Estimated distribution using ACIQ with and without directed search after 4th (left) and 6th (right) block.

---

**Table 1: Average ViT-Base model accuracy with ImageNet.**

|       | 32bit | 16bit | 8bit | 6bit | 4bit | 2bit |
|-------|-------|-------|------|------|------|------|
| PTQ   | 80.25%| 75.74%| 75.03%| 30.29%| 0.44%|      |
| ACIQ  | 80.03%| 79.35%| 78.87%| 76.46%| 34.97%|      |
| PDA   | 78.94%| 78.72%| 78.21%| 77.34%| 70.82%|      |

---

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
4. EXPERIMENTAL EVALUATION

4.1. Experimental Setup

Our evaluation environment is a testbed with 6 NVIDIA Jetson AGX Orin devices. Each device has a 12-core ARM CPU, a 1792-core GPU, and runs Linux kernel 5.10.65-tegra. We change bandwidth between devices using Linux traffic control tools (tc). We implement QuantPipe on top of the PipeEdge distributed edge computing framework [15] using Python 3.8 and PyTorch 1.12.

4.2. Adaptive Quantization to Dynamic Bandwidth

We demonstrate QuantPipe adapting quantization bitwidth between two nodes in response to dynamic bandwidth to maintain a stage output performance constraint. It is sufficient to show results using two nodes since the analysis will be similar if the communication bottleneck is located between two stages in a larger deployment. As is common in adaptive runtime systems, QuantPipe measures relevant metrics over a window period, then makes an adaptive decision based on the window average values. We set a window period of 50 microbatches, a microbatch size of 64 images, and a stage minimum target output rate of 100 images/sec. At roughly 200-microbatch intervals, we change the bandwidth between the devices. Importantly, QuantPipe is not informed of this change but relies on its own runtime measurements.

Fig. 5 demonstrates QuantPipe adapting to changing bandwidth in five phases. Based on the measured bandwidth, QuantPipe’s adaptive PDA module updates bitwidth after every window period. In Phase 0, bandwidth is high and QuantPipe performance exceeds 100 images/sec. In Phases 1 and 2, we change the bandwidth between stage1 and stage2 to 400 and 50 Mbps, respectively. On stage1, QuantPipe measures that the output rate falls below the constraint value. When the window period expires, PDA updates the bitwidth to 16 and then 2 bits. The stage1 output rate then recovers to again satisfy the target output rate, which also results in a corresponding recovery in the performance of the entire pipeline. Even under 2-bit quantization, only a tiny reduction in pipeline accuracy is incurred. In Phase 3, we increase the bandwidth to 200 Mbps. The stage1 output rate grows to almost 4× the target output rate, but the overall system performance does not improve since bandwidth is no longer the bottleneck. PDA responds by switching from 2-bit to 6-bit quantization (as the bandwidth measurements lag behind the change), and then finally to 8-bit, which satisfies the target output rate constraint while maximizing the bitwidth and thus pipeline accuracy. In Phase 4, we remove the bandwidth limitation and the system returns to its nominal state, i.e., running without quantization. Throughout all phases, model accuracy remains at an acceptable level, even under 2-bit quantization. The experiment results illustrate that QuantPipe is adaptive and resistant to network fluctuation.

5. CONCLUSION

In this paper, we propose QuantPipe, a post-training quantization (PTQ) paradigm for communication compression in distributed transformer pipelines in dynamic edge environments. We introduce DS-ACIQ, a quantization method to improve accuracy loss caused by small-bitwidth PTQ. We empirically demonstrate adaptive quantization to recover from performance degradation caused by network fluctuations, with only limited accuracy loss.

Acknowledgments

This work was supported by the Department of the Navy, Office of Naval Research under, NSF, and DARPA with grant #N00014-20-1-2143, #1763747, and #HR00112190120, respectively. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies. This work was supported in parts by NSF and DARPA with grant numbers 1763747 and HR00112190120, respectively.
6. REFERENCES

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, “Attention is all you need,” Advances in neural information processing systems, vol. 30, 2017.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” arXiv preprint arXiv:2010.11929, 2020.

[3] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in Proceedings of naacL-HLT, 2019, pp. 4171–4186.

[4] Merity Stephen, Xiong Caiming, Bradbury James, and Richard Socher, “Pointer sentinel mixture models,” in 5th International Conference on Learning Representations (ICLR), 2017.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in 2009 IEEE conference on computer vision and pattern recognition, 2009, pp. 248–255.

[6] Haonan Wang, Yuchen Mei, Jun Lin, and Zhongfeng Wang, “Temporal residual feature learning for efficient 3d convolutional neural network on action recognition task,” in 2020 IEEE Workshop on Signal Processing Systems (SiPS). IEEE, 2020, pp. 1–6.

[7] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro, “Megatron-lm: Training multi-billion parameter language models using model parallelism,” arXiv preprint arXiv:1909.08053, 2019.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al., “Language models are few-shot learners,” Advances in neural information processing systems, vol. 33, pp. 1877–1901, 2020.

[9] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu, “Communication efficient distributed machine learning with the parameter server,” Advances in Neural Information Processing Systems, vol. 27, 2014.

[10] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen, “Gshard: Scaling giant models with conditional computation and automatic sharding,” arXiv preprint arXiv:2006.16668, 2020.

[11] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiqun Ngiam, Quoc V Le, Yonghui Wu, et al., “Gpipe: Efficient training of giant neural networks using pipeline parallelism,” Advances in neural information processing systems, vol. 32, 2019.

[12] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia, “PipeDream: Generalized pipeline parallelism for DNN training,” in Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 1–15.

[13] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr, “PipeTransformer: Automated elastic pipelining for distributed training of large-scale models,” in International Conference on Machine Learning. PMLR, 2021, pp. 4150–4159.

[14] Tuo Zhang, Chaoyang He, Tianhao Ma, Lei Gao, Mark Ma, and Salman Avestimehr, “Federated learning for internet of things,” in Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, 2021, pp. 413–419.

[15] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A. Beerel, Stephen P. Crago, and John Paul Walters, “Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices,” in 2022 25th Euromicro Conference on Digital System Design (DSD), 2022, pp. 298–307.

[16] Ron Banner, Yury Nahshan, and Daniel Soudry, “Post training 4-bit quantization of convolutional networks for rapid-deployment,” Advances in Neural Information Processing Systems, vol. 32, 2019.

[17] Yu-jun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” arXiv preprint arXiv:1712.01887, 2017.

[18] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie, “Communication-efficient federated learning via knowledge distillation,” Nature communications, vol. 13, no. 1, pp. 1–8, 2022.

[19] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang, “Improving neural network quantization without retraining using outlier channel splitting,” in International conference on machine learning. PMLR, 2019, pp. 7543–7552.