









Resource-Efficient Collaborative Edge Transformer Inference With Hybrid Model Parallelism

Shengyuan Ye , *Graduate Student Member, IEEE*, Bei Ouyang , Jiangsu Du , Liekang Zeng, Tianyi Qian , Wenzhong Ou, Xiaowen Chu , *Fellow, IEEE*, Deke Guo , *Senior Member, IEEE*, Yutong Lu , *Member, IEEE*, and Xu Chen 

Abstract—Transformer-based models have unlocked a plethora of powerful intelligent applications at the edge, such as voice assistant in smart home. Traditional deployment approaches offload the inference workloads to the remote cloud server, which would induce substantial pressure on the backbone network as well as raise users' privacy concerns. To address that, in-situ inference has been recently recognized for edge intelligence, but it still confronts significant challenges stemming from the conflict between intensive workloads and limited on-device computing resources. In this paper, we leverage our observation that many edge environments usually comprise a rich set of accompanying trusted edge devices with idle resources and propose *Galaxy+*, a collaborative edge AI system that breaks the resource walls across heterogeneous edge devices for efficient Transformer inference acceleration. *Galaxy+* introduces a novel hybrid model parallelism to orchestrate collaborative inference, along with a heterogeneity and memory-aware parallelism planning for fully exploiting the resource potential. To mitigate the impact of tensor synchronizations on inference latency under bandwidth-constrained edge environments, *Galaxy+* devises a tile-based fine-grained overlapping of communication and computation. Furthermore, a fault-tolerant re-scheduling mechanism is developed to address device-level resource dynamics, ensuring stable and low-latency inference. Extensive evaluation based on prototype implementation demonstrates that *Galaxy+* remarkably outperforms state-of-the-art approaches under various edge environment setups, achieving a $1.2\times$ to $4.24\times$ end-to-end latency reduction. Besides, *Galaxy+* can adapt to device-level resource dynamics, swiftly rescheduling and restoring inference in the presence of unexpected straggler devices.

Index Terms—Edge intelligence, transformer-based model, distributed inference, model parallelism.

I. INTRODUCTION

TRANSFORMER-BASED models [1], [2] have achieved superior performance in the field of Natural Language Processing (NLP) and driven increasing intelligent applications at the network edge [3]. In edge intelligent applications, such as AI assistants in smart homes [4], voice-controlled robots in smart factories [5] and intelligent traffic management systems in smart cities [6], single-shot inference (referring to single-command requests) tasks are prevalent, necessitating efficient and low-latency inference for seamless user interactions. Currently, most Transformer-based intelligent applications heavily depend on cloud services, with the actual inference of large-scale Transformer-based models taking place in the cloud [7], [8]. At the edge, only a proxy daemon is deployed to forward user requests [4]. However, the cloud-assisted approaches suffer from following issues: (1) Quality-of-Service may suffer due to unreliable and delay-prone wide-area network (WAN) connections between edge devices and remote clouds [9]. (2) Inference requests from numerous edge clients can impose significant pressure on both the backbone network and datacenters. (3) The sensory data in edge intelligent applications can contain highly sensitive or private information. Transferring these data to the remote cloud owned by commercial companies inevitably raises users' privacy concerns [10]. A recent survey on LLM-based edge applications [11] indicates that more than 80% of industry experts advocate for personal LLMs to be fully or primarily deployed at the edge, emphasizing the critical need for privacy-preserving model inference.

To address that, in-situ inference [12], [13] on edge devices without remote assistance, which keeps data locally and avoids network transmission, has been recognized as a promising paradigm for intelligent applications at the edge. However, the computation-intensive and resource-hungry nature of Transformer inference presents significant challenges for resource-constrained edge devices [14], [15]. As we will show in Section II-B, inference on the BERT-L model [1] using an off-the-shelf edge device, the Raspberry Pi 4 Model B [16], requires a minimum available memory space of nearly 1.25 GB, while experiencing $470\times$ longer latency compared to one of today's most powerful datacenter GPUs, the NVIDIA A100 [17]. These

Received 25 October 2024; revised 20 March 2025; accepted 21 May 2025. Date of publication 29 May 2025; date of current version 3 September 2025. This work was supported in part by Guangdong S&T Programme under Grant 2024B0101040007, in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2023B1515120058, in part by Guangzhou Basic and Applied Basic Research Program under Grant 2024A04J6367, in part by Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355, and in part by Guangzhou Municipal Joint Funding Project with Universities and Enterprises under Grant 2024A03J0616. Recommended for acceptance by X. Yuan. (*Corresponding author: Xu Chen.*)

Shengyuan Ye, Bei Ouyang, Jiangsu Du, Tianyi Qian, Wenzhong Ou, Yutong Lu, and Xu Chen are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: yeshy8@mail2.sysu.edu.cn; ouyb9@mail2.sysu.edu.cn; dujiangsu@mail.sysu.edu.cn; qianty@mail2.sysu.edu.cn; ouwzh3@mail2.sysu.edu.cn; luyutong@mail.sysu.edu.cn; chenxu35@mail.sysu.edu.cn).

Liekang Zeng is with the Department of Information Engineering, The Chinese University of Hong Kong, Ma Liu Shui, Hong Kong (e-mail: lkzeng@cuhk.edu.hk).

Xiaowen Chu is with the Hong Kong University of Science and Technology, Guangzhou 511458, China (e-mail: xwchu@ust.hk).

Deke Guo is with the National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha 410073, China (e-mail: guodeke@gmail.com).

Digital Object Identifier 10.1109/TMC.2025.3574695

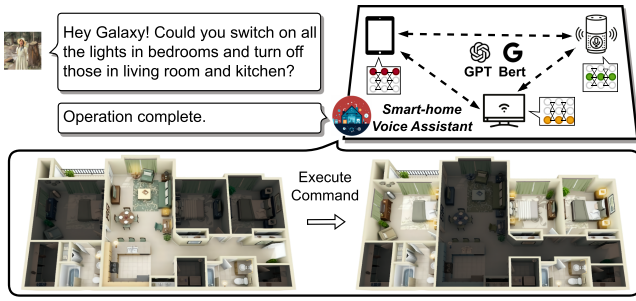


Fig. 1. AI assistant in smart home scenario empowered by Galaxy+.

results demonstrate the fundamental contradiction between intensive Transformer inference workload and constrained on-board resources. To tackle these challenges, existing arts explore to design sophisticated scheduling mechanisms to leverage the resource potential of edge devices [12], [18], [19], [20], but are still bottlenecked by the limited onboard resource of a single device.

Alternatively, we observe that prevalent edge environments like smart homes usually comprise a rich set of trusted idle devices in physical proximity [13], [21]. This motivates us to regard vicinal available edge devices as a resource augmentation and collaborate with them in a distributed manner to render expedited Transformer inference at the edge. As illustrated in Fig. 1, we can utilize the distributed computing resources in a smart home (with tablet, smart speaker, and television) to accelerate the Transformer-based (such as Bert [1] and GPT [22]) voice assistant. Nevertheless, this paradigm brings several key challenges: (1) how to parallelize the single-shot Transformer inference workload among multiple edge devices, (2) how to decide the workload partitioning strategy tailored to the resource budget of heterogeneous edge devices, (3) how to reduce distributed inference latency under bandwidth-limited edge environments, (4) how to handle the unpredictable dynamics of available edge resources, given that edge devices frequently host multiple edge intelligence applications concurrently.

To address these challenges, we propose Galaxy+, a collaborative edge AI system that breaks the resource walls across heterogeneous edge devices for low-latency Transformer inference to enable real-time in-situ edge intelligent services. Galaxy+'s contribution goes beyond merely leveraging distributed edge devices for deploying Transformer inference, instead it addresses the above challenges on four levels. (1) To orchestrate heterogeneous assisted devices in maximal resource utilization to facilitate collaborative inference, a novel hybrid model parallelism (HMP) that incorporates the best of both Tensor Parallelism (TP) and Sequence Parallelism (SP) is introduced as a novel parallel architecture to manage the distributed inference workflow. (2) To maximize resource utilization of HMP among edge devices, a parallelism planning algorithm that comprehensively accounts for both devices' resource heterogeneity and memory budget is equipped. (3) To achieve low-latency collaborative inference in bandwidth-limited edge environments, we meticulously decouple the tight data dependency between consecutive computation and communication operations by decomposing

them into fine-grained tiles, thus enabling efficient overlapping for synchronization. (4) The inherent resource dynamics of edge devices exacerbate collaborative inference latency due to the straggler effect. To mitigate this, we design and develop an on-the-fly fault-tolerant re-scheduling module for efficient runtime rescheduling and inference recovery. Extensive evaluations on practical testbeds show that Galaxy+ achieves up to $4.24\times$ speed-up over the state-of-the-art collaborative inference approaches. A 4-way parallel inference with Galaxy+ can achieve 85% scaling efficiency compared to the single device case. Our fault-tolerant module can adapt to device-level resource dynamics, swiftly rescheduling and restoring inference amid unexpected stragglers.

In summary, this paper makes the following contributions.

- Through extensive measurement studies on on-device and parallel inference methods, we introduce a novel HMP architecture to collaborate with trusted edge devices for in-situ single-shot Transformer inference acceleration.
- We devise a heterogeneity and memory-budget aware HMP planning algorithm to facilitate resource-efficient collaborative edge inference.
- We propose a tile-based fine-grained optimization that leverages the concept of communication and computation overlapping to mitigate the synchronization overhead.
- We design and develop an on-the-fly fault-tolerant re-scheduling module that efficiently adapts to device-level resource dynamics.
- We implement Galaxy+ and evaluate it in realistic edge testbeds. Experimental results show up to $4.24\times$ latency reduction over the state-of-the-art methods.

II. BACKGROUND AND MOTIVATION

A. Transformer-Based Models

Current language-related applications tend to use Transformer-based models, which consist of stacks of Transformer layers, enabling the scaling of language models to hundreds or even thousands of billions of parameters. The architectures of Transformer-based models can be broadly classified into three main categories: encoder-decoder, encoder-only, and decoder-only. The Transformer architecture was originally introduced as an encoder-decoder model [23], [24] for machine translation tasks. Encoder blocks extract high-level features from the input sentence, while the decoder blocks takes these features and generates tokens in the target language. In subsequent works, encoder-only architectures [1], [25] and decoder-only architectures [22], [26] emerged, each adopting only the encoder or decoder component from the original architecture. In this work, our design is applicable to edge collaborative parallel acceleration for general Transformer architectures, including encoder-only, decoder-only, and hybrid configurations combining both them.

Fig. 3 depicts a detailed Transformer layer architecture for encoder-only and decoder-only models. In a Transformer layer, the primary components are the Multi-head Attention (MHA) block and the Multilayer Perceptron (MLP) block. These components are connected through element-wise operations such as

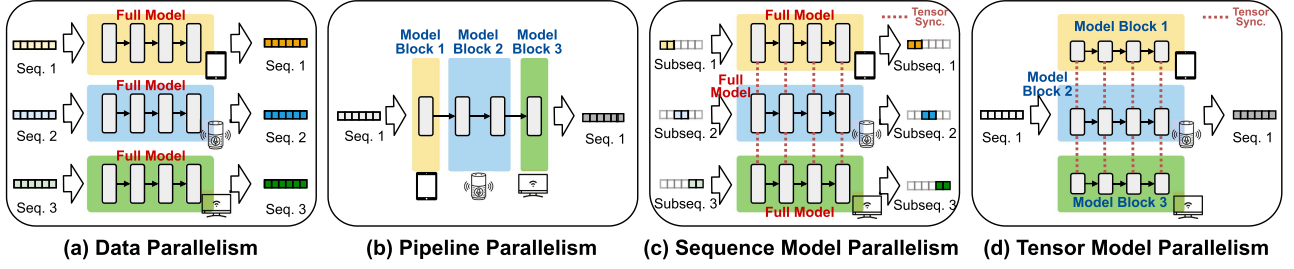


Fig. 2. Different parallelism plans of collaborative Transformer inference.

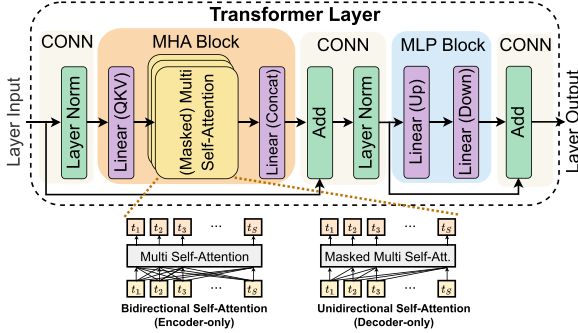


Fig. 3. A detailed Transformer layer architecture for the encoder-only and decoder-only model.

Dropout, Residual Addition, and Layer Norm. We refer to the parts connecting the MHA and MLP blocks as the connection (CONN) block. In MHA block, the first linear layer generates query (Q), key (K), and value (V) matrices for each attention head. Each head conducts self-attention independently, and their outputs are concatenated and further processed through a final linear layer to obtain the output. MLP block involves two linear operations that first expand the hidden size to a larger dimension and then compress it back to its original size. The core difference between encoder-only and decoder-only architectures lies in their attention mechanisms: encoder-only models use bidirectional self-attention, allowing each token to attend to all others, while decoder-only models apply a unidirectional mask to ensure that each input token can only attend to previous tokens.

The encoder-decoder architecture consists of a stack of N encoder layers and a stack of N decoder layers. The input is first processed by the encoder layers before passing through the decoder layers to generate the final output. The key difference between encoder-only and decoder-only models is that the encoder stack's output is fed into each decoder layer, requiring an additional multi-head attention mechanism for integration, as shown in Fig. 4. As all three architectures are fundamentally stacks of encoder- or decoder-based Transformer blocks, this paper uses encoder-only and decoder-only architectures as examples to illustrate the hybrid parallelism design (Section III-B1), with further discussion on extension to the encoder-decoder architecture in Section III-B3.

B. Transformer Inference on Resource-Limited Edge Devices

In-situ inference can leverage idle resources in edge environments while fully preserving users' data privacy, making it a widely utilized paradigm in privacy-sensitive edge

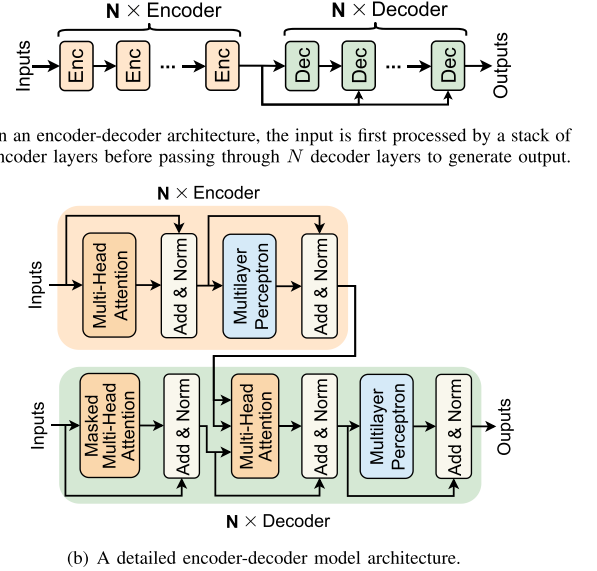


Fig. 4. Transformer-based models with the encoder-decoder architecture.

TABLE I
INFERENCE LATENCY AND MEM

Model	DistilBert	Bert-L	ViT-Huge	GPT2-L	OPT-L
Model Memory Footprint	0.25GB	1.25GB	2.46GB	2.89GB	4.92GB
Per-token Processing Latency(ms)	RPi 4B (M) [1]	0.11s	0.71s	OOM	OOM
	Nvidia A100 GPU [17]	0.3ms	1.5ms	2.2ms	2.5ms

Footprint of transformer models.

applications [13], [27]. However, the resource-intensive nature of Transformer inference presents significant challenges for resource-limited edge devices [28], [29]. We conduct experiments to analyze how limited computation resources affect on-device Transformer inference. The experimental setup is described in Section IV-A, and the results are presented in Table I. Specifically, we perform on-device inference for five typical Transformer-based models on off-the-shelf edge devices and the Nvidia server-grade GPU platform and report the average per-token inference latency and model memory footprint. We observe that the inference latency exhibits a huge gap between Nvidia A100 GPU [17] and Raspberry Pi 4 Model B [16], e.g., $360\times$ slowdown for RPi 4B(M) when comparing with Nvidia A100 GPU on DistilBert. Memory budget is another critical factor in Transformer inference. ViT-Huge, in full-precision floating-point format, incurs a 2.5 GB memory footprint during

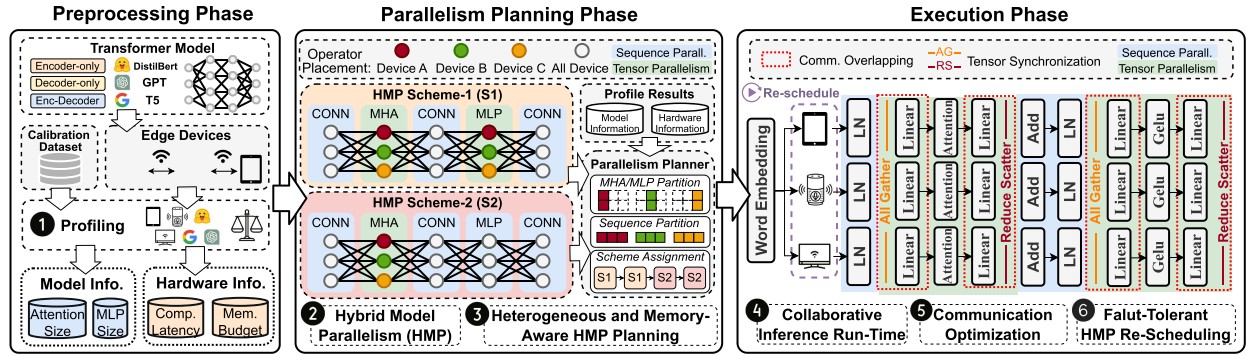


Fig. 5. Galaxy+ system overview.

inference, exceeding the predefined 1.5 GB memory budget of our RPi 4B(M) and resulting in out-of-memory (OOM) errors. Even with half-precision floating-point, the model still demands over 1.3 GB of memory, a footprint that remains unacceptable for many mobile edge devices.

To mitigate resource constraints, we leverage our observation that edge environments often consist of multiple trusted edge devices in physical proximity. This enables mutually-trustworthy computation resource sharing among these edge devices [13], [15].

C. Collaborative Transformer Inference With Multiple Devices

In collaborative Transformers inference across edge devices, the key question is the choice of parallelism strategy. We illustrate different parallelism plans in Fig. 2.

1) *Data and Pipeline Parallelism*: Data Parallelism (DP) and Pipeline Parallelism (PP) are the common way to execute Transformer-based model in parallel [30], [31], [32]. DP partitions workloads along the sample dimension, allowing each device to perform inferences independently. In edge intelligence services, where single-shot inference requests are frequently raised (e.g., sending a single piece of voice command to a smart assistant), DP is not applicable due to the absence of data batches. PP horizontally partitions the model into consecutive stages along layer dimension, with each stage mapped to a distinct device. However, in the case of single-shot inference, PP still falls short in leveraging multiple edge devices concurrently, as the inter-stage data dependencies force each device to await completion of the preceding one.

2) *Model Parallelism*: Model Parallelism (MP) is a parallel computing paradigm that horizontally partitions the operations within a model layer, facilitating concurrent execution of single-shot inference. The most common techniques of model parallelism applied to Transformer models are Tensor Model Parallelism (TP) [28], [33] and Sequence Model Parallelism (SP) [34]. TP partitions model weights across devices, each hosting a subset of parameters, yet it fails to parallelize some element-wise operations between MHA and MLP block. In contrast, SP segments the input along the sequence dimension, facilitating parallelism for all operations, but requires each device to store the entire model parameters. Due to intra-layer data dependencies, synchronization points are inserted during model parallelism to ensure consistency between collaborative

and local inference results. However, these synchronization points introduce significant communication latency, potentially becoming a bottleneck in inference performance, especially in bandwidth-limited edge environments.

Summarizing the above analysis motivates our design of a hybrid model parallelism architecture that incorporates the best of both TP and SP, with a communication optimization approach to mitigate synchronization overhead.

III. Galaxy+ DESIGN

A. Galaxy+ Workflow

Our system design aims to concurrently utilize multiple heterogeneous edge devices to achieve low-latency in-situ Transformer inference. Fig. 5 illustrates the workflow of our proposed Galaxy+, which features three primary phases: *Preprocessing Phase*, *Parallelism Planning Phase* and *Execution Phase*. *Preprocessing Phase* is an offline procedure that runs once before deployment. *Galaxy+ Profiler* performs an inference process using calibration data as input on the physical edge devices to record the run-time traces necessary for parallelism planning (step ①). In parallelism planning phase, *Galaxy+* adopts a novel hybrid model parallelism (HMP) architecture that incorporates both TP and SP to orchestrate distributed edge devices (step ②). *Galaxy+ Planner* takes profiling results from *Galaxy+ Profiler* as input to generate a parallelism planning configuration (step ③). This configuration comprehensively considers both resource heterogeneity and memory budget, and is subsequently applied to target Transformer-based models and edge devices in *Execution Phase* for efficient edge collaborative inference (step ④). Distributed inference inevitably involves tensor synchronization operations. *Galaxy+* incorporates a tile-based fine-grained communication optimization to mitigate the performance degradation brought by additional communication overhead (step ⑤). The dynamic resource fluctuations of edge devices amplify collaborative inference latency due to the straggler effect. *Galaxy+* further propose an on-the-fly fault-tolerant rescheduling module that enables efficient runtime rescheduling and inference recovery (step ⑥). With the above modules, *Galaxy+* focuses on the following design goals:

- A novel HMP architecture incorporates both TP and SP for low-latency single-shot Transformer inference across multiple edge devices (Section III-B).

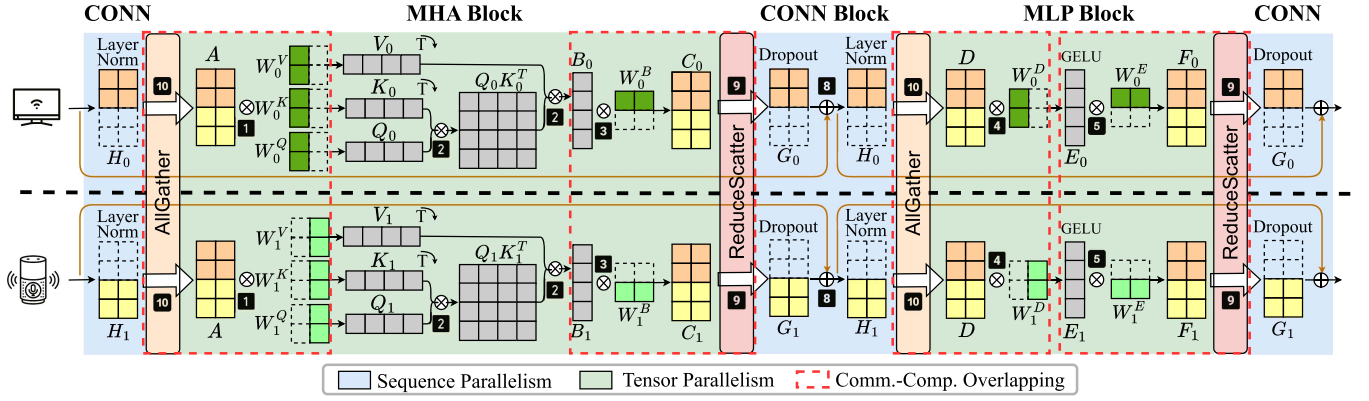


Fig. 6. Matrix representation of the first parallelism scheme (Scheme-1) in our hybrid model parallelism architecture, where TP is applied to both the MHA block and MLP block while SP is applied to CONN block.

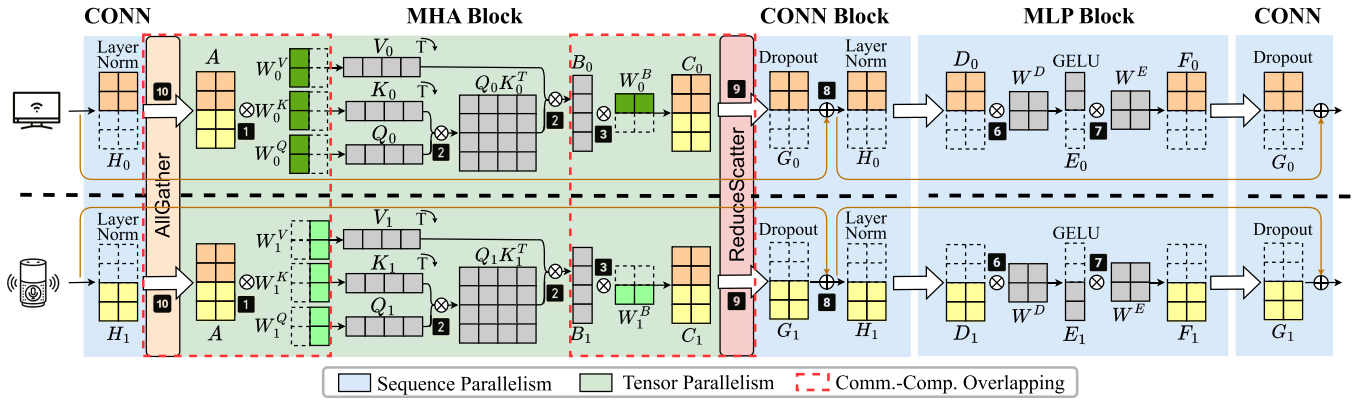


Fig. 7. Matrix representation of the second parallelism scheme (Scheme-2) in our hybrid model parallelism architecture, where TP is applied only to the MHA block, while SP is applied to both the MLP block and the CONN block.

- A judicious HMP planner that comprehensively considers the edge device heterogeneity and memory budget, aiming at distributing workload in a load-balanced manner and fully exploit computing and memory resources of edge devices (Section III-C).
- A tile-based fine-grained communication optimization decouples the tight dependency between consecutive computation and communication operations, enabling efficient overlapping between them (Section III-D).
- An on-the-fly fault-tolerant rescheduling module enables efficient runtime rescheduling and inference recovery (Section III-E).

B. Hybrid Model Parallelism

Galaxy+ incorporates an innovative hybrid model parallelism (HMP) architecture that incorporates the merits of TP and SP to facilitate efficient parallel Transformer inference within edge environments. This section elaborates on our HMP architecture using a collaborative inference example of encoder-decoder-only architectures deployed across two edge devices, with further discussion on extending it to the encoder-decoder model in Section III-B3. Figs. 6 and 7 illustrate two distinct HMP schemes that optimize Transformer-based model inference, with one focusing on memory scalability and the other on

communication efficiency. The proposed HMP architecture combines these two schemes to achieve resource-efficient Transformer inference. Specifically, Fig. 6 depicts the first parallelism scheme (Scheme-1), where TP is applied to both the MHA block and MLP block while SP is applied to CONN block. Fig. 7 depicts the second parallelism scheme (Scheme-2), where TP is applied only to the MHA block, while SP is applied to both the MLP block and the CONN block.

1) *Detailed Breakdown of Model Parallel Inference Process for Each Block:* In this subsection, we first provide a detailed breakdown of the TP or SP inference processes for each block in Transformer layers, as depicted in Figs. 6 and 7.

Tensor Parallelism on MHA Block. The aim of designing an efficient TP approach is to reduce the data dependencies among operators split across various devices, thereby reducing the frequency of tensor synchronization [33], [35]. As illustrated in Fig. 6, we exploit the inherent parallelism advantage of MHA: the computation of multiple attention heads is entirely independent. This head-level dependency allows us to split the operations of each attention head across edge devices without any tensor synchronization during the execution of Multi Self-Attention operations. With this in mind, we partition the weight matrices associated with key (W^K), query (W^Q), and value (W^V) along their head dimension. The initial General Matrix Multiply (GEMM) is distributed to distinct devices and

parallelized along head dimension (1). Subsequently, Self-Attention corresponding to each attention head is carried out locally on each respective device (2). The final GEMM from the output linear layer is parallelized along its row dimension, ensuring alignment with the initial GEMM's head-wise partition (3). The operations on device i ($i \in \{0, 1\}$) can be formulated as follows ($[\cdot|\cdot]$ is the concat operation):

$$\begin{aligned} [Q_i|K_i|V_i] &= [W_i^Q|W_i^K|W_i^V] \cdot A, \\ B_i &= \text{Self-Attention}(Q_i, K_i, V_i), \\ C_i &= W_i^B B_i. \end{aligned} \quad (1)$$

Tensor Parallelism on MLP Block: As illustrated in Fig. 6, the MLP block which comprises two consecutive GEMMs is applied with TP. To obviate tensor synchronization between the first and second GEMM operations, we leverage the concept of matrix tiling to remove data dependencies. We partition the weight matrix of the first GEMM along its column dimension (4), and partition the second GEMM along its row to align with the column-wise partition of the first GEMM (5). The second GEMM can directly take the output of the first GEMM as input without a tensor synchronization point. The operations on device i ($i \in \{0, 1\}$) can be formulated as follows:

$$\begin{aligned} E_i &= \text{GELU}(W_i^D D), \\ F_i &= W_i^E E_i. \end{aligned} \quad (2)$$

Sequence Parallelism on MLP Block: In addition to TP, our HMP architecture also employs SP to parallelize the MLP block, as illustrated in Fig. 7. We observe that the inference process of MLP blocks enables fully independent token computation within the input sequence, unlike the MHA module, which requires calculating inter-token relationships. This independence provides opportunities for parallel inference without introducing additional tensor synchronization overhead. Specifically, full model weights required for the MLP block are loaded into devices' memory. Prior to MLP block inference, the input sequence is partitioned along the sequence dimension into multiple sub-sequences, which are then processed independently on separate devices without tensor synchronization (6 and 7). The operations on device i ($i \in \{0, 1\}$) can be formulated as follows:

$$\begin{aligned} E_i &= \text{GELU}(W^D D_i), \\ F_i &= W^E E_i. \end{aligned} \quad (3)$$

Sequence Parallelism on Connective Block: The above parallelism designs accelerate the most computationally intensive parts of each Transformer layer (MHA and MLP blocks) while leaving the Dropout, Residual Addition and Layer Norm connecting the MHA block and the MLP block untouched (8). Although these operations are element-wise and entail no intensive matrix multiplication, they require a considerable amount of memory access, thus also yielding a non-negligible execution latency. We notice that these element-wise operations are independent along the sequence dimension which allows us to parallelize them by partitioning the input sequence. The

TABLE II
ANALYSIS ON COMPLEXITY OF VARIOUS PARALLELISM METHODS

Features	Model Memory Usage (per device)	Comp. Volume (per device)	Comm. Volume (per sequence)
HMP Scheme-1	$\mathcal{O}(\frac{P_1+P_2}{N})$	$\mathcal{O}(C/N)$	$4LSH$
HMP Scheme-2	$\mathcal{O}(\frac{P_1}{N} + P_2)$	$\mathcal{O}(C/N)$	$2LSH$

operations on device i ($i \in \{0, 1\}$) can be formulated as follows:

$$H_i = \text{LayerNorm}(\text{ResidualAdd}(\text{Dropout}(G_i))). \quad (4)$$

2) **Two Schemes of Hybrid Model Parallelism:** To achieve optimal inference latency under limited computation and communication resources in edge environments, we design two distinct HMP schemes.

Scheme-1. As illustrated in Fig. 6, the first scheme applies TP to both the MHA and MLP blocks, while SP is assigned to the CONN block. To ensure that the inference results from our parallel inference align with the local inference results, synchronization points are required at the end of these parallelism blocks. Towards the completion of TP on both MHA and MLP blocks, a *ReduceSum* operation is required to aggregate the computation results across multiple devices ($G \leftarrow C_0 + C_1$ and $G \leftarrow F_0 + F_1$). Subsequently, the aggregated results are partitioned along the sequence dimension and scattered across various edge devices for SP ($[G_0|G_1] \leftarrow G$). These two operations can be efficiently combined and implemented using a single *ReduceScatter* operation (9). Towards the completion of SP on CONN block, each device retains only a segment of the input sequences. It is essential to gather all these fragments, concatenate them, and distribute them across all devices for subsequent TP ($A \leftarrow [H_0|H_1]$ and $D \leftarrow [H_0|H_1]$). Consequently, we perform an *AllGather* communication primitive at the end of each CONN block (10).

Scheme-2. As illustrated in Fig. 7, the second scheme applies TP to only the MHA blocks, whereas SP is applied to both the MLP block and the CONN block. Similar to Scheme-1, Scheme-2 uses a *ReduceScatter* operation to connect the end of the MHA block to the beginning of the CONN block (9), and an *AllGather* operation to link the end of the CONN block back to the MHA block (10). In contrast to Scheme-1, Scheme-2 adopts SP for both the MLP and CONN blocks, eliminating the need for tensor synchronization at their boundaries and significantly reduces communication overhead. However, applying SP to the MLP block necessitates that each device holds the full block weights (W^D and W^E), which prevents leveraging the collective memory of multiple edge devices.

We analyze and summary the model memory usage, computation volume, and communication volume for above two HMP schemes in single sequence inference, as detailed in Table II. P_1 and P_2 represent the total number of model parameters in the MHA and MLP blocks, respectively. N refers to the number of edge devices, C denotes the total floating-point operations for a single sequence inference, L indicates the number of Transformer layers, S represents the input sequence length, and H is

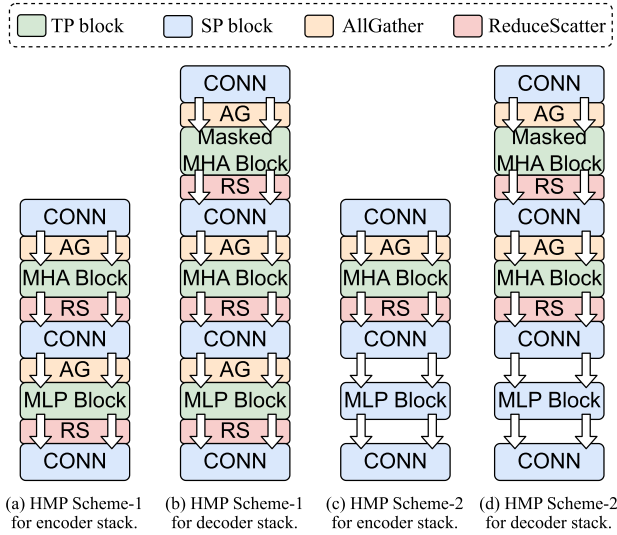


Fig. 8. Illustration of applying the HMP design to an encoder-decoder model. The colors of the parallel blocks are consistent with those in Figs. 6 and 7.

the hidden state size of the target Transformer-based models. We can observe that both Scheme-1 and Scheme-2 enable parallel acceleration of Transformer-based model inference. Scheme-1 more effectively utilizes collective memory, while Scheme-2 significantly reduces communication overhead by up to 50%.

3) *Extension to Encoder-Decoder Architecture:* In the previous sections, we introduced our hybrid model parallelism (HMP) design using the encoder- or decoder-only models illustrated in Fig. 3 as examples. In practical deployments, our HMP architecture is highly versatile and seamlessly extends to various Transformer architectures, including encoder-decoder models. Fig. 8 illustrates the application of the HMP architecture to an encoder-decoder Transformer model. As illustrated in Fig. 4(a), an encoder-decoder model processes the input through stacked encoder layers followed by stacked decoder layers. Thus, the HMP design can be applied separately to the encoder and decoder stacks. The application of HMP Scheme-1 and Scheme-2 to the encoder stack is consistent with that in encoder- and decoder-only models, as shown in Fig. 8(a) and (c). The decoder layer contains an additional masked multi-head attention block compared to the encoder layer. We parallelize the masked MHA block using the same TP as the MHA block and apply SP to the surrounding CONN blocks to achieve full parallelization of the decoder stack, as shown in Fig. 8(b) and (d).

4) *Merits of Hybrid Model Parallelism Architecture:* Employing the HMP architecture presents numerous advantages over straight TP [33] or SP [34] architecture.

Compared to TP: (1) Both HMP Scheme-1 and Scheme-2 architectures eliminate redundant computations in the CONN blocks through SP, further enhancing the parallel potential of Transformer layers. (2) HMP Scheme-1 does not introduce additional communication overhead. At first glance, state-of-the-art TP [35] requires two *AllReduce*, while the HMP Scheme-1 requires two *ReduceScatter* and two *AllGather* operations per Transformer layer inference. However, in the implementation of

TABLE III
TABLE OF NOTATIONS FOR PARALLELISM PLANNING

Notation	Definition
N	Total number of edge devices involves in inference.
\mathcal{A}	The partition of the MHA block along the head dimension across N devices.
\mathcal{B}	The partition of the MLP block along the row dimension of the weight matrix across N devices.
\mathcal{S}	The partition of the MLP and CONN block along the sequence dimension across N devices.
P	The number of Transformer layers assigned to Scheme-1.
Q	The number of Transformer layers assigned to Scheme-2.
μ_d	The memory budget allocated to device d .
\mathcal{V}_d	The computational capacity of device d .
$T(x, y_d, d)$	The execution latency of the block x on device d under partition configurations y_d .
$T_x(y)$	The collaborative execution latency of block x across N devices under partition configurations y .
$T_{\text{scheme-}i}$	Total inference latency for the Transformer layers processed using scheme- i in a target model.
$M_{\text{scheme-}i}^d$	Memory footprint of device d for the Transformer layers processed using scheme- i in a target model.
$T_{\text{AG}}/T_{\text{RS}}$	The latency for a single <i>AllGather</i> / <i>ReduceScatter</i> operation.

communication primitives, the communication volume of a single *Ring-AllReduce* operation equates to a *Ring-ReduceScatter* followed by a *Ring-AllGather* [36]. HMP Scheme-2 can significantly reduce communication overhead by up to 50% compared to TP, as it requires only one *AllGather* and one *ReduceScatter* operation. (3) HMP architecture splits the larger *AllReduce* operation used by TP into two smaller primitives, *ReduceScatter* and *AllGather*, which greatly facilitates our tiled-based communication overlapping proposed in Section III-D.

Compared to SP: SP partitions the input tensor along sequence dimension without partitioning the weight matrices. This paradigm requires each device to accommodate a holistic copy of the global model. Both HMP Scheme-1 and Scheme-2 mitigate this issue by distributing all or part of model parameters across edge devices, thereby breaking the memory wall of individual devices and achieving memory resource scalability.

C. Heterogeneity and Memory-Aware HMP Planning

To achieve resource-efficient edge collaborative HMP inference, a heterogeneity and memory-aware HMP planning algorithm is essential for determining model partitioning and optimizing the use of different schemes. As shown in Figs. 6 and 7, synchronization points are necessary after the completion of TP or SP blocks, with their initiation constrained by the completion time of the slowest device (straggler). This straggler effect can lead to resource under-utilization on faster devices. Given the inherent heterogeneity in both computational capabilities and memory budgets across edge devices, it is crucial to adopt a workload planning strategy that not only ensures balanced distribution but also prevents any device from experiencing OOM issue. Additionally, since Scheme-1 prioritizes memory efficiency and Scheme-2 focuses on reducing communication overhead, the planning algorithm must also determine the number of transformer layers assigned to Scheme-1 and Scheme-2, respectively, to achieve an optimal trade-off between memory usage and communication latency. Table III summarizes the notations used in our planning algorithm.

1) *Optimization Target Formulation:* As illustrated in Figs. 6 and 7, our HMP architecture allocates workload by partitioning along three dimensions: the head dimension for the MHA block, the row dimension of the weight matrix for the MLP block, and the sequence dimension of the input tensor for both the MLP and CONN blocks. Our workload planning focuses on determining the partition configuration for each of these blocks, namely: the MHA blocks partition $\mathcal{A} = \{a_0, a_1, \dots, a_{N-1}\}$, the MLP blocks partition $\mathcal{B} = \{b_0, b_1, \dots, b_{N-1}\}$, and the input sequence partition $\mathcal{S} = \{s_0, s_1, \dots, s_{N-1}\}$, where N is the number of edge devices. We introduce the notation $T(\text{MHA}, \mathcal{A}_d, d)$, $T(\text{MLP}, \mathcal{B}_d, d)$, $T(\text{MLP}, \mathcal{S}_d, d)$, and $T(\text{CON}, \mathcal{S}_d, d)$ to represent the execution latency of the MHA block, the MLP block, and the CONN block on device d , given their partition configurations \mathcal{A}_d , \mathcal{B}_d , and \mathcal{S}_d , respectively. It is important to note that, within the MLP block, \mathcal{B}_d partition configurations utilize TP, whereas \mathcal{S}_d configurations adopt SP. Since each TP or SP block is followed by a tensor synchronization point, its collaborative execution time for each TP or SP block determined by the straggler:

$$\begin{aligned} T_{\text{MHA}}(\mathcal{A}) &= \max_{d \in \{0, 1, \dots, N-1\}} T(\text{MHA}, \mathcal{A}_d, d), \\ T_{\text{MLP}}(\mathcal{B}) &= \max_{d \in \{0, 1, \dots, N-1\}} T(\text{MLP}, \mathcal{B}_d, d), \\ T_{\text{MLP}}(\mathcal{S}) &= \max_{d \in \{0, 1, \dots, N-1\}} T(\text{MLP}, \mathcal{S}_d, d), \\ T_{\text{CON}}(\mathcal{S}) &= \max_{d \in \{0, 1, \dots, N-1\}} T(\text{CON}, \mathcal{S}_d, d). \end{aligned} \quad (5)$$

In addition to determining the block partitioning results, another configuration that our HMP planning must determine is the allocation of each Transformer layer in the target model between Scheme-1 and Scheme-2. We denote the number of layers assigned to Scheme-1 as P , and those assigned to Scheme-2 as Q . We denote the time for a single *AllGather* operation as T_{AG} and for a *ReduceScatter* operation as T_{RS} . We formulate the inference latency for a Transformer-based model, with layers processed by Scheme-1 and Scheme-2, respectively, as follows:

$$T_{\text{scheme-1}} = P \times [T_{\text{MHA}}(\mathcal{A}) + T_{\text{MLP}}(\mathcal{B}) + T_{\text{CON}}(\mathcal{S}) + 2 \times (T_{\text{AG}} + T_{\text{RS}})], \quad (6)$$

$$T_{\text{scheme-2}} = Q \times [T_{\text{MHA}}(\mathcal{A}) + T_{\text{MLP}}(\mathcal{S}) + T_{\text{CON}}(\mathcal{S}) + (T_{\text{AG}} + T_{\text{RS}})]. \quad (7)$$

Beyond minimizing the execution latency, our planning algorithm also requires to prevent OOM errors during inference. The overwhelming memory footprint in deploying Transformer-based models stems from the substantial weight matrices housed within the MHA and MLP blocks. We denote M_{MHA} and M_{MLP} as the memory footprint of loading one MHA block and one MLP block, respectively. For Scheme-1, the memory footprint for MHA and MLP blocks on each device is directly proportional to the number of assigned attention heads and the number of rows in the weight matrix. For Scheme-2, the memory footprint for MHA is proportional to the number of assigned attention heads, while each device must store the full weight matrix for

the MLP block. We formulate the memory footprint of device d during collaborative inference of a Transformer-based model, with separate formulations for layers processed by Scheme-1 and Scheme-2, as follows:

$$\begin{aligned} M_{\text{scheme-1}}^d &= P \times \left(\frac{a_d}{\sum \mathcal{A}} M_{\text{MHA}} + \frac{b_d}{\sum \mathcal{B}} M_{\text{MLP}} \right), \\ M_{\text{scheme-2}}^d &= Q \times \left(\frac{a_d}{\sum \mathcal{A}} M_{\text{MHA}} + M_{\text{MLP}} \right). \end{aligned} \quad (8)$$

Let μ_d denotes the memory budget allocated to device d . Putting them together, the optimization objective for minimizing the latency under memory constraints is as follows:

$$\begin{aligned} \min_{\mathcal{A}, \mathcal{B}, \mathcal{S}, P, Q} & \left(T_{\text{scheme-1}} + T_{\text{scheme-2}} \right), \\ \text{s.t.} & \quad M_{\text{scheme-1}}^d + M_{\text{scheme-2}}^d < \mu_d, \\ & \quad \text{where } d \in \{0, 1, \dots, N-1\}. \end{aligned} \quad (9)$$

To facilitate our workload planning algorithm, we employ *Galaxy+ Profiler*, which conducts an inference process using calibration dataset as input on the physical edge devices to record the run-time profile necessary for parallelism planning. The profiler meticulously captures the computation and communication latency under a variety of partition configurations, for both TP and SP blocks. Simultaneously, *Galaxy+ Profiler* records model information, including the number of parameters and the memory footprint of the MHA and MLP blocks.

2) *HMP Planning Algorithm:* A straw-man approach to address the above constrained optimization problem (9) would involve an exhaustive search of all possible partitioning and scheme combinations, subsequently selecting the optimal solution that satisfies the memory constraints. However, this method suffers from an exponential complexity, rendering it infeasible for large-scale Transformer models.

To efficiently solve (9), we designed a two-step heuristic algorithm, outlined in Algorithm 1. In the first step, our algorithm disregards memory constraints and applies Scheme-1 to all Transformer layers, distributing the workload according to each device's computing capacity to achieve a balanced allocation (lines 28-30). This *proportional partition* (lines 1-6) allows for a fast and approximate division, enabling all devices to complete their tasks as simultaneously as possible. Since the partitioning strategy for input sequence length does not impact memory footprint, we consistently employ proportional partitioning to determine \mathcal{S} for input sequences of any length. We define a device's computing capacity \mathcal{V}_d as the inverse of the total time required to execute a MHA, MLP, and CONN block on device d .

$$\begin{aligned} \mathcal{V}_d &= \left[T(\text{MHA}, \sum \mathcal{A}, d) + T(\text{MLP}, \sum \mathcal{B}, d) \right. \\ & \quad \left. + T(\text{CON}, \sum \mathcal{S}, d) \right]^{-1}. \end{aligned} \quad (10)$$

Subsequently, building on this initial distribution, the second step involves first checking whether any device encounters an OOM issue. If any device encounters an OOM issue, we fine-tune the workload allocation accordingly. It redistributes excess

workloads from devices that surpass their memory budgets to those with spare memory capacity. (lines 31-35). Considering that the granularity of partitioning for MHA block (head dimension) is typically coarser than that of MLP block (column dimension), we first redistribute the workload for MLP block (line 32), followed by MHA block (line 33). If OOM errors persist despite workload redistribution, this indicates that the edge devices involved in collaborative inference are not capable of accommodating the target Transformer-based model, thus resulting in the algorithm's failure (lines 34-35). If no device encounters an OOM issue after evenly distributing the workload and additional memory remains available, we then iteratively analyze each Transformer layer currently using Scheme-1, progressively substituting them with Scheme-2 until an OOM exception occurs on any device. This refinement strategy enhances memory efficiency while minimizing communication overhead, thereby leveraging memory resources more effectively (lines 36-37).

The HMP planning is an offline procedure that runs once before deployment. The time complexity for Algorithm 1 exhibits an upper bound of $O(N + N^3)$. In our experiment, the planning time is under ten seconds on a domestic desktop for 4 heterogeneous edge devices.

D. Tile-Based Communication Optimization

In contrast to stable, high-bandwidth networks in datacenters, edge environments frequently grapple with inconsistent, bandwidth-limited connections. This amplifies synchronization latency during the collaborative inference, serving as a significant bottleneck of global system performance. Overlapping communication and computation is an effective optimization strategy. However, its implementation becomes intricate in the Transformer inference due to the strict data dependencies between communication and computation. To address this, Galaxy+ introduces a tile-based approach to effectively decouple their dependency to achieve a fine-grained overlapping. We observe from Fig. 6 that each TP block starts and ends with GEMM operations. We design to overlap these GEMM operations with the *AllGather* and *ReduceScatter* operations when entering and exiting the TP blocks. To illustrate this, the following section provides an example of collaborative inference across three devices, demonstrating how to overlap GEMMs with synchronization points before and after the MLP blocks (also applicable to the MHA blocks).

1) *AllGather Overlapping*: As illustrated in Fig. 6, a strict data dependency exists between the *AllGather* and the initial matrix multiply (GEMM1) in MLP block. Specifically, GEMM1 on device i ($i \in \{0, 1, 2\}$) can only commence after the *AllGather* has finished aggregating all sub-sequences:

$$D = \text{AllGather}(H_0, H_1, H_2), E_i = \text{GEMM1}(D, W_i^D). \quad (11)$$

To decouple the strict dependency between *AllGather* and GEMM1, we leverage matrix tiling to decompose GEMM1. We discover that the direct calculation of GEMM1 can be equivalently achieved by segmenting matrix D horizontally into tiles, executing the GEMM1 independently on each tile, and

Algorithm 1: Heterogeneity and Memory Aware HMP Planning.

Input: Profiling results of models and devices. \mathcal{V} : The list of computing capacity of devices. Seq : An input sequence.

Output: \mathcal{A} and \mathcal{B} : Partition configurations of MHA and MLP block. \mathcal{S} : Partition configurations of input sequences. P and Q : The number of layers assigned to Scheme-1 and Scheme-2.

```

1 Function BalancedPartitionScheme1( $T, \mathcal{V}$ ):
2   Initialize partition configuration  $C$ ;
3   Workload  $\leftarrow$  Total workload in  $T$ ;
4   foreach  $d \in \{0, 1, 2, \dots, N-1\}$  do
5      $C_d \leftarrow (\mathcal{V}_d / \sum \mathcal{V}) \cdot \text{Workload}$ ;
6   Return  $C$ ;
7 Function BalanceMemory( $T, C, \mathcal{V}, \mathcal{L}$ ):
8    $OOM\_Devices \leftarrow$  Out-of-memory devices under
    partition config.  $C$  in  $\mathcal{L}$ ;
9    $Free\_Devices \leftarrow$  Devices retaining available
    memory under partition config.  $C$  in  $\mathcal{L}$ ;
10  if  $OOM\_Devices = \emptyset$  then
11    Return  $C$ ;
12  foreach  $o \in OOM\_Devices$  do
13     $Waiting\_Shift \leftarrow$  Overflowing workload on
    device  $o$ ;
14    foreach  $f \in Free\_Devices$  do
15       $Shift(\mathcal{V}_f / \sum_{i \in Free\_Devices} \mathcal{V}_i) \cdot$ 
         $Waiting\_Shift$  workload from  $o$  to  $f$ ;
16    Remove device  $o$  from  $\mathcal{L}$ ;
17  BalanceMemory ( $T, C, \mathcal{V}, \mathcal{L}$ );
18 Function SubstituteScheme2():
19    $P \leftarrow$  Number of Transformer Layers,  $Q \leftarrow 0$ ;
20   foreach Transformer Layers do
21     Replace the current layer with Scheme-2;
22     if Out-of-memory devices exist then
23       Revert the current layer back to Scheme-1;
24       break;
25      $P \leftarrow P-1, Q \leftarrow Q+1$ ;
26   Return  $P, Q$ ;
27  $\mathcal{L} \leftarrow [0, 1, \dots, N-1]$ ;  $\triangleright$  List of all devices
28  $\mathcal{A} \leftarrow \text{BalancedPartitionScheme1}(MHA, \mathcal{V})$ ;
29  $\mathcal{B} \leftarrow \text{BalancedPartitionScheme1}(MLP, \mathcal{V})$ ;
30  $\mathcal{S} \leftarrow \text{BalancedPartitionScheme1}(Seq, \mathcal{V})$ ;
31 if Out-of-memory devices exist then
32    $\mathcal{B} \leftarrow \text{BalanceMemory}(MLP, \mathcal{B}, \mathcal{V}, \mathcal{L})$ ;
33    $\mathcal{A} \leftarrow \text{BalanceMemory}(MHA, \mathcal{A}, \mathcal{V}, \mathcal{L})$ ;
34   if Out-of-memory devices still exist then
35     Exit with Fail;
36 else
37    $P, Q \leftarrow \text{SubstituteScheme2}()$ ;

```

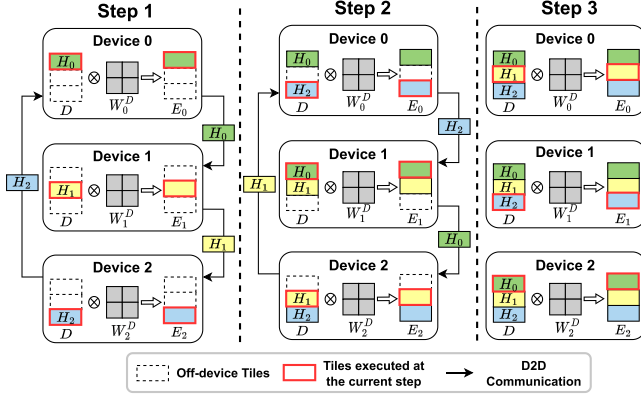


Fig. 9. Ring-AllGather overlapping.

subsequently concatenating the results.

$$E_i = \begin{bmatrix} H_0 \cdot W_i^D \\ H_1 \cdot W_i^D \\ H_2 \cdot W_i^D \end{bmatrix} = \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix} \cdot W_i^D = D \cdot W_i^D. \quad (12)$$

We employ a *Ring-AllGather* implementation and integrate it with the above matrix tiling approach to overlap communication and computation. An example of an overlapping process involving three collaborative devices is illustrated in Fig. 9. In the context of a tile-based overlapping process that incorporates N devices, typically N steps are required (three steps in this case). We define $(i+1)\%3$ and $(i-1)\%3$ represent the index of succeeding and preceding device of device i within a 3-device ring topology. *Step 1*: Device i performs GEMM operation between on-device tile H_i and W_i^D , and concurrently dispatches H_i to the succeeding device. In parallel, Device i receives and stores the tile $H_{(i-1)\%3}$ transmitted from its preceding device. *Step 2*: Device i performs GEMM operation on tile $H_{(i-1)\%3}$ and concurrently dispatches it to the succeeding device. In parallel, Device i receives the tile $H_{(i-2)\%3}$ transmitted from its preceding device. *Step 3*: Device i executes the GEMM operation on the tile $H_{(i-2)\%3}$. Notably, the final step does not necessitate any communication. The outcomes of the three GEMM operations are concatenated along the sequence dimension, yielding the final result E_i .

2) *ReduceScatter Overlapping*: As illustrated in Fig. 6, a strict data dependency exists between the final matrix multiplication (GEMM2) in the MLP block and the *ReduceScatter* operation ($i \in \{0, 1, 2\}$):

$$F_i = \text{GEMM2}(E_i, W_i^E), G_i = \text{ReduceScatter}(F_0, F_1, F_2). \quad (13)$$

To decouple the strict dependency between *ReduceScatter* and GEMM2, we mirroring the tiling approach used with the *AllGather*. We split the matrix E_i into three equally-sized tiles $E_{i,r}$ ($r \in \{0, 1, 2\}$) along the row dimension (aligns with the partition configuration of connective block) and compute GEMM2 independently for each tile (14). To obtain the final result G_r , an additional ReduceSum operation across all devices

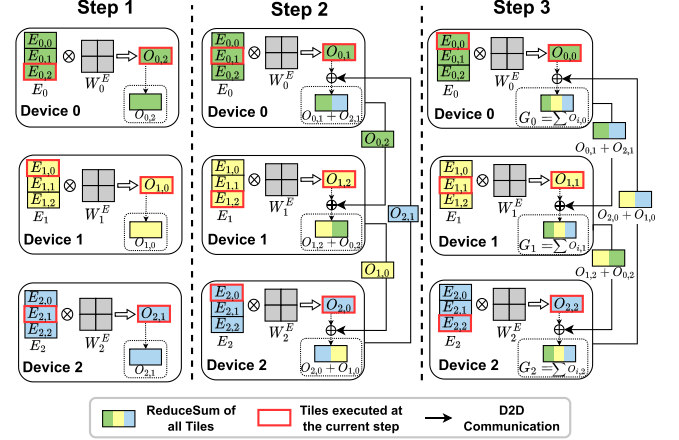


Fig. 10. Ring-ReduceScatter overlapping.

is necessary (15).

$$\begin{bmatrix} O_{i,0} \\ O_{i,1} \\ O_{i,2} \end{bmatrix} = \begin{bmatrix} E_{i,0} \cdot W_i^E \\ E_{i,1} \cdot W_i^E \\ E_{i,2} \cdot W_i^E \end{bmatrix} = \begin{bmatrix} E_{i,0} \\ E_{i,1} \\ E_{i,2} \end{bmatrix} \cdot W_i^E = E_i \cdot W_i^E, \quad (14)$$

$$G_r = \sum_i O_{i,r}. \quad (15)$$

Similar to *AllGather*, we employ a *Ring-ReduceScatter* implementation coupled with matrix tiling to achieve communication and computation overlapping. As illustrated in Fig. 10, the process of *ReduceScatter* overlapping also involves three steps. *Step 1*: Device i performs GEMM operation between tile $E_{i,(i+2)\%3}$ and W_i^E , yielding the result $O_{i,(i+2)\%3}$. *Step 2*: Device i perform GEMM operation on tile $E_{i,(i+1)\%3}$ and yield the result $O_{i,(i+1)\%3}$. In parallel, device i forwards the GEMM result in step 1 to the subsequent device. Upon receiving the tile from the preceding device, Device i conducts a ReduceSum operation between it and $O_{i,(i+1)\%3}$. *Step 3*: Device i perform GEMM operation on tile $E_{i,i}$ and yield the result $O_{i,i}$. Device i concurrently sends the result of ReduceSum in Step 2 to the subsequent device. A ReduceSum operation is performed between the tile received from the preceding device and $O_{i,i}$, yielding the final result G_i .

Our tile-based communication optimization seamlessly overlaps $N-1$ rounds of ring communication with N rounds of GEMM operation, without imposing additional overhead or yielding results inconsistent with non-overlapping approaches.

E. Fault-Tolerant HMP Re-Scheduling

Existing collaborative inference system usually statically partitions inference workload among the workers [37]. However, this static partitioning approach is not suitable for the edge environment because of two reasons: (1) In practical deployments, edge devices in environments such as smart homes typically host multiple edge intelligence applications alongside collaborative inference. The resource availability of an edge device depends on user habits and application demands. When an edge device engaged in collaborative inference runs additional applications,

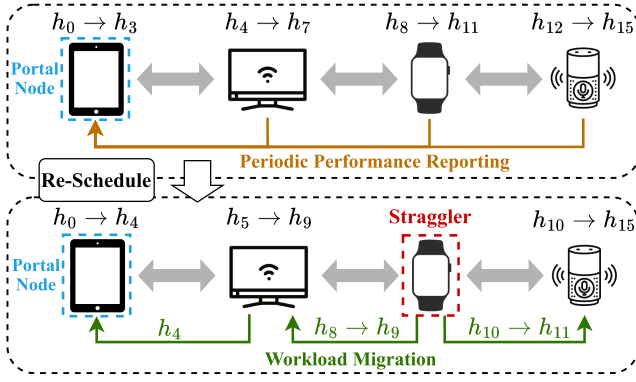


Fig. 11. Example of our fault-tolerant HMP re-scheduling. Here, h_i represents attention head i . When a straggler occurs, its assigned attention head computations are reallocated to other devices through re-scheduling.

the computational resources available for inference may be significantly diminished or experience considerable fluctuations. (2) As previously discussed, our HMP architecture incorporates synchronization points across parallel blocks, making inference latency bottlenecked by the slowest device. A significant drop or fluctuation in a device’s computational capacity can cause it to become a straggler, stalling other edge devices and severely degrading the system’s overall response performance. To address these challenges, we design and implement an on-the-fly fault-tolerant HMP re-scheduling module, which consists of two key steps to efficiently adapt to resource fluctuations:

1) *Periodic Performance Reporting*: As illustrated in Fig. 11(Top), before collaborative inference begins, we designate one device as the *portal node*. During collaborative inference, all involved devices periodically report their execution latency for each parallel inference task. The portal node collects and compares the execution times of the same inference task across different devices. If a device repeatedly becomes a straggler, meaning its inference latency is significantly higher than that of other devices, we consider it to be experiencing a significant drop or fluctuation in computational capacity. Consequently, the device will be marked as unsuitable for participating in collaborative inference, triggering *On-the-fly HMP Re-scheduling* step.

2) *On-the-fly HMP Re-scheduling*: When the portal node detects a device experiencing significant computational instability, it will be temporarily excluded from collaborative inference, and its workload is redistributed to other devices, as shown in Fig. 11(Down). Specifically, we rerun Algorithm 1 to derive a new balanced parallelism configuration, ensuring efficient workload migration among the remaining edge devices. Notably, this migration is purely logical, as inference tasks do not require updating model weights. Rather than transferring weights over the network, each device maintains a complete model weight copy on disk and selectively loads it into memory according to the updated configuration. After migration, the system continues serving inference requests according to the updated parallelism configuration. Similarly, once the straggler becomes idle again, our re-scheduling mechanism will dynamically involve it back into our collaborative inference system.

TABLE IV
SPECIFICATIONS OF RASPBERRY PI 4 MODEL B [16] AND THE HOMOGENEOUS AND HETEROGENEOUS EDGE ENVIRONMENTS USED IN THE EVALUATION

Hardware	Raspberry Pi 4 Model B	
CPU	64-Bit Quad Core ARM Cortex-A72 CPU	
CPU Frequency Mode	RPi-S	403MHz
	RPi-M	825MHz
	RPi-L	1.47GHz
Edge Environments	Homo. Env. A	4×RPi-M
	Hetero. Env. B	2×RPi-L + RPi-M + RPi-S

IV. IMPLEMENTATION AND EVALUATION

We have fully implemented the prototype system of Galaxy+ and baselines with ~ 2500 LoC in Python and C/C++ atop Pytorch [38]. Galaxy+’s idea is also portable and can work well with other lightweight ML frameworks such as llama.cpp [39], MNN [40] and TF-Lite [41]. In this section, we evaluate the performance of Galaxy+ prototype for five different sizes of Transformer-based models on physical testbeds.

A. Experimental Setup

Models and Datasets. We evaluated Galaxy+ using six representative Transformer-based models for language and vision tasks, ranging in size from 66 million to 1.3 billion parameters. These include popular open-source models: the encoder-only architectures DistilBERT [42], BERT [1], and Vision Transformer [25]; the encoder-decoder architectures T5 [24]; and the decoder-only architectures GPT [22] and OPT [26], as detailed in Table V. For language models, we extracted a subset of samples with an average sequence length of 284 from the QNLI corpus within the popular GLUE benchmark [43] for evaluation. For vision models, we used the Mini-ImageNet dataset [44] with an input size of $3 \times 224 \times 224$.

Edge Environment Setup. We evaluate Galaxy+ across a diverse range of realistic edge environments, incorporating both homogeneous and heterogeneous configurations of off-the-shelf edge devices (Raspberry Pi 4 Model B [16]), as detailed in Table IV. In homogeneous environments, the available memory budget for model inference on the RPi-M is set to 1.5 GB. In the heterogeneous environments, the memory budgets are set at 2 GB for RPi-L, 1.5 GB for RPi-M, and 1 GB for RPi-S, respectively. In our experiments, we primarily used the Raspberry Pi 4 Model B, equipped with an ARM Cortex-A72 CPU, to simulate resource-constrained edge devices commonly deployed in environments such as smart homes or smart factories. However, in Section IV-D, we also conducted a case study to demonstrate the applicability of our Galaxy+ framework for deployment on consumer-grade GPU clusters. We interconnected multiple edge devices via a switch with adjustable network speed and modified the D2D bandwidth to emulate varying network conditions representative of real-world edge environments.

Baseline Methods. We compare Galaxy+ with both single-device method and state-of-the-art parallel methods:

- *Local Inference (Local)*: Inference models on a single device. We compare with it to analyze the scalability performance of Galaxy+.

TABLE V
MODEL SPECIFICATIONS AND OVERALL PERFORMANCE COMPARISON OF THE Galaxy+ WITH BASELINE PARALLEL INFERENCE METHODS

Structure	Model	Layers	Heads	Hidden Size	Parameter Volume	Edge Env.	HMP Configs	Speedup Over Baseline			
								M-LM	SP	ES	Galaxy
Encoder-only	DistilBert [42]	6	12	768	66M	Env. A	[0,6]	1.34×	1.20×	2.31×	1.19×
						Env. B	[0,6]	3.91×	3.54×	4.01×	1.41×
	Bert-L [1]	24	16	1024	340M	Env. A	[0,24]	1.82×	1.34×	3.55×	1.20×
						Env. B	[0,24]	3.52×	OOM	3.55×	1.28×
	ViT-Huge [25]	32	16	1280	632M	Env. A	[10,22]	1.25×	OOM	2.21×	1.15×
						Env. B	[10,22]	2.91×	OOM	2.58×	1.20×
Encoder-Decoder	T5-L [24]	48	16	2048	740M	Env. A	[18,30]	1.64×	OOM	3.21×	1.16×
						Env. B	[21,27]	3.97×	OOM	3.75×	1.14×
Decoder-only	GPT2-L [22]	36	20	1280	774M	Env. A	[20,16]	1.42×	OOM	2.97×	1.12×
						Env. B	[15,21]	3.41×	OOM	3.59×	1.16×
	OPT-L [26]	24	16	2048	1.3B	Env. A	[24,0]	1.41×	OOM	3.6×	1.0×
						Env. B	[16,8]	OOM	OOM	4.24×	1.09×

OOM denotes an out-of-memory error. The performance metric is the average per-token processing latency.

- *Megatron-LM (M-LM)* [33] is a state-of-the-art TP method splits the weight matrix in MHA and MLP blocks to parallelize the GEMM operators. An AllReduce synchronization is required after each MHA and MLP block.
- *Sequence Parallelism (SP)* [34] is a state-of-the-art SP method partitions the input along its sequence dimension and parallelizes inference across workers. Two AllGather synchronizations are required among each MHA block.
- *EdgeShard (ES)* [45] is a pioneering collaborative edge LLMs inference system that employs pipelined architecture to orchestrate edge devices.
- *Galaxy* [37] is the shorter conference version of this work. It adopts a hybrid model parallel architecture but is limited to designing and utilizing only HMP Scheme-1.

B. Comparison to Baselines

Table V summarizes the general performance results comparing Galaxy+ with state-of-the-art parallel inference methods M-LM, SP, EdgeShard and Galaxy. We conduct experiments on both homogeneous and heterogeneous edge environments, Env. A and B, each with 500 Mbps intra-cluster bandwidth. We employ the average per-token processing latency as our performance metric. The results indicate that owing to our HMP architecture and tile-based communication optimization, Galaxy+ outperforms baselines across various models and edge environments.

In the homogeneous edge environment A, Galaxy+ achieves an inference speedup of 1.25×-1.82× compared to Megatron-LM (M-LM). This improvement is primarily due to Galaxy+'s HMP architecture, which significantly reduces communication overhead. While Megatron-LM's tensor parallelism requires an AllReduce tensor synchronization at every Transformer layer during inference, Galaxy+'s HMP Scheme-2 architecture minimizes communication by performing only one AllGather and one ReduceScatter operation, cutting the communication cost to half compared to two AllReduce operations. Moreover, our tile-based computation-communication overlapping optimization effectively reduces the impact of network latency on system

performance. When compare to Sequence Parallelism (SP), Galaxy+ achieves an inference speedup of 1.2×-1.34×. SP requires each device to maintain a full set of model weights, which limits memory scalability in edge environments as the number of devices increases. This constraint leads to out-of-memory (OOM) issues with larger models such as ViT-Huge, GPT2-L, and OPT-L. Galaxy+'s HMP architecture address this by partitioning the model weights across multiple devices, enabling support for larger models through collective memory. In single-input sequence tasks, the pipelined approach of EdgeShard degrades into sequential inference. In comparison, Galaxy+ reduces latency by up to 3.59× by concurrently utilizing the computational resources of multiple devices.

We further compared the performance of Galaxy+ with the baselines in the heterogeneous edge environment B. We observe that Galaxy+ consistently and significantly outperforms M-LM and SP parallel inference methods in heterogeneous edge environments, achieving an inference latency reduction of 2.91×-3.91×. This speedup is notably greater than the improvements observed in homogeneous environments. Galaxy+'s superior performance in heterogeneous edge environments derives from its consideration of device heterogeneity, a factor overlooked by M-LM and SP, both tailored for datacenters equipped with homogeneous accelerators. While EdgeShard accounts for the heterogeneity of edge resources, its performance is surpassed by Galaxy+, which achieves up to a 4.24× speedup by fully leveraging parallel resource utilization across multiple edge devices. In addition to device heterogeneity, Galaxy+ workload planning comprehensively considers the memory budget of edge devices, enabling them to collaboratively accommodate the target model. In contrast, M-LM and SP overlook the memory constraints during parallelism planning, resulting in OOM errors.

We also compare the shorter conference version of this work: Galaxy. Galaxy employs hybrid model parallelism that integrates the advantages of TP and SP but is limited to designing and utilizing only HMP Scheme-1. Galaxy+'s further research reveals that using only the HMP architecture with Scheme-1 overlooks the potential optimization of applying sequence

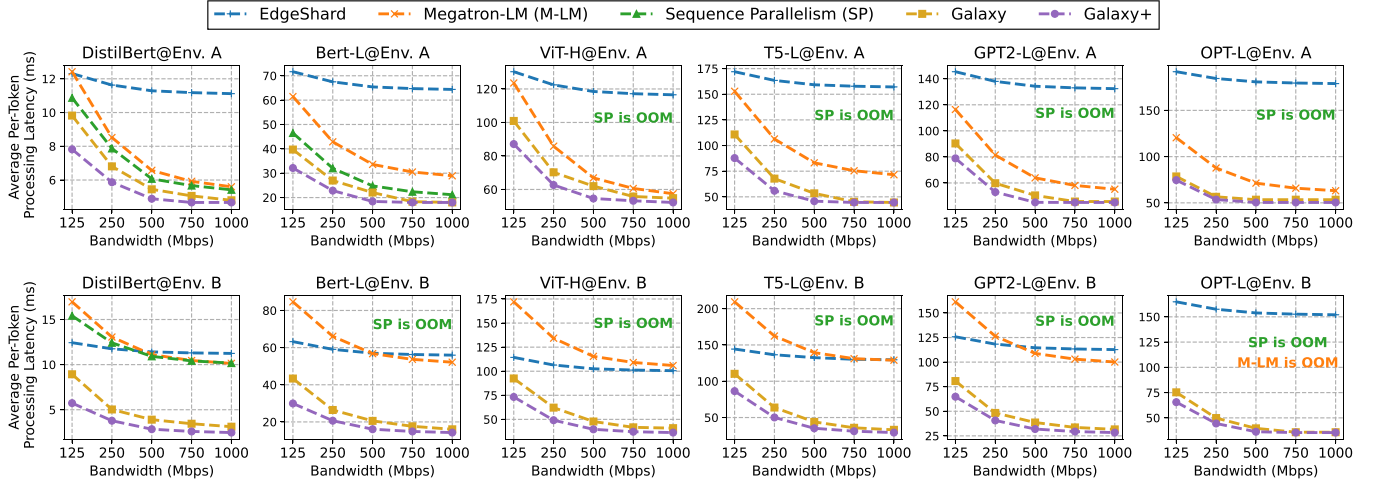


Fig. 12. Performance comparison of the Galaxy+ with baselines under various network bandwidth.

parallelism to the MLP block, which can reduce communication overhead by 50% per Transformer layer during inference. Table V reports the speedup achieved over Galaxy and the scheme configurations of Galaxy+’s HMP architecture, where $[a, b]$ indicates that a Transformer layers use Scheme-1 and b layers use Scheme-2, with $a + b$ equaling the total number of model layers. We observe that Galaxy+’s parallelism planning algorithm strategically prioritizes the use of HMP Scheme-2, which incurs lower communication overhead, while staying within the constraints of the memory budget. Specifically, for smaller models like DistilBert and Bert-L, Galaxy+ employs HMP Scheme-2 across all Transformer layers to reduce communication overhead. For larger models such as ViT-Huge, T5-L, GPT2-L, and OPT-L, Galaxy+ optimally utilizes HMP Scheme-2 while carefully managing memory constraints to prevent OOM issues. Experimental results show that Galaxy+ can achieve up to a $1.41\times$ speedup compared to Galaxy across various Transformer-based models.

C. Evaluate With Various Network Bandwidth

We further compare Galaxy+’s performance with baselines under varied network conditions in both Environment A and B. Using the switcher’s traffic control, we simulate five D2D bandwidths to mimic various network conditions at edge. Evaluation results are shown in Fig. 12. We observe that under varying network bandwidth conditions, Galaxy+ consistently outperforms all baseline methods in both homogeneous and heterogeneous edge environments. When compare to Megatron-LM, Sequence Parallelism and EdgeShard, Galaxy+ achieves an inference latency reduction of $1.2\times$ – $2.57\times$ in homogeneous environments and $1.56\times$ to $4.3\times$ in heterogeneous environments. In homogeneous environments with high network bandwidth (e.g., 1000 Mbps), M-LM and SP deliver inference performance comparable to Galaxy+. However, as bandwidth decreases, Galaxy+ exhibits greater resilience, owing to its HMP architecture and optimizations for overlapping communication and computation. At very low bandwidth (e.g., 125 Mbps), M-LM’s inference performance even falls below that of the sequential

TABLE VI
SPECIFICATIONS OF EDGE ENVIRONMENTS WITH MOBILE/CONSUMER-GRADE GPUs

Edge GPUs Environments	Env. C (Mobile GPUs)	4×Jetson Nano [46]
	Env. D (Consumer-Grade GPUs)	4×Nvidia 1080Ti [47]

inference method, EdgeShard, as the communication bottleneck negates the benefits of edge collaboration. This further demonstrates the deployment advantages of the Galaxy+ in low-bandwidth edge environments. When compared to Galaxy, Galaxy+ achieves up to a $1.56\times$ reduction in inference latency. This improvement is particularly crucial in low-bandwidth edge environments (e.g., 125 Mbps and 250 Mbps), where tensor synchronization becomes a system performance bottleneck. In extremely low-bandwidth scenarios, optimizing communication-computation overlap has limited effectiveness, and directly reducing the frequency of tensor synchronization is the most effective way to enhance inference performance.

D. GPU Support

Graphics Processing Units (GPUs), specialized for parallel computing, offer significantly higher performance than CPUs for tasks requiring concurrent processing. In recent years, the adoption of GPU-equipped edge devices and servers has rapidly increased, enabling more efficient real-time AI and machine learning applications at the edge. In this section, to further validate the effectiveness of the Galaxy+ framework in more resource-rich edge environments, we evaluate its performance on both mobile GPU-equipped edge device clusters and clusters of consumer-grade Nvidia GPUs. We introduced two new edge environments, C and D in Table VI. Environment C consists of four COTS Nvidia Jetson Nano [46] edge devices connected via a switch, with a D2D network bandwidth of 500 Mbps. Environment D is an edge server node with four consumer-grade Nvidia GeForce GTX 1080Ti GPUs [47], connected via a PCIe switch. We also included a new decoder-only model, OPT-XL, with 2.7B parameters in our experiments. We reported the average per-token processing latency for Galaxy+ and

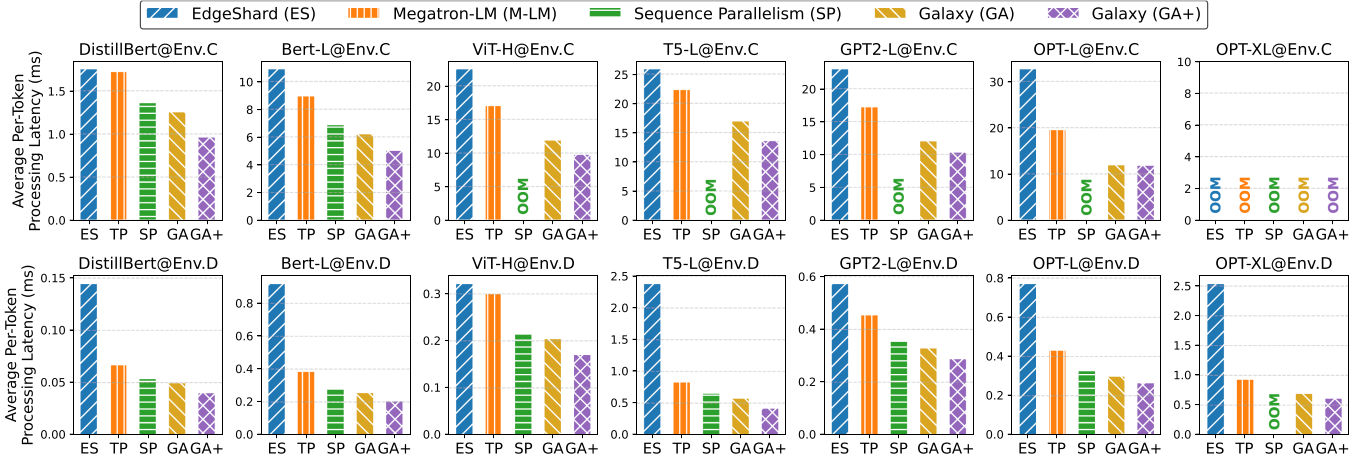


Fig. 13. Performance comparison of Galaxy+ and baseline methods in GPU-equipped edge environments.

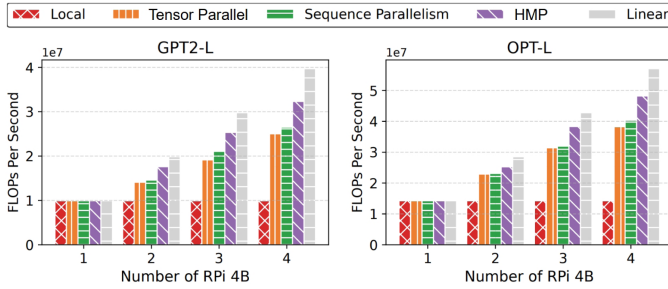


Fig. 14. Performance under weak scaling setup.

the baseline methods in environments C and D, as shown in Fig. 13. Galaxy+ consistently demonstrates lower per-token processing latency on both the Jetson Nano and Nvidia 1080Ti clusters, achieving up to a $3.01\times$ speedup on the Jetson Nano cluster and up to a $3.62\times$ speedup on the Nvidia 1080Ti cluster.

E. Scalability Analysis

To explore the scalability of our proposed hybrid model parallelism (HMP) architecture, we conducted both weak and strong scaling experiments in edge environment A (1000 Mbps) and compared it with two classic parallel architectures, tensor parallelism (TP) and sequence parallelism (SP), both of which can reduce single-shot Transformer inference latency. To obviate the impact of OOM errors on our experimental observations, we load and repeatedly perform inference on one single layer, rather than loading entire model.

1) *Weak Scaling*: In a weak scaling setup, the global workload increases proportionally with the number of devices. We set a weak scaling with a fixed sequence length of 96 per device (e.g. sequence length is equal to 384 for 4 RPi-M). The overall system's floating-point operations per second (FLOPS) are then evaluated. As depicted in Fig. 14, we observe excellent scaling performance of Galaxy+ in both GPT2-L and OPT-L. Specifically, the GPT2-L case with 4-way (four RPi-M) HMP can achieve 82% of linear scaling while the OPT-L case with 4-way can achieve 85% of linear scaling. Compared to the baseline methods, Galaxy+ exhibits superior scalability in terms of

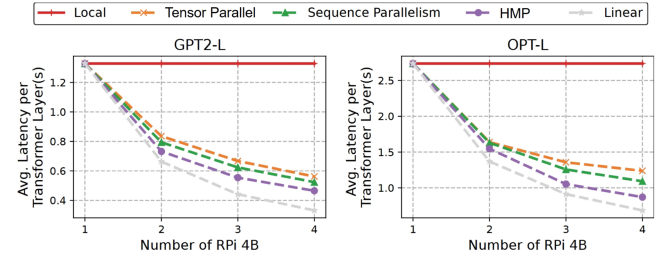


Fig. 15. Performance under strong scaling setup.

FLOPS as the number of devices increases, demonstrating its more efficient resource utilization.

2) *Strong Scaling*: In a strong scaling setup, the global workload is independent of the number of participating devices. We fix the sequence length to a constant value of 384. As depicted in Fig. 15, we measure the average inference latency per Transformer layer for a varying number of edge devices. Galaxy+ also demonstrates superior scalability under a strong scaling setup. Specifically, Galaxy+ achieves $2.9\times$ inference latency reduction compared to Local Inference in GPT2-L case, while achieving $3.12\times$ inference latency reduction compare to Local Inference in OPT-L case.

F. On-the-Fly HMP Re-Scheduling

We evaluate our on-the-fly fault-tolerant re-scheduling module on the GPT-L model using a realistic edge cluster consisting of one RPi-M and two RPi-L devices. As shown in Fig. 16, we applied an external CPU workload consisting of mobile DNN training and inference on one of the RPi-L devices at the 50-th timestamp (Fig. 16(a)), simulating resource contention caused by another edge intelligent application. Without fault-tolerant re-scheduling, the external CPU workload significantly reduces this device's computational capacity for collaborative inference, making it a straggler. As a result, it starves the other two devices and severely degrades the system's overall parallel efficiency (Fig. 16(b) and (c)), leading to a significant drop in overall inference throughput (Fig. 16(d)). In contrast, with our re-scheduling module, the collaborative inference system temporarily removes

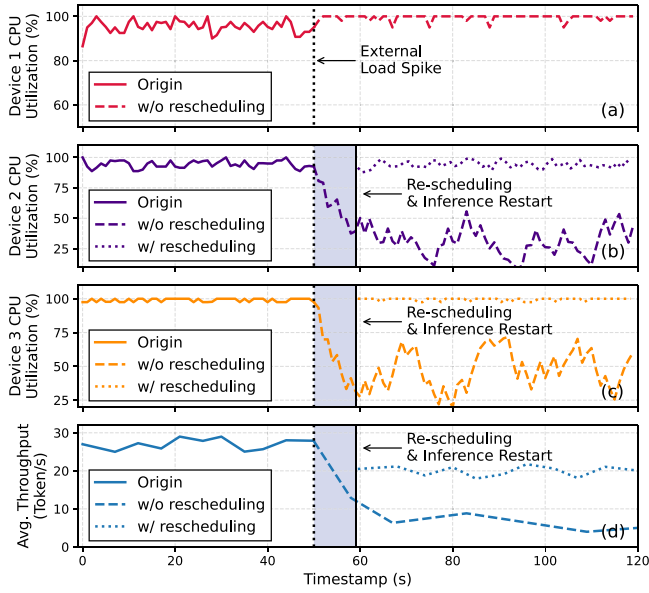


Fig. 16. When Device 1 experiences an external load spike, Galaxy+’s on-the-fly fault-tolerant module will swiftly re-schedule and restore inference.

the unavailable devices and redistributes its workload to remaining devices. Removing the straggler device improve resource utilization of the remaining nodes, significantly enhances overall inference throughput (Fig. 16(d)).

V. RELATED WORK

Edge AI with Transformer-Based LLMs. Transformer-based LLMs have driven diverse intelligence in today’s edge intelligent applications [3], ranging from AI assistants in smart homes [4] and voice-controlled robots in smart factories [5] to intelligent traffic management systems and urban planning tools in smart cities [6]. Current LLM-empowered edge applications predominantly rely on cloud data centers for deployment [48], [49]. However, a recent survey on LLM-based edge applications [11] indicates that more than 80% of industry experts advocate for personal LLMs to be fully or primarily deployed at the edge, emphasizing the critical need for privacy-preserving model inference. Consequently, emerging research efforts have begun to explore direct deployment and serving of Transformer-based LLMs on local edge devices [14].

On-Device Inference Acceleration. Deploying DNN on edge devices not only reduces latency and enhances user experience but also addresses privacy concerns by enabling local data processing. Pipe-It and Asymo [18], [19] scheduling workload according to the computing power of asymmetry mobile CPU cores to achieve higher throughput. BlastNet, CoDL and μ layer [12], [20], [50] perform a collaborative DNN inference on mobile CPU and GPU concurrently. Band [51] coordinates multi-DNN inference on heterogeneous mobile processors. Recent advances have explored deploying Transformer-based LLMs on edge devices, with research efforts primarily focusing on both algorithmic optimization and system-level enhancement. From an algorithmic perspective, significant progress has been made in reducing computational complexity through model

quantization [52], structural pruning [53], and knowledge distillation [54]. Concurrently, system-level approaches have emerged to fully leverage the computational potential of edge devices. Mllm-NPU [55] focuses on the prevalent decoder-only transformer architecture of LLMs, introducing an inference system optimized for efficient mobile NPU offloading. PowerInfer and PowerInfer-2 [56], [57] introduce a NPU-GPU-CPU hybrid inference system designed to accelerate computations on a single personal computer or mobile device.

Collaborative Execution of Transformer. Data Parallelism [30], [58] is the most extensively used distributed training approach in datacenters. Pipeline Parallelism is further proposed to conquer the memory issues of training large-scale transformer-based models [31], [35], but suffers from pipeline bubbles. Model Parallelism simultaneously tackles both memory and bubble issues, and is widely used in both training [34], [35], [59] and inference [7], [28], [60] tasks at datacenters. However, few of above approaches are designed for in-situ Transformer-based model inference at the edge. In addition to running entirely in cloud datacenter, pioneering research has explored leveraging edge devices as a collaborative resource pool to accelerate Transformer training and inference directly at the edge. Asteroid [61] introduces a pioneering distributed edge training system to accelerate full-parameter tuning for both CNNs and Transformer-based LLMs. Pluto and Charon [62] further integrate parameter-efficient fine-tuning techniques into edge collaborative training to reduce computational and memory demands. Galaxy [37] presents a collaborative edge AI system for low-latency Transformer-based model inference, enabling real-time in-situ intelligent services. DeTransformer [63] introduces a decoupled layer to minimize tensor synchronization overhead, but it necessitates full parameter tuning of the target model’s parameters.

Communication Optimization for Distributed Deep Learning. ZeRO++ [64] utilizes quantized communication to reduce the overhead of tensor synchronization. Hermes [65] applies model structured pruning techniques to achieve communication volume reduction. FeS and AdaFL [66], [67] utilize parameter-efficient fine-tuning techniques to reduce the number of parameters that need synchronization during distributed federated learning of Transformer-based DNNs, thereby minimizing communication overhead. CoCoNet, ASE, and Liger [68], [69], [70] leverage computation-communication overlap to mitigate the impact of communication latency on system performance.

VI. CONCLUSION

This paper introduces Galaxy+, an innovative collaborative in-situ Transformer inference system featuring a hybrid model parallelism architecture, a heterogeneity and memory-budget aware parallelism planning algorithm, and a tile-based communication optimization. Our extensive evaluation shows that Galaxy+ achieves a $1.2\times$ to $4.24\times$ performance improvement over state-of-the-art parallel inference methods across various edge environment setups and network bandwidth conditions. Galaxy+ can also adapt to device-level resource dynamics, swiftly rescheduling and restoring inference.

REFERENCES

- [1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
- [2] A. Radford et al., "Improving language understanding by generative pre-training," 2018.
- [3] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile edge intelligence for large language models: A contemporary survey," *IEEE Commun. Surv. Tuts.*, early access, Jan. 09, 2025, doi: [10.1109/COMST.2025.3527641](https://doi.org/10.1109/COMST.2025.3527641).
- [4] E. King, H. Yu, S. Lee, and C. Julien, "Sasha: Creative goal-oriented reasoning in smart homes with large language models," 2023, *arXiv:2305.09802*.
- [5] S. H. Vemprala, R. Bonatti, A. Buckner, and A. Kapoor, "ChatGPT for robotics: Design principles and model abilities," *IEEE Access*, 2024.
- [6] S. Masri, H. I. Ashqar, and M. Elhenawy, "Large language models (LLMs) as traffic control systems at urban intersections: A new paradigm," *Vehicles*, vol. 7, no. 1, 2025, Art. no. 11.
- [7] Z. Li et al., "[AlpaServe]: Statistical multiplexing with model parallelism for deep learning serving," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2023, pp. 663–679.
- [8] J. Fang, Y. Yu, C. Zhao, and J. Zhou, "TurboTransformers: An efficient GPU serving system for transformer models," in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2021, pp. 389–402.
- [9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.
- [10] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed., vol. 10, Cham, Switzerland: Springer, 2017.
- [11] Y. Li et al., "Personal LLM agents: Insights and survey about the capability, efficiency and security," 2024, *arXiv:2401.05459*.
- [12] F. Jia et al., "CoDL: Efficient CPU-GPU co-execution for deep learning inference on mobile devices," in *Proc. ACM Int. Conf. Mobile Syst., Appl., Serv.*, New York, NY, USA, 2022, pp. 209–221.
- [13] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.
- [14] G. Bai et al., "Beyond efficiency: A systematic survey of resource-efficient large language models," 2024, *arXiv:2401.00625*.
- [15] S. Ye, L. Zeng, Q. Wu, K. Luo, Q. Fang, and X. Chen, "Eco-FL: Adaptive federated learning with efficient edge collaborative pipeline training," in *Proc. 51st Int. Conf. Parallel Process.*, 2022, pp. 1–11.
- [16] Raspberry Pi Foundation, "Raspberry Pi 4 model B specifications," 2024. Accessed: Oct. 17, 2024. [Online]. Available: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>
- [17] NVIDIA Corporation, NVIDIA A100 tensor core GPU, 2020, Accessed: Oct. 13, 2024. [Online]. Available: <https://www.nvidia.com/en-us/data-center/a100/>
- [18] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "Asymo: Scalable and efficient deep-learning inference on asymmetric mobile cpus," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 215–228.
- [19] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput CNN inference on embedded arm big. little multicore processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2254–2267, Oct. 2020.
- [20] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, " μ layer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1–15.
- [21] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.
- [22] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019, Art. no. 9.
- [23] A. Vaswani et al., "Attention is all you need," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
- [24] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.
- [25] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2020.
- [26] S. Zhang et al., "OPT: Open pre-trained transformer language models," 2022, *arXiv:2205.01068*.
- [27] R. Bhardwaj et al., "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. 19th Symp. Netw. Syst. Des. Implementation*, 2022, pp. 119–135.
- [28] R. Y. Aminabadi et al., "Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2022, pp. 646–660.
- [29] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 22941–22954.
- [30] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 19–27.
- [31] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 103–112.
- [32] D. Narayanan et al., "Pipedream: Generalized pipeline parallelism for DNN training," in *Proc. Symp. Operating Syst. Princ.*, 2019, pp. 1–15.
- [33] M. Shoybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*.
- [34] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence parallelism: Long sequence training from system perspective," 2021, *arXiv:2105.13120*.
- [35] D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–15.
- [36] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," 2018, *arXiv:1802.05799*.
- [37] S. Ye et al., "Galaxy: A resource-efficient collaborative edge AI system for in-situ transformer inference," 2024, *arXiv:2405.17245*.
- [38] "Pytorch," 2019. [Online]. Available: <https://github.com/pytorch/pytorch>
- [39] G. Gerganov, "LLaMA.cpp: Port of Facebook's LLaMA model in C/C++," 2023, Accessed: Oct. 17, 2024. [Online]. Available: <https://github.com/ggerganov/llama.cpp>
- [40] X. Jiang et al., "MNN: A universal and efficient inference engine," in *Proc. Mach. Learn. Syst.*, pp. 1–13, 2020Mnn: A universal and efficient inference engine.
- [41] "Tensorflow-lite," 2021. [Online]. Available: <https://www.tensorflow.org/lite/examples>
- [42] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," 2019, *arXiv:1910.01108*.
- [43] 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," 2018, *arXiv:1804.07461*.
- [44] O. Vinyals et al., "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3637–3645.
- [45] M. Zhang, J. Cao, X. Shen, and Z. Cui, "Edgeshard: Efficient LLM inference via collaborative edge computing," 2024, *arXiv:2405.14371*.
- [46] "Jetson-nano," 2019. [Online]. Available: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>
- [47] NVIDIA Corporation, "Nvidia geforce GTX 1080 Ti graphics card," 2017, Accessed: Oct. 24, 2024. [Online]. Available: <https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/>
- [48] Y. Fu et al., "Serverlessllm: Low-latency serverless inference for large language models," in *Proc. 18th USENIX Symp. Operating Syst. Des. Implementation*, USENIX Association, 2024, pp. 135–153.
- [49] B. Wu et al., "Fast distributed inference serving for large language models," 2023, *arXiv:2305.05920*.
- [50] N. Ling, X. Huang, Z. Zhao, N. Guan, Z. Yan, and G. Xing, "BlastNet: Exploiting duo-blocks for cross-processor real-time DNN inference," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.*, 2022, pp. 91–105.
- [51] J. S. Jeong et al., "Band: Coordinated multi-DNN inference on heterogeneous mobile processors," in *Proc. 20th Int. Conf. Mobile Syst., Appl., Serv.*, 2022, pp. 235–247.
- [52] J. Lin et al., "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proc. Mach. Learn. Syst.*, 2024, pp. 87–100.
- [53] X. Ma, G. Fang, and X. Wang, "LLM-pruner: On the structural pruning of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 21702–21720.
- [54] Y. Yu et al., "EchoLM: Accelerating LLM serving with real-time knowledge distillation," 2025, *arXiv:2501.12689*.
- [55] D. Xu et al., "Empowering 1000 tokens/second on-device LLM prefilling with MLLM-NPU," 2024, *arXiv:2407.05858*.

- [56] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade GPU," 2023, *arXiv:2312.12456*.
- [57] Z. Xue, Y. Song, Z. Mi, L. Chen, Y. Xia, and H. Chen, "Powerinfer-2: Fast large language model inference on a smartphone," 2024, *arXiv:2406.06282*.
- [58] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRo: Memory optimizations toward training trillion parameter models," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 1–16.
- [59] V. A. Korthikanti et al., "Reducing activation recomputation in large transformer models," in *Proc. Mach. Learn. Syst.*, 2023.
- [60] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-based} generative models," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2022, pp. 521–538.
- [61] S. Ye, L. Zeng, X. Chu, G. Xing, and X. Chen, "Asteroid: Resource-efficient hybrid pipeline parallelism for collaborative DNN training on heterogeneous edge devices," in *Proc. 30th Annu. Int. Conf. Mobile Comput. Netw.*, 2024, pp. 312–326.
- [62] B. Ouyang, S. Ye, L. Zeng, T. Qian, J. Li, and X. Chen, "Pluto and Charon: A time and memory efficient collaborative edge AI framework for personal LLMs fine-tuning," in *Proc. 53rd Int. Conf. Parallel Process.*, 2024, pp. 762–771.
- [63] Y. Wei et al., "Communication-efficient model parallelism for distributed in-situ transformer inference," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2024, pp. 1–6.
- [64] G. Wang et al., "ZeRO : Extremely efficient collective communication for giant model training," 2023, *arXiv:2306.10209*.
- [65] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: An efficient federated learning framework for heterogeneous mobile clients," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 420–437.
- [66] D. Cai, S. Wang, Y. Wu, F. X. Lin, and M. Xu, "Federated few-shot learning for mobile NLP," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–17.
- [67] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "Efficient federated learning for modern NLP," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–16.
- [68] A. Jangda et al., "Breaking the computation and communication abstraction barrier in distributed machine learning workloads," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2022, pp. 402–416.
- [69] S. Rashidi et al., "Enabling compute-communication overlap in distributed deep learning training platforms," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Architecture*, 2021, pp. 540–553.
- [70] J. Du et al., "Liger: Interleaving intra-and inter-operator parallelism for distributed large model inference," in *Proc. 29th ACM SIGPLAN Annu. Symp. Princ. Pract. Parallel Program.*, 2024, pp. 42–54.



Shengyuan Ye (Graduate Student Member, IEEE) received the BE degree from the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou, China, in 2021. He is currently working toward the PhD degree with Sun Yat-sen University. His current research interests include mobile edge computing, resource-efficient mobile AI systems, distributed machine learning systems.



Bei Ouyang received the BS degree in computer science from the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou, China, in 2023. She is currently working toward the master's degree with the School of Computer Science and Engineering, Sun Yat-sen University. Her research interests include mobile edge computing and distributed computing.



Jiangsu Du received the BSc degree from the Wuhan University, in 2016, the MSc degree from the Edinburgh Parallel Computing Center, University of Edinburgh, in 2017, and the PhD degree from the School of Computer Science and Engineering, Sun Yat-sen University, in 2022. He is now the postdoctoral researcher with the School of Computer Science and Engineering, Sun Yat-sen University. His research interests focus on high performance computing, parallel and distributed artificial intelligent system.



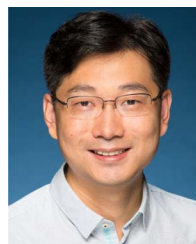
Liekang Zeng received the BE and PhD degrees from Sun Yat-sen University, Guangzhou, China. His current research interests include edge intelligence, mobile computing, and distributed machine learning systems.



Tianyi Qian received the BS degree in computer science and technology from the Honors College of Engineering Interdisciplinary Excellence Program, Northwestern Polytechnical University (NPU), Xi'an, China, in 2023. She is currently working toward the master's degree with the School of Computer Science, Sun Yat-sen University (SYSU), Guangzhou, China. Her research interests include mobile edge computing and distributed computing.



Wenzhong Ou received the BE degree from the School of Electronics and Information Technology (School of Microelectronics), Sun Yat-sen University, and the MS degree in computer science from the School of Computer Science, Sun Yat-sen University (SYSU), Guangzhou, China, in 2024. He is currently working on Mobile Edge Computing and Interactive Large Language Model Application.



Xiaowen Chu (Fellow, IEEE) received the BEng degree in computer science from Tsinghua University, Beijing, China, in 1999, and the PhD degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, in 2003. He is currently a professor with the Data Science and Analytics Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China. His current research interests include Graphics Processing Unit (GPU) computing, distributed machine learning, and wireless networks.



Deke Guo (Senior Member, IEEE) received the BS degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the PhD degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a professor with the College of System Engineering, National University of Defense Technology. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks.



Xu Chen received the PhD degree in information engineering from the Chinese University of Hong Kong, in 2012. He is a full professor with Sun Yat-sen University, Guangzhou, China, and the vice director with National and Local Joint Engineering Laboratory. His research interests include edge computing, edge intelligence, edge robotics, AI for networking, mobile social networks game theory, deep learning, dynamic optimization, and resource allocation.



Yutong Lu (Member, IEEE) received the MSc and PhD degrees in computer science from the National University of Defense Technology (NUDT), Changsha, China. She is currently a professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. She is also a director with National Supercomputer Center, Guangzhou. Her research interests include parallel system management, high-speed communication, distributed file systems, and advanced programming environments with the MPI.