Communication-Efficient Model Parallelism for Distributed In-Situ Transformer Inference

Yuanxin Wei, Shengyuan Ye, Jiazhi Jiang, Xu Chen, Dan Huang*, Jiangsu Du*, Yutong Lu
School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
{weiyx25, yeshy8, jiangjzh6}@mail2.sysu.edu.cn,
{chenxu35, huangd79, dujiangsu, luyutong}@mail.sysu.edu.cn

***Abstract*—Transformer models have shown significant success in a wide range of tasks. Meanwhile, massive resources required by its inference prevent scenarios with resource-constrained devices from in-situ deployment, leaving a high threshold of integrating its advances. Observing that these scenarios, e.g. smart home of edge computing, are usually comprise a rich set of trusted devices with untapped resources, it is promising to distribute Transformer inference onto multiple devices. However, due to the tightly-coupled feature of Transformer model, existing model parallelism approaches necessitate frequent communication to resolve data dependencies, making them unacceptable for distributed inference, especially under weak interconnect of edge scenarios.**

In this paper, we propose DeTransformer, a communication-efficient distributed in-situ Transformer inference system for edge scenarios. DeTransformer is based on a novel block parallelism approach, with the key idea of restructuring the original Transformer layer with a single block to the decoupled layer with multiple sub-blocks and exploit model parallelism between sub-blocks. Next, DeTransformer contains an adaptive placement approach to automatically select the optimal placement strategy by striking a trade-off among communication capability, computing power and memory budget. Experimental results show that DeTransformer can reduce distributed inference latency by up to 2.81× compared to the SOTA approach on 4 devices, while effectively maintaining task accuracy and a consistent model size.

***Index Terms*—Model Parallelism, Distributed Transformer Inference, Edge Computing, Deep Learning System**

I. Introduction

Transformer models [1] has become the bedrock of various tasks, e.g. natural language processing [2], and are increasingly gaining significance within the AI community. Due to the resource-hungry feature of Transformer inference tasks, it is traditional to take cloud-assisted approaches in edge scenarios, relying on the backbone network and remote servers. By contrast, the in-situ processing in edge devices enjoys the following benefits compared to cloud-assisted deployment: (1) *Enhanced privacy and security*: the risk of unauthorized access or privacy breaches can be reduced by keeping data on local devices instead of sending data to external servers; (2) *Improved efficiency and robustness*: deploying Transformer models locally allows for faster inference without the need to communicate with remote servers, preventing the poor latency case caused by fluctuated network conditions; (3) *Ease of pressure on the backbone network and cloud center*: by local deployment, only a minimal amount of data needs to be transmitted to remote facility, thereby reducing network traffic and lightening the server load.

However, as Transformer models scale up to achieve remarkable performance, it is non-trivial to deploy them on a single local device with limited computing power and memory. Alternatively, we observe that typical edge scenarios usually comprise a variety of idle trusted devices, this presents an opportunity to distribute Transformer inference across multiple adjacent devices, thereby leveraging additional resources for improved performance. Unfortunately, it is challenging to take existing model parallelism approaches for distributed in-situ Transformer inference. This is because Transformer models are tightly-coupled that its multi-head attention mechanism jointly aggregates the results of all heads into the same linear function. Thus, it always takes frequent communications to remove data dependencies when breaking up linear layers onto multiple devices. The current state-of-the-art model parallelism approach, tensor parallelism in Megatron-LM [5], requires two all-reduce communications in a single Transformer layer. This introduces a level of overhead that is often unacceptable under bandwidth-limited edge scenarios.

In this paper, we propose **DeTransformer**, a system designed for distributed in-situ Transformer inference in edge scenarios, with the primary focus on minimizing communication overheads. DeTransformer firstly introduces a novel model parallelism approach, called block parallelism (BP) approach. BP will be applied to the majority of original Transformer layers, with the key idea of restructuring the original Transformer layer with a single block to the decoupled layer with multiple sub-blocks, so as to expose inter-block parallelism without data dependency. Next, to better leverage the model parallelism of decoupled models, an adaptive placement approach is proposed to automatically search for the optimal model-to-device mapping. This approach takes latency reduction as the primary goal and considers constraints including communication capability, computing power and memory budget. Main contributions are summarized as follows:

- We reveal the communication bottleneck when applying existing model parallelism approaches to distributed Transformer inference through preliminary experiments and hot-spot analysis across edge devices.
- We introduce a novel communication-efficient model parallelism approach, block parallelism (BP), which decouples the majority of original Transformer layers and

*Corresponding authors.

979-8-3503-4859-0/DATE24/© 2024 EDAA

greatly enhances the model parallelism.

- We devise an adaptive placement approach to search for the optimal model-to-device mapping. It includes an offline profiling procedure and an expertly crafted search space for deciding the placement with the least latency.
- We implement DeTransformer prototype system and evaluate it in realistic computing testbeds.

Experimental results demonstrate a speedup of up to 2.81× compared to the state-of-the-art model parallelism approach when distributing across 4 devices. Meanwhile, DeTransformer achieves promising accuracy results on downstream tasks, equivalent to or even surpassing the original Transformer model, without enlarging the model size. This showcases its significant potential in distributed Transformer inference. In this paper, we use notations in Table I to elaborate on our methods.

TABLE I: Notation

sq	Length of input sequence	N_b	Number of original blocks
h	Hidden size	N_h	Number of Attention heads
bs	Batch size	N_{div}	Number of divisions
l	Number of Transformer layers	N_{dev}	Number of devices

II. Background and Motivation

A. Distributed inference of Transformer Models

When distributing Transformer inference tasks across multiple devices, there are essentially three primary parallelism approaches, i.e. data parallelism (DP) [3], pipeline parallelism (PP) [4], and model parallelism (MP) [5] [6]. As in Table II, these approaches differ in terms of latency reduction, throughput increase, and memory reduction. DP keeps a complete model copy in each device and distributes requests across devices. PP divides the model by layers, assigning distinct sets of consecutive layers to different devices. These two approaches cannot achieve multi-device parallelism within a single inference request, making it unable to reduce latency.

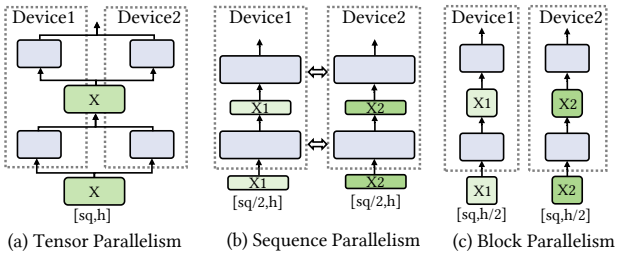


Fig. 1: The illustration of different parallelism approaches.

MP divides each layer's workload and distributes them across devices. Given that each layer is processed in parallel by several devices at a time, it can reduce the latency of a single request. Thus, MP stands out as the most promising approach in edge scenarios. Since the multi-head attention mechanism jointly aggregates the results of all heads into the same linear function, existing model parallelism approaches take communications to eliminate the data dependencies. There are two existing popular model parallelism approaches, e.g. tensor parallelism (TP) [5] and sequence parallelism (SP) [6]. In Fig. 1, TP involves the partitioning of a layer's weight, whereas SP involves the partitioning of a layer's input. They have distinct communication and memory utilization characteristics. When distributing a single Transformer layer, TP necessitates two *allreduce* communications and retains a portion of the weight, whereas SP necessitates two *allgather* communications and preserves the entire weight. By contrast, our block parallelism (BP) approach tends to distribute the workload of the decoupled Transformer layer without necessitating communication.

TABLE II: Attributes of different parallelism approaches.

	DP	PP	MP		
			TP	SP	BP
Latency Reduction	×	×	✓	✓	✓
Throughput Increase	✓	✓	✓	✓	✓
Memory Reduction	×	✓	✓	×	✓

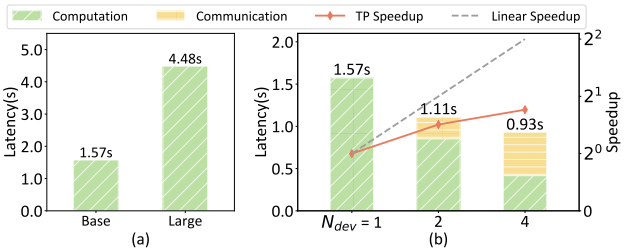


Fig. 2: (a) Inference latency on single device of Bert-Base and Bert-Large. (b) Latency and speedup when applying TP on Bert-Base across N_{dev} devices.

B. Communication Bottleneck in Model Parallelism

At first, to demonstrate the necessity of introducing distributed inference, we conduct an experiment involving single-request inference of Bert-Base and Bert-Large [2] in Fig. 2 (a). A single Raspberry Pi 4B device takes 1.57s and 4.48s respectively, which significantly deviates from meeting the low-latency demands in edge scenarios.

Next, we take TP as an example to illustrate that applying existing model parallelism approaches leads to significant communication bottleneck and weak scaling ability during distributed inference. As illustrated in Fig. 2 (b), which shows the performance of scaling Bert-Base inference from 1 to 4 devices under bandwidth of 500Mbps, the communication overhead takes a significant portion in the overall inference latency and seriously hinders the scalability of distributed inference.

III. System and Methodologies

A. Overview

Our design aims to perform low-latency distributed Transformer inference over multiple resource-constrained devices. Fig. 3 shows the workflow of our proposed DeTransformer system, which features three phases: *Preliminary training* (**Phase 1**), *Adaptive placement* (**Phase 2**) and *Distributed inference* (**Phase 3**). Under the block parallelism approach, Phase 1 is responsible for decoupling the original Transformer model and training the resultant decoupled model with the advanced GPU cluster. Next, Phase 2 generates the placement plan of the decoupled model with our adaptive placement approach. This

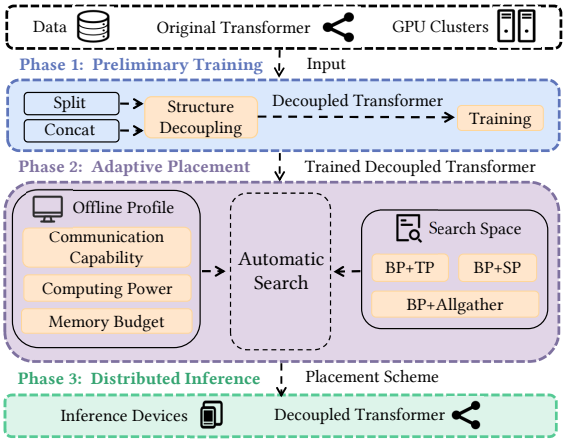
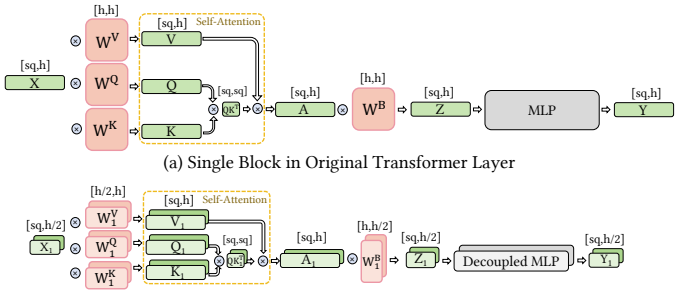


Fig. 3: DeTransformer Overview: A three phase workflow.

placement approach includes an offline profiling procedure and an expertly crafted search space. By balancing communication capability, computing power and memory budget, it generates the placement plan for the decoupled Transformer model that targets on reducing the latency. In Phase 3, guided by the placement plan generated in Phase 2, we deploy the decoupled model trained in Phase 1 onto multiple devices.

B. Block Parallelism through Structure Decoupling



(a) Single Block in Original Transformer Layer

(b) Multiple Sub-Blocks in Decoupled Transformer Layer

Fig. 4: Original and decoupled Transformer layers.

DeTransformer tends to partition the Transformer model with reduced data dependency, thereby introducing less communication overhead during distributed inference. With the concept of co-design, block parallelism (BP) firstly decouples existing Transformer models and then harnesses enhanced model parallelism capabilities.

Our decoupled Transformer model also consists of stacks of Transformer layers, and we start by describing the decoupled layer, which is the basic building unit inherited from the original layer. Fig. 4(a) illustrates an original Transformer layer, which can be formulated as Equation (1), where $[.|.]$ is concatenate operation.

$$\begin{aligned} &[Q|K|V] = [W^Q|W^K|W^V] \cdot X \\ &A = \text{Self-Attention}(Q, K, V) \\ &Z = A \cdot W^B \\ &Y = \text{MLP}(Z) \end{aligned} \tag{1}$$

$$\begin{aligned} &[Q_i|K_i|V_i] = [W_i^Q|W_i^K|W_i^V] \cdot X_i \\ &A_i = \text{Self-Attention}(Q_i, K_i, V_i) \\ &Z_i = A_i \cdot W_i^B \\ &Y_i = \text{MLP}(Z_i) \end{aligned} \tag{2}$$

As indicated in §II-B, existing model parallelism approaches are unable to fully separate the general matrix multiplications

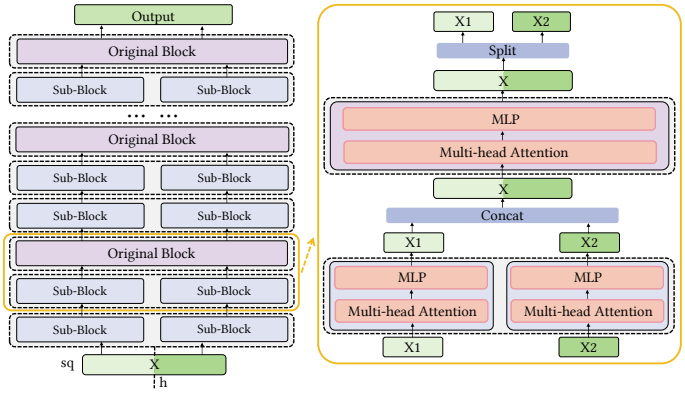


Fig. 5: Structure of decoupled Transformer model.

(GEMM) involved in linear operations into entirely independent computations. Inspired by the splitting strategy of tensor parallelism, we achieve independence by dividing both weights and inputs into fragments and recombining them into sub-blocks. In other words, BP approach eliminates the dependency among various block matrix multiplications in the original layer. In details, as shown in Fig. 4(b), the weight matrix of the first linear operation is divided by rows, while that of the second linear operation is divided by columns. This division is matched with a corresponding splitting of the input, resulting in each input fragment having a size of $[bs, sq, h/N_{div}]$. Then, each input fragment will only flow through the corresponding group of operations (Sub-Block). Notably, the order that splits by rows first can preserve the size of intermediate results and help maintain the accuracy. A similar strategy is also applied to the MLP module. N_{div} sub-blocks make up the decoupled layer, and the i-th sub-block can be formulated as Equation (2) ($i \in \{0, 1, ..., N_{div} - 1\}$).

Next, we build the Transformer model by stacking both the original layer and the decoupled layer, as shown in Fig. 5. Although the decoupled layer completely eliminates the data dependency between sub-blocks, it is unacceptable to only stack decoupled layers. This is because the lack of information exchange between intermediate results among sub-blocks can lead to a significant reduction in accuracy [7]. Thus, we use an original layer after every several decoupled layers to achieve information exchange between sub-blocks and ensure the model quality. The input and output of each original block are tensors of $[bs, sq, h]$ and that of each sub-block are tensors of $[bs, sq, h/N_{div}]$. Thus, before the original layer, we concatenate the tensors of N_{div} divisions along h dimension to form the tensor of size $[bs, sq, h]$, which are then passed into the original layer. After the original layer, we perform a split operation, splitting the tensor into N_{div} divisions in reverse for the subsequent decoupled layer. Notably, our decoupled model does not increase the model size that the sum of total parameters remains consistent regardless of N_{div}.

When distributing the decoupled model on devices, a trade-off exists between model accuracy and inference efficiency as each original layer naturally becomes a synchronization point across devices. Specifically, a larger N_b results in higher com-

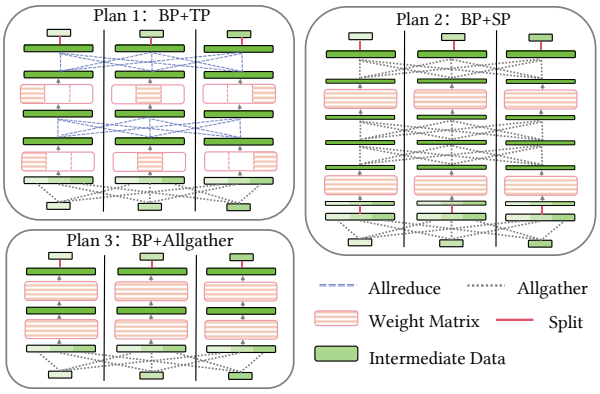


Fig. 6: Placement plan search space. There displays three heuristic placement plans of the original layer.

munication costs, which will offset the latency reduction gained from structure decoupling. Apart from inference efficiency, N_b also influences model accuracy. Having more original layers leads to more frequent feature interchanges, which leads to better accuracy, while having fewer original layers tends to hurt the model accuracy due to insufficient information interchange. We handle this balance by experiments as in IV-B.

C. Adaptive Placement Approach

The adaptive placement approach explores the optimal placement plan for the decoupled Transformer model with the primary goal of reducing latency. This approach decides the placement plan by comprehensively balancing the communication capability, computing power and memory budget in a multi-device system. It incorporates an offline profiling procedure to produce execution durations and an expertly crafted search space. With execution durations and the predefined search space, the adaptive placement approach takes a linear summation method to select the optimal placement plan from the search space.

In terms of the offline profiling procedure, DeTransformer leverages the observation that Transformer model inference exhibits little variance in the duration ratio between communication and computation when processing inputs with diverse sequence lengths. This observation enables us to conduct profiling using a typical sequence length and determine a general placement plan. Additionally, the presence of repeated layers in decoupled Transformer models significantly mitigates the profiling overhead. Consequently, we get operation durations on the target platform by averaging repeated executions.

To determine the search space, we focus on the placement plan of two distinct component types within the decoupled Transformer model respectively, namely decoupled layers and original layers. For decoupled layers, although we can also apply TP and SP for distributing them, they are less efficient than BP. Since a original block is decoupled into N_{div} divisions and they are independent, we can place these N_{div} sub-blocks of the decoupled layer evenly on the involving N_{dev} devices to exploit the inter-block parallelism revealed by BP, introducing no communication.

For original layers, it can be treated as a synchronization point. To pursue the optimal latency, it requires to strike a trade-off among the communication capability, computing power and memory budget, so as to determine the final placement plan. Based on these considerations, we propose three heuristic placement plans for the original layer as shown in Fig. 6, which is also the search space of our adaptive approach:

- **Plan 1 (BP+TP)** applies BP to decoupled layers and TP to original layers. The weights are partitioned equally in the original layer. In this way, there is an *allgather* communication before each original layer to collect outputs from all devices. Besides, each original layer requires another two *allreduce* communications.
- **Plan 2 (BP+SP)** applies SP to original layers and each device keeps all weights. There needs two *allgather* communications before and after each original layer. SP partitions the input sequence along sq dimension and takes another two *allgather* communications in each original layer.
- **Plan 3 (BP+Allgather)** asks that each device execute an original layer individually. This plan necessitates only an *allgather* communication. By contrast, each device has to keep all weights.

These three placement plans have their own advantages and drawbacks. Plan 1 and Plan 2 require frequent collective communications at a synchronization point, but each device bears only part of computation. Although Plan 2 has one more communication, the *allgather* communication typically has a shorter duration compared to the *allreduce* communication. Besides, Plan 1 causes minimal memory pressure on the device as each device only needs to load part of the weights, while Plan 2 requires each device to store all weights. Plan 3 requires the least communication but also the highest computation, and it also poses a challenge to the memory budget as a single device need to load the all weights. In a specific multi-device system, different plans will show diverse affinity towards network environment, computing power and memory budget.

With profiling results and search space, the adaptive placement approach takes a linear summation method to determine the final choice. At first, the adaptive approach will exclude the plan that will run out of the device memory. Next, we compare the request latency of different plans by accumulating durations of all operations required in the decoupled model. Thus, DeTransformer can generate the optimal placement plan with the shortest execution time from the search space for the distributed inference, which is then executed by DeTransformer's runtime.

IV. Evaluation

In this section, we evaluate the effectiveness of our DeTransformer system in two steps. As DeTransformer adopts decoupled Transformer structure, we firstly study the impacts of structure decoupling on model accuracy across a range of downstream tasks, which we call **Accuracy Experiments**. Second, we study the distributed inference performance of DeTransformer under a variety of network bandwidth, namely **Performance Experiments**.

TABLE III: Impacts of structure decoupling on model accuracy. Bold indicates that the decoupled model has equal or higher accuracy than the original one, while * indicates the best result.

Model	N_b	N_{div}	GLUE Mcc/Acc(%)				SQUAD Acc(%)
			CoLA	SST-2	MRPC	MNLI	
Original Bert-Base	\	\	40.43	91.28	84.56	81.59	77.54
Decoupled Bert-Base (Ours)	1	4	39.85	89.45	80.64	78.80	74.77
	2	4	**41.26**	89.68	**87.75***	80.54	75.82
	3	4	**41.20**	90.37	83.82	80.77	76.66
	4	4	**42.06**	**92.20***	**86.52**	81.21	76.91
	4	8	36.75	**91.28**	83.58	80.40	74.82
	4	2	**47.51***	89.91	**84.56**	**81.90***	**78.37***
Original Bert-Large	\	\	51.00*	91.39*	80.39	81.73	79.29
Decoupled Bert-Large (Ours)	4	4	47.20	90.82	**83.82**	**81.96***	78.85
	6	4	47.40	91.28	**85.04**	**81.81**	**79.86**
	8	4	44.25	90.37	**86.01***	**81.85**	**79.88***

A. Experiment Setup

Models. We evaluate the efficiency of our proposed DeTransformer on the trending Transformer-based model: BERT [2]. We focus on BERT-Base ($l = 12, h = 768, N_h = 12$) and BERT-Large ($l = 24, h = 1024, N_h = 16$) with 110M and 340M parameters respectively.

Accuracy Experiments. We pre-train BERT models on a high-performance cluster, equipped with 4 NVIDIA A800 GPUs, each having 80 GB of GPU device memory. We train both the original Bert and the decoupled Bert with identical hyper-parameters in Pytorch. In details, we train 7038 steps using a sequence length of 128 on English Wikipedia data corpus with 2.5B words, as used in the original Bert [2]. Notably, each training case takes more than 1 week and Table III shows decoupled models with varied N_b and N_{div}. Furthermore, we employ 4 tasks of GLUE benchmark [8] and SQuAD v1.1 task [9] as downstream tasks, which are commonly-used for evaluating a Transformer model, to demonstrate the performance of BERT models.

Performance Experiments. We conduct distributed inference on a multi-ARM platform with four Raspberry Pi 4B as edge devices, each of which consists of a ARM-Cortex-A72 Soc operating at 1.5GHz and 4 GB memory. Note that these edge devices are connected via a gigabyte H3C S1850V2 switch, providing an interface to adjust the D2D bandwidth so that we can evaluate the performance under different network conditions. We set the batch size to 1 and fix the sequence length to a constant value of 128. We employ the average end-to-end inference latency and throughput as our performance metrics. To ensure validity and reliability, we perform warm-up runs and record the average results of executing 100 samples, and leave devices idle for cooling down before the next run. DeTransformer is implemented atop of Pytorch with its distributed communication package.

For baselines in performance experiments, we compare DeTransformer with both single-device and two SOTA model parallelism approaches. All cases are summarized as follows:

- **Single Device**: The model inference is executed on a single device. We compare with it to analyze the scalability of DeTransformer. Notably, the inference latency on a single device of the original model and the decoupled model is very similar through experiments, therefore we treat them as the same baseline.
- **TP**: The tensor parallelism approach [5] that splits the weight matrices, requiring two times of *allreduce* communication in each Transformer layer.
- **SP**: The sequence parallelism approach [6] that splits the input sequence, requiring two times of *allgather* communication in each Transformer layer.
- **BP+Auto**: This uses both block parallelism approach and adaptive placement approach of DeTransformer for distributed inference.
- **BP+TP**: Only block parallelism approach is employed, and these original layers are parallelized using tensor parallelism approach.

B. Model Accuracy Analysis

Table III shows the accuracy comparison between the decoupled models and the original models, illustrating the impact of structure decoupling on model accuracy. Overall, these decoupled models can achieve comparable accuracy results to their original counterparts across most downstream tasks. Additionally, it is noteworthy that the decoupling process does not result in an enlargement of the model size. Consequently, although there are some cases with decreased performance, the structure decoupling in BP approach can achieve comparable accuracy results without increasing model sizes.

Moving onto decoupled models with different number of original layers, it is validated that decoupled models with more original layers generally have better accuracy. Taking decoupled Bert-Base with $N_{div} = 4$ and $N_b = 1, 2, 3, 4$ as the example, as the number of original layers increases, the accuracy results demonstrate an upward trend with slight fluctuations in most tasks. Regarding decoupled models with varying number of divisions, it is observed that a higher degree of division generally results in lower accuracy. For decoupled Bert-Base with $N_b = 4$ and $N_{div} = 2, 4, 8$, increasing N_b damages the accuracy in most downstream tasks. The same phenomenon can also be seen on Bert-large. In this way, we need to balance these configurations based on specific requirements.

C. Inference Efficiency Analysis

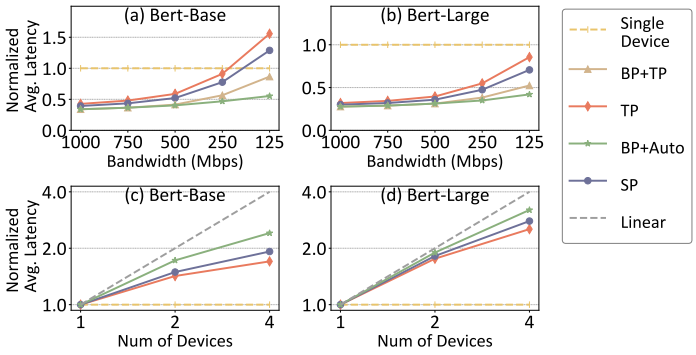


Fig. 7: Latency and throughput normalized to the single-device baseline. Results of (c) and (d) are under 500Mbps. Linear represents the ideal scaling.

TABLE IV: DeTransformer's Speedup over TP and SP.

Bandwidth (Mbps)	Bert-Base		Bert-Large	
	TP	SP	TP	SP
1000	1.25×	1.55×	1.14×	1.08×
750	1.31×	1.19×	1.19×	1.10×
500	1.46×	1.29×	1.26×	1.14×
250	1.94×	1.65×	1.57×	1.35×
125	2.81×	2.32×	2.03×	1.68×

1) Overall performance under various network conditions: We conduct experiments on Bert-Base and Bert-Large using 4 devices under varied bandwidths, including 1000, 750, 500, 250 and 125Mbps. DeTransformer is configured with $N_b = 4$ and $N_{div} = 4$. Fig. 7 (a)(b) illustrates the normalized latency results and Table IV shows corresponding speedup over TP and SP. All experiments show that DeTransformer consistently outperforms TP and SP with lower latency. Both TP and SP show a negative optimization trend in network condition with a bandwidth worse than 250Mbps, while DeTransformer remains positively optimized even at a bandwidth of 125Mbps. For example, distributing Bert-Base ($l = 12$) requires a total of 24 *allreduce* communications using TP approach. In comparison, our decoupled models have 4 original layers and 8 decoupled layers, and BP+TP approach only requires a total of 4 *all-gather* and 8 *allreduce* communications. Thus, DeTransformer can significantly reduce the communication overhead during distributed inference.

Furthermore, comparing BP+TP with BP+Auto, it shows that our adaptive placement approach can generate a better placement plan, outperforming the fixed strategy under lower network bandwidths. This advantage arises from the approach's ability to prioritize the placement plan with reduced communication frequency in such low network bandwidth conditions.

2) Scalability analysis: We select Bert-Base and Bert-Large, and decouple them with $N_{div} = 4$ and $N_b = 4$ using DeTransformer for comparing the strong scaling ability. We choose 500Mbps bandwidth for it is a common setting in edge scenarios. As illustrated in Fig. 7 (c)(d), BP+Auto of DeTransformer demonstrates superior strong scaling ability to TP and SP. Specifically, for Bert-Large, BP achieves 3.27× inference latency reduction compared to the single-device inference, while TP only achieves 2.58× inference latency reduction under the same condition. Moreover, we observe that Bert-Large shows better scaling results. This is because larger models have a greater ratio of computation.

3) Impacts of the number of original layers: We further compare DeTransformer's performance under different N_b configurations of decoupled Bert-Base on 4 devices. As in Fig. 8, we observe that reducing the number of original layers leads to improved performance. However, this optimization comes at the expense of model quality, as indicated in Table III, posing a trade-off between model accuracy and inference efficiency, meeting our expectations for the analysis in III-C.

V. Conclustion

Due to tightly-coupled structure, existing model parallelism approaches are difficult to expose a high degree of model parallelism in the distributed Transformer inference. In this paper, we propose DeTransformer, a communication-efficient distributed in-suit Transformer inference system for edge scenarios. DeTransformer includes two optimization approaches, namely the block parallelism approach and the adaptive placement approach. Block parallelism restructures the original Transformer layer with a single block to the decoupled layer with multiple sub-blocks and exploits model parallelism between sub-blocks. With loosely-coupled Transformer models, the adaptive placement approach finds the optimal placement plan with the latency as the primary goal, under constraints of communication capability, computing power and memory budget. Experimental results demonstrate a speedup of up to 2.81× compared to the state-of-the-art model parallelism approach when distributing across 4 devices. Meanwhile, DeTransformer achieves promising accuracy results on downstream tasks, equivalent to or even surpassing the original Transformer model, without enlarging the model size.

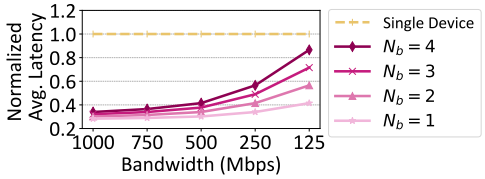


Fig. 8: Latency normalized to the single-device baseline of different N_b configurations on Bert-Base using 4 devices.

Acknowledgment

This research was supported by the National Key R&D Program of China 2021YFB0301300, and was also supported by the Major Program of Guangdong Basic and Applied Research: 2019B030302002 and Guangdong Province Special Support Program for Cultivating High-Level Talents: 2021TQ06X160 and Pazhou Lab Research Project: PZL2023KF0001.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," Advances in neural information processing systems, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[3] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," Advances in Neural Information Processing Systems, vol. 27, 2014.

[4] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen et al., "Gpipe: Efficient training of giant neural networks using pipeline parallelism," Advances in neural information processing systems, vol. 32, 2019.

[5] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," arXiv preprint arXiv:1909.08053, 2019.

[6] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence parallelism: Long sequence training from system perspective," in Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Jul. 2023.

[7] J. Du, X. Zhu, M. Shen, Y. Du et al., "Model parallelism optimization for distributed inference via decoupled cnn structure," IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 7, pp. 1665–1676, 2020.

[8] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," arXiv preprint arXiv:1804.07461, 2018.

[9] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," arXiv preprint arXiv:1606.05250, 2016.