EdgeFormer: Latency-Aware Collaborative Multi-Head Attention of Transformer Inference in Edge Networks

Anonymous ACL submission

Abstract

Recent breakthroughs in Transformer-based large models, have driven widespread tasks, yet their reliance on centralized cloud deployment raises significant privacy risks due to sensitive data exposure. While edge-based collaborative inference offers a privacy-preserving alternative, existing methods face critical limitations: static model partitioning cannot adapt to dynamic edge resource fluctuations, and rigid multi-head attention handling overlooks semantic-critical prioritization and parallelism. We propose EdgeFormer, a latency-aware framework for distributed Transformer inference in resource-constrained edge networks. EdgeFormer dynamically allocates model blocks across devices via efficiency-storage trade-off optimization and introduces collaborative Multi-Head Attention (cMHA), which distributes semantic-critical attention heads across devices while pruning redundant ones under real-time constraints. We further develop LiScore, a composite metric integrating attention diversity and latency costs, alongside a similarity-based retrieval method to reduce recomputation overhead. Extensive experiments demonstrate that EdgeFormer achieves up to 2.01× inference acceleration over state-of-the-art baselines with ≤1.06% accuracy loss, maintaining robustness under varying edge conditions.

1 Introduction

Recent advancements in large language models (LLMs), such as GPT-4 (Achiam et al., 2023), LLaMA (Touvron et al., 2023) and DeepSeek (Guo et al., 2025), have shown remarkable generalization across diverse tasks, e.g., question answering, semantic understanding,and automated code generation, propelling their adoption in real-world applications (Liu et al., 2023b). Most LLM services currently rely on centralized cloud infrastructures, requiring users to upload data to remote servers for inference (Lazuka et al., 2024). While this paradigm benefits from strong cloud capabilities, it poses serious privacy risks, as sensitive data is transmitted to and processed by third-party platforms (Das et al., 2025; Pan et al., 2020). To mitigate these concerns, edge deployment has emerged as a privacy-preserving alternative (Hou et al., 2021). However, edge devices often lack sufficient computational and storage resources, making such deployment challenging in constrained environments (e.g., mobile laptops or terminals) (Schlegel et al., 2022).

A promising solution lies in distributed collaborative inference across trusted edge devices (Li et al., 2022; Malka et al., 2024), which harmonizes privacy protection with efficient resource utilization. The central challenge in this paradigm is to systematically exploit the inherent parallelism of LLMs, particularly their Transformer architectures, while dynamically adapting to volatile edge environments characterized by fluctuating bandwidth and heterogeneous device capabilities (Golpayegani et al., 2024). Achieving this requires innovative strategies to partition, allocate, and schedule computational workloads across devices (Zeng and Chen, 2020; Chen et al., 2024b), ensuring both high inference efficiency and strict adherence to model accuracy constraints.

Existing research on distributed edge inference has made strides in resource allocation (Liu et al., 2023a; Cai et al., 2024), model partition (Yang et al., 2022) and compression (Wang et al., 2024; Yu et al., 2024), yet critical gaps persist in addressing the unique demands of Transformer-based models. Prior efforts often focus on task offloading based on static device capabilities (Chen et al., 2024a) or rely on simplified model architectures, such as DNNs (Li et al., 2024) and CNNs (Chen et al., 2024b), which overlook the complexity and granularity of Transformer workloads. Notably, static partitioning approaches (Hu and Li, 2024; Ye et al., 2024) allocate Transformer blocks in a fixed manner, failing to account for dynamic fluctuations

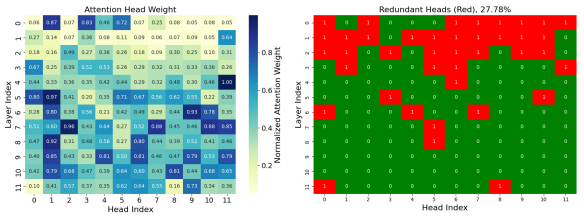
in edge resources — a key limitation for latency-sensitive inference. Furthermore, many existing frameworks treat multi-head attention (MHA) layers as indivisible components of the Transformer structure (Cai et al., 2023; Wei et al., 2024), missing the opportunity to dynamically distribute attention heads across devices in response to real-time resource variability.

To address these limitations, we propose **EdgeFormer**, a framework for distributed collaborative Transformer inference on the edge. EdgeFormer optimizes the deployment of Transformer blocks across heterogeneous devices by formulating block allocation as a constrained optimization problem, prioritizing devices with high computational efficiency per unit storage. Furthermore, it introduces **cMHA**, a novel collaborative Multi-Head Attention mechanism that distributes attention heads across devices to exploit parallelism while adhering to storage and latency constraints. cMHA dynamically allocates heads to local and remote computation based on resource conditions.

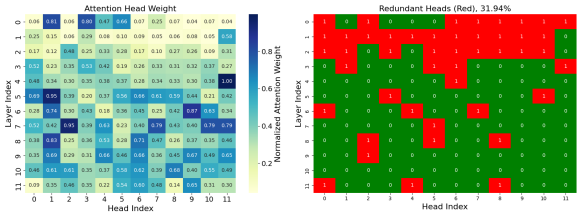
While cMHA improves resource utilization, frequent cross-device exchanges of Q/K/V projections and attention results introduce nontrivial communication overhead. To mitigate this, we analyze the distribution of attention head weights across diverse input types, as illustrated in Fig. 1, drawing inspiration from structured pruning techniques (Xia et al., 2022; Fang et al., 2023). For a short input (Fig. 1a), lower layers (1-4) exhibit low attention weight variance (with 27.78% of heads identified as prunable), indicating redundancy and limited token correlation due to shallow feature extraction. In contrast, for a long input (Fig. 1b), the pruning potential rises to 31.94%, revealing increased input-dependent sparsity as more tokens contribute redundant information. These suggest that heads with low attention weights and variances contribute minimally to model performance, making them ideal candidates for dynamic pruning under resource constraints. These raise a pivotal question:

Can we dynamically identify and prioritize semantically important heads under fluctuating edge resources to maximize inference efficiency without sacrificing accuracy?

To answer this, we introduce **LiScore**, a Latency-aware head importance Score that jointly captures *semantic contribution* (quantified via attention diversity, that is measured by the variance of attention weights across tokens) and *execution latency* (estimated from local and remote computation costs



(a) Short Input: "*Tracy Morgan hasn't appeared on stage since ...*". Lower layers (1-4): High redundancy (prunable heads: 20.83%), with low attention weight variance, indicating weak token correlation. Higher layers (>4): Sparse redundancy (prunable: <7%), reflecting specialized semantic roles.



(b) Long Input: "*For four years we have waited expectantly for the pitter patter of tiny paws. Soon, that wait could finally be over ...*". Lower layers (1-4): Increased pruning potential (22.22%) due to token abundance amplifying redundancy. Higher layers (>4): Critical heads dominate (prunable: 9.72%), emphasizing task-specific reasoning.

Figure 1: Attention redundancy analysis across layers in GPT2-Small for (a) short and (b) long input sequences from the ReCoRD dataset of SuperGLUE(Wang et al., 2019).

based on device capabilities and network conditions). We formulate head allocation as a combinatorial optimization problem that seeks to maximize this score under storage and latency constraints. Given its NP-hard, a lightweight heuristic algorithm is designed to iteratively assign critical heads to optimal devices and prune redundant ones. To further minimize the overhead of continuously recalculating head importance, we propose *a similarity-based retrieval mechanism* that leverages historical Q-K interaction patterns and feature statistics to reuse precomputed LiScores for similar heads. Extensive experiments demonstrate obvious inference speedups over state-of-the-arts with minimal accuracy degradation. *Code is at https://anonymous.4open.science/r/EdgeFormer-E3C8.*

The main contributions are summarized as:

- We propose EdgeFormer, a distributed inference framework for Transformer-based models that optimizes block allocation across heterogeneous edge devices via adaptive efficiency-storage prioritization, while enabling collaborative multi-head attention through cross-device parallelism.
- We formulate dynamic head allocation as a combinatorial optimization problem and solve it using a two-stage heuristic that combines greedy

prioritization with iterative pruning, effectively balancing semantic integrity and real-time constraints.

• We introduce a latency-aware composite importance metric that balances semantic importance and execution efficiency, alongside a hierarchical similarity-based retrieval mechanism that reuses precomputed scores from analogous historical heads to reduce recomputation overhead.

• Extensive experiments demonstrate that EdgeFormer achieves up to 2.01× inference speedup over state-of-the-art frameworks, with ≤1.06% accuracy loss on benchmark tasks. Visualization analysis further validates the framework's effectiveness in collaborative head attention.

Due to space limitations, we defer a detailed discussion of related work to Appendix A.

2 System Model

2.1 System Overview

1) System Composition

The system comprises K devices, $\mathcal{D} = \{D_1, D_2, ..., D_K\}$. Each device D_k is characterized by:

$$C_k = (s_k, f_k), \tag{1}$$

where s_k denotes its storage capacity (*measured by the maximum number of continuous Transformer blocks it can store*), and f_k represents its computational capability (FLOPS).

2) Model Architecture

The LLM consists of L identical *Transformer blocks* [1]. Each block $l \in \{1, 2, ..., L\}$ accepts and outputs a hidden state tensor of shape $n_{\text{ctx}} \times d_{\text{model}}$, where n_{ctx} is the context length (number of tokens), and d_{model} is the hidden state dimension.

Within each block l, there are Q/K/V projection maxtrices $W^l_{\text{q/k/v}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$, and an output projection matrix $W^l_o \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$. Also, there are H heads, where each head $h \in \{1, \ldots, H\}$ has the dimension $d_{\text{head}} = \frac{d_{\text{model}}}{H}$.

2.2 Block Allocation

To balance computational efficiency and storage constraints, we formulate block allocation as a constrained optimization problem, with objective of minimize the total computation latency while enforcing device storage limits and block continuity:

$$\begin{aligned}
&\min_{M_k} \textstyle\sum_{l=1}^{L} \frac{C_l}{f_{k(l)}} && (2)\\
&\text{s.t. } \forall k,\ |M_k| \le s_k && \text{(Storage Capacity)}\\
&\quad b^{(k)}_{\text{start}} = b^{(k-1)}_{\text{end}} + 1,\ \forall k > 1, && \text{(Continuity)}\\
&\quad \cup_{k=1}^{K} M_k = \{1, 2, \ldots, L\}, && \text{(Completeness)}\\
&\quad M_k = && \\
&\quad [b^{(k)}_{\text{start}}, b^{(k)}_{\text{end}}], b^{(k)}_{\text{start}}, b^{(k)}_{\text{end}} \in \mathbb{Z}^+ && \text{(Integer Interval)}
\end{aligned}$$

where C_l is the computational workload of block l (fixed for all blocks), $f_{k(l)}$ is the computational capability (FLOPS) of the device assigned to block l, and $M_k = [b^{(k)}_{\text{start}}, b^{(k)}_{\text{end}}]$ is the continuous block interval assigned to device k. The optimization is solved with the following two steps to allocate each device a partition of continuous blocks, as in Fig 2.

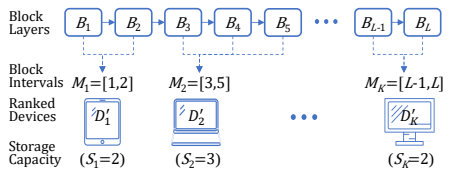


Figure 2: Transformer block partition based on device prioritization (computational capability per unit storage).

1) Device Prioritization ranks devices by their efficiency $\eta_k = \frac{f_k}{s_k}$, which quantifies *computational capability per unit storage*. We can get the sorted device set:

$$\mathcal{D}' = \{D'_1, ..., D'_K\}, \forall i < j \in \{1, ..., K\}, \eta_i > \eta_j.$$

2) Greedy Allocation sequentially assigns the longest feasible continuous block interval to the highest-priority device:

$$|M_k| = \min\left(s_k,\ L - \textstyle\sum_{j=1}^{k-1} |M_j|\right), \forall D'_k \in \mathcal{D}'.$$

These steps minimize $\sum_{l=1}^{L} \frac{C_l}{f_{k(l)}}$ under greedy constraints, w.r.t. storage and continuity requirements.

2.3 Collaborative Attention

We detail collaborative MHA workflow and formulations in Appendix B.1, due to space limitations.

2.4 System Cost

The total latency for each block l is calculated by:

$$T^l = T^l_{QKV} + T^l_{\text{collab-att}} + T^l_{W_o+\text{Norm}}. \tag{3}$$

In the following, the superscript l in T is omitted for simplicity.

[1] The input token embedding and output logits projection are omitted for brevity, since this work focuses on collaboratively computing the most resource-intensive blocks. They could be deployed in any D_k to accept user input.

1) Q/K/V Projection Latency: It depends on the matrix multiplication in Eq. 18 as follows,

$$T_{QKV} = \frac{3d_{\text{model}}^2}{f_k}. \tag{4}$$

2) Collaborative Attention Latency: Let device k execute block l, and define:

- $\mathcal{H}_k$: Set of attention heads **locally computed** on D_k.
- $\mathcal{H}_{k'}$: Set of heads **offloaded** to a neighbor device $D_{k'}$.

The latency splits into two components: *One is the local head computation*

$$T_{\text{local-att}} = T_{\text{local-att}}(b) \cdot |\mathcal{H}_k|, \tag{5}$$

$$\text{where} \quad T_{\text{local-att}}(b) = \frac{n_{\text{ctx}}^2 d_{\text{head}}}{f_k}, \quad \forall b \in \mathcal{H}_k \tag{6}$$

The other is the remote head computation for all offloaded heads $\mathcal{H}_{k'}$ to device $D_{k'}$:

$$T_{\text{remote-att}}(k') = T_{\text{remote-att}}(k', b) \cdot |\mathcal{H}_{k'}|, \; b \in \mathcal{H}_{k'} \tag{7}$$

$$T_{\text{remote-att}}(k',b) = \underbrace{\frac{3d_{\text{head}} n_{\text{ctx}}}{B_{k\to k'}}}_{\text{Send QKV}} + \underbrace{\frac{n_{\text{ctx}}^2 d_{\text{head}}}{f_{k'}}}_{\text{Compute on } D_{k'}} + \underbrace{\frac{d_{\text{head}} n_{\text{ctx}}}{B_{k'\to k}}}_{\text{Receive Result}} \tag{8}$$

Considering all collaborating neighbors, the overall remote attention latency is

$$T_{\text{remote-att}} = \max_{D_{k'}} T_{\text{remote-att}}(k'). \tag{9}$$

Hence, the final collaborative attention latency is

$$T_{\text{collab-att}} = \max\left(T_{\text{local-att}}, T_{\text{remote-att}}\right). \tag{10}$$

3) Local Post-Processing Latency:

$$T_{W_o+\text{Norm}} = \frac{d_{\text{model}}^2}{f_k} + \frac{d_{\text{model}}}{f_k}. \tag{11}$$

With T^l, the total end-to-end latency (omitting the input token embedding and output logit projection) of once inference is

$$T_{\text{total}} = \textstyle\sum_{l=1}^{L} T^l.$$

3 Method

Based on the system model, this section will detail the key component (cMHA) of EdgeFormer.

3.1 Overview

The cMHA aims to optimize the execution of multi-head attention layers in Transformer blocks by dynamically selecting attention heads for local computation, offloading to neighbor devices, or pruning, based on their semantic importance and system efficiency. This approach balances two critical objectives:

• Semantic Preservation: Prioritize heads contributing significantly to model accuracy through attention diversity.

• Latency Minimization: Reduce end-to-end inference latency by leveraging edge devices' heterogeneous resources.

Specifically, cMHA operates in three key phases. 1) Latency-Aware Head Importance, integrates semantic importance (via attention variance) and execution latency (local/remote cost). 2) Dynamic Importance Management, reuses precomputed importance scores to reduce overhead. 3) Head Allocation and Pruning, take a heuristic to assign heads to devices under storage and latency constraints, with iterative pruning to meet real-time requirements.

3.2 Latency-Aware Head Importance

To dynamically select each heads for local/remote computing or pruning while balancing semantic contributions and system efficiency, we introduce a **latency-aware importance metric** that integrates both attention diversity and execution latency.

For the b-th head in an MHA layer, let its attention weight matrix $\boldsymbol{A}_b \in \mathbb{R}^{n_{\text{ctx}} \times n_{\text{ctx}}}$ be:

$$\boldsymbol{A}_b = \text{softmax}\left(\frac{\boldsymbol{Q}_b \cdot \boldsymbol{K}_b^\top}{\sqrt{d_{\text{head}}}}\right), \tag{12}$$

where $\boldsymbol{Q}_b$ and $\boldsymbol{K}_b$ are query and key matrices of b. Semantically, the importance score $\boldsymbol{I}_b$ is computed by $\boldsymbol{A}_b$:

$$\boldsymbol{I}_b = \frac{1}{n_{\text{ctx}}^2} \sum_{x=1}^{n_{\text{ctx}}} \sum_{y=1}^{n_{\text{ctx}}} \left(A_b^{x,y} - \mu_b\right)^2 + H_b, \tag{13}$$

with $\mu_b = \frac{1}{n_{\text{ctx}}^2} \sum_{x,y} A_b^{x,y}$. The entropy-based importance of b-th head is defined as $H_b = -\mathbb{E}\left[\sum_i^{n_{\text{ctx}}} A_b^i \log A_b^i\right]$. Normalized scores $\boldsymbol{I}_b$ are derived as:

$$\boldsymbol{I}_b \leftarrow \frac{\boldsymbol{I}_b - \boldsymbol{I}_{\min}}{\boldsymbol{I}_{\max} - \boldsymbol{I}_{\min}}, \qquad \boldsymbol{I}_b \in [0, 1] \tag{14}$$

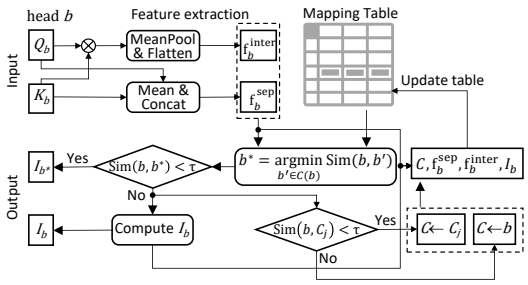


Figure 3: Workflow of importance retrival and update for a head b (Input: Q_b and K_b, Output: I_{b^*} or I_b).

where $\boldsymbol{I}_{\text{min}}$ and $\boldsymbol{I}_{\text{max}}$ are the minimum and maximum scores.

To jointly prioritize heads with high semantic contributions and low execution latency, we define a latency-aware composite importance score $\mathcal{S}_b$:

$$\mathcal{S}_b = \lambda \boldsymbol{I}_b + (1-\lambda)\frac{1}{T(b)}, \tag{15}$$

where $\lambda \in [0,1]$ balances accuracy and efficiency, and $T(b)$ is the attention latency[2] of head b executed locally ($T_{\text{local-att}}$ in Eq. 5) or remotely ($T_{\text{remote-att}}$ in Eq. 7) in $D_{k'}$.

This composite score prioritizes heads that are both semantic-critical and resource-efficient.

3.3 Importance Retrieval and Update

To minimize the computational overhead of recalculating head importance scores I_b for every head's hidden states (i.e., the Q_b and K_b in Eq. 12), we propose an approximate retrieval mechanism that reuses precomputed importance scores from semantically similar historical states. This part details the methodology for clustering hidden states, performing fast similarity search, and dynamically updating the importance mapping table, as shown in Fig. 3.

1) Head-Specific Feature Extraction

We capture unique patterns in $\{Q_b, K_b\}$ to define head-specific similarity. Specifically, for each head b, two feature vectors are extracted.

• Query-Key Interaction Feature ($\mathbf{f}_b^{\text{inter}}$):

$$\mathbf{f}_b^{\text{inter}} = (Q \odot K)\,\mathbf{1} \in \mathbb{R}^{n_{\text{ctx}}}, \qquad \mathbf{1} \in \mathbb{R}^{d_{\text{head}}},$$

representing the attention interaction of each token. Here, $\mathbf{1}$ is a all-ones vector.

• Query/Key Separable Feature ($\mathbf{f}_b^{\text{sep}}$):

$$\mathbf{f}_b^{\text{sep}} = \text{Concat}\big(\text{Mean}(Q_b), \text{Mean}(K_b)\big) \in \mathbb{R}^{2d_{\text{head}}}.$$

In this way, $\mathbf{f}_b^{\text{inter}}$ encodes head's attention focus, and $\mathbf{f}_b^{\text{sep}}$ preserves query/key characteristics.

[2]Note that the latency $T(b)$ not only depends on head b itself but also all others executed on the same device. Hence, $T(b)$ refers to Eq. 5 or Eq. 7, rather than Eq. 6 or Eq. 8.

2) Head-Conditioned Clustering

We group heads with similar $\{Q_b, K_b\}$ patterns to enable targeted retrieval, using hierarchical clustering (Fig. 4):

• Fold 1: Cluster heads into coarse groups using $\mathbf{f}_b^{\text{sep}}$ (Euclidean distance).

• Fold 2: Refine clusters within each group using $\mathbf{f}_b^{\text{inter}}$ (cosine similarity). Then, we get a feature-clustered importance mapping table:

$$\mathcal{M} \triangleq \left\{\left(C, \{\mathbf{f}_b^{\text{sep}}, \mathbf{f}_b^{\text{inter}}\}, I_b\right)\right\},$$

where the clusters reflect both separable and interactive head behaviors, and automatically adapt to diverse attention patterns (e.g., sparse vs. dense heads).

3) Dynamic Importance Reuse

For a new Q/K pair $\{Q_b, K_b\}$, we take the following three steps to achieve importance reuse.

• First, we perform head-level feature extraction, that computes $\mathbf{f}_b^{\text{inter}}$ and $\mathbf{f}_b^{\text{inter}}$ for head b (without confusion, b is taken to denote the head's two-feature sample in the following parts of Section 3.3).

• Second, we retrieve the most similar historical head b^* within its cluster:

$$b^* = \arg\min_{b' \in C(b)} \text{Sim}(b, b'), \tag{16}$$

where $\text{Sim}(b, b')$ is the similarity between b and b' defined as:

$$\text{Sim}(b, b') = \|\mathbf{f}_b^{\text{sep}} - \mathbf{f}_{b'}^{\text{sep}}\|_2 + \zeta\left(1 - \cos(\mathbf{f}_b^{\text{inter}}, \mathbf{f}_{b'}^{\text{inter}})\right), \tag{17}$$

whose uniqueness is proved in Theorem 1. Here, ζ weights the interaction importance of samples.

• Third, we reuse I_{b^*} for b, if their similarity $\text{Sim}(b^*, b) \le \tau$.

4) Head Importance Update

Due to space limitations, the detailed *head importance update* and *allocation strategies*, together with the *algorithms* and *theoretical analysis*, are provided in Appendix C.1, C.2, C.3 and C.4.

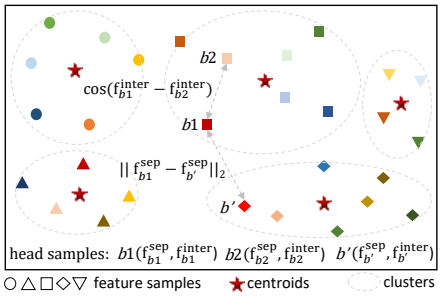


Figure 4: Hierarchical clustering of similar heads patterns.

4 Experiment

4.1 Experimental Setup

Models and Datasets. We evaluate EdgeFormer on four Transformer-based models: GPT2-Large (0.77B), OPT-1.3B, OPT-6.7B and OPT-13B (FP16), with specifications listed in Table 1. Experiments span the official validation (dev) splits of four SuperGLUE (Wang et al., 2019) tasks (ReCoRD, BoolQ, MultiRC, and WiC) to assess generalization across question answering, textual entailment, and sense disambiguation.

Method Parameters. We set the parameters of cMHA as follows, the accuracy-efficiency balance coeeficient $\lambda = 0.8$, interaction importance $\zeta = 0.6$, similarity threshold $\tau = 0.6$, and collaborative-attention latency threshold $T_{\text{max}} = 3.0$, respectively, otherwise specified elsewhere.

Testbed and Baselines. We build the EdgeFormer testbed using PyTorch and the NVIDIA collective communication library, consisting of three devices with equivalent compute performance and memory configuration, each equipped with approximately 13 TFLOPS of computational capability and 11 GB of memory (otherwise specified elsewhere). Deployed with the same testbed, we compared EdgeFormer against:

- **HFAccelerate** (Gugger et al., 2022): A Hugging Face library abstracting distributed PyTorch execution on heterogeneous hardware.
- **DeepSpeed** (Rasley et al., 2020): A deep learning optimization library that supports massive billion parameter models at scale by sharding model parameters across GPUs.
- **Galaxy** (Ye et al., 2024): A hybrid distributed parallel inference that integrates both tensor parallelism and sequence parallelism in fine-grained computation and communication tiles.

Evaluation Metrics. We employ three metrics:

- **Lat**: Average inference latency (seconds).
- **Acc**: Average task accuracy across samples.
- **Spd**: Latency reduction relative to baselines.

Table 1: Transformer-based Model Specifications.

Model	Layers	Heads	Dimension	Model Size
GPT2-L	36	20	1280	3.1 GB
OPT-1.3B	24	32	2048	5.2 GB
OPT-6.7B	32	32	4096	26 GB
OPT-13B (FP16)	40	40	5120	26 GB

4.2 Main Results across LLMs and Datasets

Table 2 presents a comprehensive performance comparison between EdgeFormer and baselines across multiple LLMs and datasets. For GPT2-L, EdgeFormer outperforms baselines, achieving latency reductions of up to 2.08× on ReCoRD, 2.07× on BoolQ, 1.91× on MultiRC, and 1.92× on WiC, resulting in a 2.01× overall speedup compared to HFAccelerate. These results highlight EdgeFormer's efficiency in optimizing distributed inference for resource-constrained edge networks, with minimal accuracy degradation (overall accuracy drop ≤1.06%, from 43.75% to 42.69%). In comparison, DeepSpeed and Galaxy show smaller speedups of 1.20× and 1.75×, respectively. As the LLM size increases to OPT-1.3B and OPT-6.7B, EdgeFormer continues to outperform the baselines, achieving overall speedups of 1.79× on OPT-1.3B and 1.96× on OPT-6.7B, demonstrating its scalability. In contrast, DeepSpeed and Galaxy show more modest improvements, with speedups of 1.06× and 1.40× on OPT-1.3B, and 1.10× and 1.40× on OPT-6.7B. Notably, for OPT-13B (FP16), EdgeFormer achieves an exceptional 1.98× speedup in overall latency compared to HFAccelerate, showcasing its robust scalability and efficient task distribution across devices for large-scale models.

4.3 Latency Breakdown

In Fig. 5, we detail the breakdown of the total latency of EdgeFormer into three stages: Head Pruning, Device Assignment, and Distributed Inference. Notably, Distributed Inference (Stage 3) dominates the overall latency across all datasets, accounting for over ≥90% of the total time. In contrast, Head Pruning (Stage 1) and Device Assignment (Stage 2) contribute only marginally, typically under 2% and 6%, respectively. For instance, in the ReCoRD dataset, Stage 1 and Stage 2 together account for just 4% of the total 2.89s. This pattern is consistent across datasets, indicating that the bulk of latency stems from executing the model itself rather than pre-inference dynamic planning.

4.4 Memory Overhead Analysis

In Fig. 6, we detail the memory overhead introduced by EdgeFormer on three homogeneous devices, decomposed into three components: operating system (**OS**), LLM parameters (**LLM**), and the mapping table (**MT**). As expected, LLM memory constitutes the majority of consumption, scaling

Table 2: Performance of EdgeFormer and baselines on various LLMs and datasets (Acc:%, Lat: s).

Method	ReCoRD		BoolQ		MultiRC		WiC		Overall	
	Acc	Lat (Spd)	Acc	Lat (Spd)	Acc	Lat (Spd)	Acc	Lat (Spd)	Acc	Lat (Spd)
GPT2-L										
HFAccelerate	32.48	3.18(×1.00)	38.97	2.61(×1.00)	54.95	1.99 (×1.00)	48.59	1.44(×1.00)	43.75	2.31(×1.00)
DeepSpeed	32.67	2.69(×1.18)	39.12	2.18(×1.20)	55.12	1.66 (×1.20)	48.75	1.16(×1.24)	43.92	1.92(×1.20)
Galaxy	32.44	1.84(×1.72)	38.87	1.49(×1.75)	54.83	1.14(×1.74)	48.12	0.79(×1.82)	43.57	1.32(×1.75)
EdgeFormer	31.88	**1.53(×2.08)**	37.64	**1.26(×2.07)**	54.33	**1.04 (×1.91)**	46.90	**0.75(×1.92)**	42.69	**1.15 (×2.01)**
OPT-1.3B										
HFAccelerate	41.36	5.21(×1.00)	69.25	4.07(×1.00)	52.63	3.28(×1.00)	50.43	2.15(×1.00)	53.42	3.68(×1.00)
DeepSpeed	40.92	4.94(×1.05)	69.35	3.81(×1.07)	53.01	3.08(×1.06)	50.90	2.02(×1.06)	53.55	3.46(×1.06)
Galaxy	40.65	3.74(×1.39)	68.87	2.89(×1.41)	52.89	2.33(×1.41)	50.33	1.53(×1.41)	53.19	2.62(×1.40)
EdgeFormer	38.25	**2.84(×1.83)**	67.82	**2.29(×1.78)**	52.17	**1.86(×1.76)**	50.16	**1.23(×1.75)**	52.10	**2.06(×1.79)**
OPT-6.7B										
HFAccelerate	62.65	13.61(×1.00)	75.06	11.05(×1.00)	74.55	8.25(×1.00)	73.08	6.04(×1.00)	71.34	9.74(×1.00)
DeepSpeed	61.97	12.58(×1.08)	75.17	10.17(×1.09)	75.48	7.43(×1.11)	73.90	5.36(×1.13)	71.63	8.89(×1.10)
Galaxy	61.56	9.83(×1.38)	74.59	7.94(×1.39)	74.87	5.83(×1.42)	72.57	4.19(×1.44)	70.90	6.94(×1.40)
EdgeFormer	61.24	**6.97(×1.95)**	73.04	**5.67(×1.95)**	72.51	**4.20(×1.96)**	71.09	**3.06(×1.97)**	69.47	**4.98(×1.96)**
OPT-13B (FP16)										
HFAccelerate	64.54	14.26(×1.00)	77.64	11.68(×1.00)	78.42	8.69(×1.00)	75.93	6.45(×1.00)	74.13	10.27(×1.00)
DeepSpeed	64.19	12.87(×1.11)	77.16	10.55(×1.11)	77.81	7.93(×1.09)	75.63	5.81(×1.11)	73.70	9.29(×1.11)
Galaxy	64.05	10.05(×1.42)	76.87	8.24(×1.41)	76.94	6.19(×1.40)	74.38	4.54(×1.42)	73.06	7.26(×1.41)
EdgeFormer	63.24	**7.09(×2.01)**	75.42	**5.91(×1.98)**	75.31	**4.42(×1.97)**	73.34	**3.38(×1.91)**	71.83	**5.20(×1.98)**

proportionally with model size across all devices. For instance, in the OPT-13B (FP16) configuration, Device-1 allocates 9.7 GB solely for LLM parameters, while Devices 2 and 3 hold 9.7 GB and 9.8 GB, respectively. In contrast, the overhead from MT remains minimal ($\leq$ 0.5 GB per device even for large models), highlighting the efficiency of EdgeFormer's coordination mechanism. OS memory usage is consistent across devices, typically remaining at 1.3 GB. Notably, these trends persist across all evaluated LLMs, indicating that EdgeFormer introduces only minor coordination overhead atop the standard memory footprint, thereby enabling efficient deployment of large-scale models in distributed edge environments without introducing significant memory bottlenecks.

4.5 System Parameter Analysis

Varying Edge Conditions[3]. We evaluate the performance of EdgeFormer under varying edge conditions, including bandwidths (Fig. 7) and device capacities (Fig. 8), respectively. Across all datasets (each 1500 samples), it can be observed that increasing bandwidth sequences (e.g., 200→400→600 Mbps, at the 500- and 1000-sample, respectively) generally yield lower latency and slightly better accuracy compared to decreasing or unstable bandwidth scenarios (e.g., 600→400→200 or 400→600→200), indicating that EdgeFormer benefits from early-stage communication efficiency. In terms of device heterogeneity, upgrading Device 1 (↑ at the 750-sample) consistently improves latency and accuracy, whereas downgrading Device 1 (↓ at the 750-sample) degrades both metrics, showing EdgeFormer's sensitivity to the slowest device in the system. Interestingly, mixed-capacity settings (Device 1 ↑ & Device 2 ↓) maintain balanced performance, suggesting that EdgeFormer's dynamic head scheduling mitigates the impact of weaker devices through adaptive computation distribution. These results highlight EdgeFormer's robustness and adaptability in diverse edge environments.

We detail the remaining experiment in Appendix D.1, due to space limit.

4.6 Ablation Study

We perform ablation studies with three variants:

- **EdgeFormer-Full**: Incorporates both Head Pruning (HP) and Dynamic Scheduling (DS) as defined in Eq. 15.
- **EdgeFormer-w/o HP**: Disables head pruning by retaining all heads; only DS is applied.

[3]Due to space limit, we only present experimental results on OPT-1.3B here, ***putting results on other LLMs in our source code***, similarly for the following analysis on method parameters, latency breakdown, memory overhead, etc.

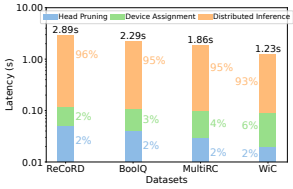


Figure 5: Latency Breakdown.

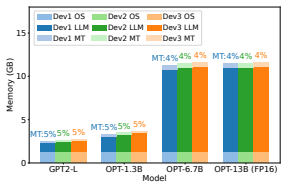


Figure 6: Memory Overhead.

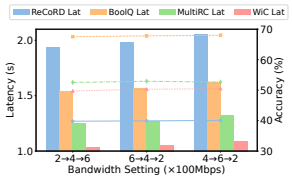


Figure 7: Varying Bandwidths.

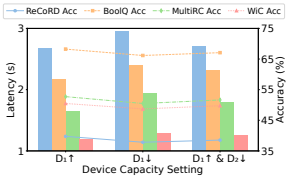


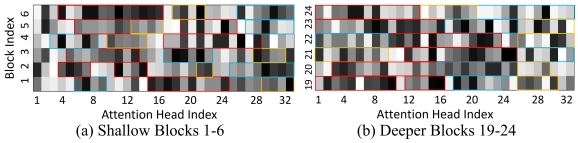
Figure 8: Varying Capacities.

- **EdgeFormer-w/o DS**: Disables dynamic scheduling by statically allocating heads; only HP is applied.

Table 3 presents an ablation study on EdgeFormer using OPT-1.3B, evaluating the individual contributions of Head Pruning (HP) and Dynamic Scheduling (DS). The full model, incorporating both components, achieves the lowest latency across all datasets, with 2.8s on ReCoRD, offering a 3.4× speedup, while maintaining competitive accuracy. Removing HP (w/o HP) increases latency to 3.8s on ReCoRD, as it computes over all heads, although accuracy slightly improves due to the preservation of semantic capacity. In contrast, excluding DS (w/o DS) leads to higher latency of 4.4s on ReCoRD, as static head allocation fails to exploit resource availability, reducing parallelism. Finally, the vanilla with both components removed (w/o both) shows the highest latency (9.5s on ReCoRD), confirming the necessity of both HP and DS for optimal performance. These results highlight that HP reduces redundant computation, while DS enables efficient parallelism, achieving the optimal balance between latency and accuracy in dynamic edge environments.

Table 3: Ablation study of EdgeFormer (OPT-1.3B).

Variants	ReCoRD		BoolQ		MultiRC		WiC	
	Lat	Acc	Lat	Acc	Lat	Acc	Lat	Acc
Full	**2.8**	38.2	**2.3**	67.8	**1.9**	52.2	**1.2**	50.2
w/o HP	3.8	39.8	3.5	68.9	2.6	53.7	1.8	52.1
w/o DS	4.4	38.8	4.4	67.7	3.1	52.7	2.1	50.9
w/o both	9.5	40.3	7.4	69.0	5.2	53.5	3.5	52.2



(a) Shallow Blocks 1-6 (b) Deeper Blocks 19-24

Figure 9: Visualization of pruned and allocated attention heads across different blocks in heterogeneous device environments, where heads are allocated to 20 TFLOPS device (***Red Boxes***), 13 TFLOPS device (***Blue Boxes***), and 7 TFLOPS device (***Yellow Boxes***), respectively (LLM: OPT-1.3B).

4.7 Visualizing Head Pruning & Allocation

To clarify the impact of EdgeFormer's pruning and allocation strategies, we visualize the distribution of pruned and retained attention heads across layers and edge devices in heterogeneous environments, as shown in Fig. 9. *On one hand*, we can see the layer-specific pruning patterns. Specifically, in shallow layers (Blocks 1-6), selective pruning removes 32% of heads (w/o any boxes in Fig. 9 (a)), primarily those with low attention weight variance (e.g., variance <0.1), propably due to that shallow layers focus on basic token correlations (e.g., syntax), where redundancy is prevalent. In deeper layers (Blocks 19-24), over 80% of heads are retained (Fig. 9 (b)), as they encode task-specific semantics (e.g., contextual reasoning). *On the other hand*, we find that heads are dynamically assigned to heterogeneous devices based on computational capabilities, verifying the effectiveness of the cross-device allocation strategy. Specifically, 20 TFLOPS device processes about 70% of retained heads in deeper layers, leveraging its high memory and compute power, 13 TFLOPS device handles 20% of heads, balancing workload and resource constraints, and 7 TFLOPS device manages 10% of heads, focusing on lightweight computations.

We detail *Method Parameter Analysis*, *Case Study* in Appendix D.2, D.3 due to space limits.

5 Conclusion

This paper proposes EdgeFormer, a latency-aware framework for efficient and privacy-preserving Transformer inference in dynamic edge networks. Through dynamic block allocation and collaborative multi-head attention guided by LiScore. EdgeFormer significantly reduces inference latency across diverse LLMs with negligible accuracy loss. This work lays the groundwork for resource-efficient, latency-critical LLM deployment at the edge, with future extensions toward federated learning for enhanced privacy.

Limitations

The proposed latency-aware framework has some limitations. First, it relies heavily on high-speed communication among edge devices, which may become a bottleneck in practical deployments with constrained network bandwidth. Second, the framework assumes trusted edge devices for collaborative inference, which may limit its applicability in open or untrusted environments. Third, dynamic allocation and coordination among edge devices introduce additional overhead, particularly in highly volatile networks with frequent topology or resource changes.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Fenglong Cai, Dong Yuan, Mengwei Xie, Wei He, Lanju Kong, Wei Guo, Yali Jiang, and Lizhen Cui. 2023. Paratra: A parallel transformer inference framework for concurrent service provision in edge computing. In *2023 IEEE International Conference on Web Services (ICWS)*, pages 258–268.

Huaiguang Cai, Zhi Zhou, and Qianyi Huang. 2024. Online resource allocation for edge intelligence with colocated model retraining and inference. In *IEEE Conference on Computer Communications*, pages 1900–1909. IEEE.

Quan Chen, Song Guo, Kaijia Wang, Wenchao Xu, Jing Li, Zhipeng Cai, Hong Gao, and Albert Y. Zomaya. 2024a. Towards real-time inference offloading with distributed edge computing: The framework and algorithms. *IEEE Transactions on Mobile Computing*, 23(7):7552–7571.

Yanming Chen, Tong Luo, and Weiwei Fang. 2024b. Edgeci: Distributed workload assignment and model partitioning for cnn inference on edge clusters. *ACM Transactions on Internet Technology*, 24(2):1–24.

Badhan Chandra Das, M Hadi Amini, and Yanzhao Wu. 2025. Security and privacy challenges of large language models: A survey. *ACM Computing Surveys*, 57(6):1–39.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16091–16101.

Fateneh Golpayegani, Nanxi Chen, Nima Afraz, Eric Gyamfi, Abdollah Malekjafarian, Dominik Schäfer, and Christian Krupitzer. 2024. Adaptation in edge computing: a review on design principles and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 19(3):1–43.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable.

Daya Guo, Dejian Yang, Haowei Zhang, and Junxiao Song. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Xiaotian Guo, Quan Jiang, and Yixian Shen. 2024. Easter: Learning to split transformers at the edge robustly. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):3626–3637.

Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 2021. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4270–4284.

Xueyu Hou, Yongjie Guan, Tao Han, and Ning Zhang. 2022. Distredge: Speeding up convolutional neural network inference on distributed edge devices. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1097–1107.

Chenghao Hu and Baochun Li. 2024. When the edge meets transformers: Distributed inference with transformer models. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pages 82–92.

Hao Kong, Di Liu, and Shuo Huai. 2023. Edgecompress: Coupling multidimensional model compression and dynamic inference for edgeai. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(12):4657–4670.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629.

Malgorzata Lazuka, Andreea Anghel, and Thomas Parnell. 2024. Llm-pilot: Characterize and optimize performance of your llm inference services. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–18. IEEE.

Juhyeon Lee, Insung Bahk, Hoseung Kim, Sinjin Jeong, Suyeon Lee, and Donghyun Min. 2024. An autonomous parallelization of transformer model inference on heterogeneous edge devices. In *Proceedings of the 38th ACM International Conference on Supercomputing*, ICS '24, page 50–61, New York, NY, USA. Association for Computing Machinery.

Hui Li, Xiuhua Li, Qilin Fan, Qiang He, Xiaofei Wang, and Victor CM Leung. 2024. Distributed dnn inference with fine-grained model partitioning in mobile edge computing networks. *IEEE Transactions on Mobile Computing*, 23(10):9060–9074.

Jing Li, Weifa Liang, Yuchen Li, Zichuan Xu, Xiaohua Jia, and Song Guo. 2021. Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism. *IEEE Transactions on Mobile Computing*, 22(5):3017–3030.

Nan Li, Alexandros Iosifidis, and Qi Zhang. 2022. Collaborative edge computing for distributed cnn inference acceleration using receptive field-based segmentation. *Computer Networks*, 214:109150.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538.

Zhiyan Liu, Qiao Lan, and Kaibin Huang. 2023a. Resource allocation for multiuser edge inference with batching and early exiting. *IEEE Journal on Selected Areas in Communications*, 41(4):1186–1200.

Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. 2023b. Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13.

May Malka, Erez Farhan, Hai Morgenstern, and Nir Shlezinger. 2024. Decentralized low-latency collaborative inference via ensembles on the edge. *IEEE Transactions on Wireless Communications*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Xudong Pan, Mi Zhang, Shouling Ji, and Min Yang. 2020. Privacy risks of general-purpose language models. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1314–1331. IEEE.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *26th ACM International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 3505–3506.

Reent Schlegel, Siddhartha Kumar, and Eirik Rosnes. 2022. Privacy-preserving coded mobile edge computing for low-latency distributed inference. *IEEE Journal on Selected Areas in Communications*, 40(3):788–799.

Hugo Touvron, Louis Martin, and Kevin Stone. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808. ACL Anthology.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. 2020. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*.

Li Wang, Liang Li, Lianming Xu, Xian Peng, and Aiguo Fei. 2024. Failure-resilient distributed inference with model compression over heterogeneous edge devices. *IEEE Transactions on Mobile Computing*.

Yuanxin Wei, Shengyuan Ye, Jiazhi Jiang, Xu Chen, Dan Huang, Jiangsu Du, and Yutong Lu. 2024. Communication-efficient model parallelism for distributed in-situ transformer inference. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1513–1528.

Zichuan Xu, Liqian Zhao, Weifa Liang, Omer F. Rana, Pan Zhou, Qiufen Xia, Wenzheng Xu, and Guowei Wu. 2021. Energy-aware inference offloading for dnn-driven applications in mobile edge clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(4):799–814.

Shusen Yang, Zhanhua Zhang, Cong Zhao, Xin Song, Siyan Guo, and Hailiang Li. 2022. Cnnpc: End-edge-cloud collaborative cnn inference with joint model partition and compression. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4039–4056.

Shengyuan Ye, Jiangsu Du, and Liekang Zeng. 2024. Galaxy: A resource-efficient collaborative edge ai system for in-situ transformer inference. In *IEEE Conference on Computer Communications*, pages 1001–1010.

Jeffrey Yu, Kartik Prabhu, Yonatan Urman, Robert M. Radway, Eric Han, and Priyanka Raina. 2024. 8-bit transformer inference and fine-tuning for edge accelerators. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, page 5–21, New York, NY, USA.

Liekang Zeng and Xu Chen. 2020. Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Transactions on Networking*, 29(2):595–608.

A Appendix for Related Work

A.1 Distributed Edge Inference

Recent advances in distributed edge inference aim to improve efficiency under resource constraints through optimized task offloading, model partitioning, and resource allocation. Early works (Li et al., 2021; Hou et al., 2022) proposed static partitioning strategies for DNN/CNN models, allocating layers to edge devices based on predefined computational capabilities. Frameworks like (Xu et al., 2021) introduced dynamic offloading to balance latency and energy consumption, while (Yang et al., 2022; Kong et al., 2023) explored model compression to reduce communication overhead. However, these approaches primarily target conventional architectures (e.g., CNNs), treating models as monolithic units and overlooking the fine-grained parallelism inherent in Transformers. More recent studies (Cai et al., 2023; Guo et al., 2024) have considered Transformer block allocation but neglect the dynamic behavior of multi-head attention (MHA) and fluctuating edge conditions. Approaches like (Ye et al., 2024; Lee et al., 2024) attempt MHA partitioning but rely on rigid head assignments, ignoring semantic heterogeneity and real-time network variability and leading to suboptimal latency and limited adaptability. In contrast, Edgeformer integrates dynamic block allocation via efficiency-storage prioritization and latency-aware collaborative MHA, enabling fine-grained, adaptive parallelism under real-time resource constraints.

A.2 Transformer Model Pruning

Pruning has become a key strategy for reducing the computational cost of Transformer models while maintaining accuracy. Early studies (Michel et al., 2019; Voita et al., 2019) applied structured pruning of entire attention heads or feed-forward layers, leveraging magnitude or gradient-based importance metrics. For instance, (Wang et al., 2020) removed heads with low attention entropy, while (Lin et al., 2020) used reinforcement learning to identify redundancy. More recent efforts (Lagunas et al., 2021) introduced dynamic pruning that adapts sparsity to input complexity. However, these methods primarily target cloud-based scenarios, assuming abundant computational resources and ignoring latency constraints in edge environments. Furthermore, they lack a unified framework that integrates pruning with distributed computation, often prioritizing compression over real-time efficiency. In

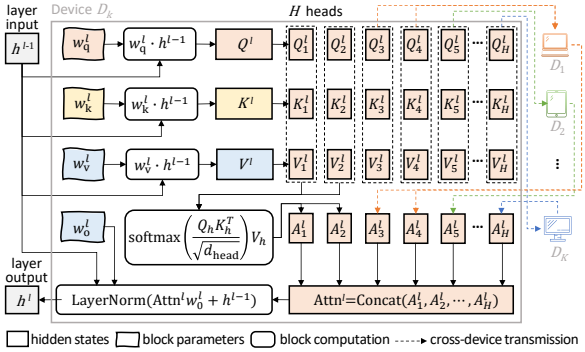


Figure 10: Workflow of the cMHA inference for block l in device D_k.

contrast, this work introduces LiScore, a composite metric that jointly optimizes semantic importance and execution latency, while integrating pruning with collaborative offloading to achieve both efficiency and accuracy in dynamic edge settings.

B Appendix for System Model

B.1 Collaborative Attention

As shown in Fig 10, for block l assigned to device D_k, the collaborative inference for multi-head attention includes:

1) Local Q/K/V Projection: D_k performs Q/K/V projections locally and then splits them into H heads as follows,

$$Q/K/V^l = W^l_{\text{q/k/v}} \cdot h^{l-1}, \tag{18}$$

$$\{Q_h, K_h, V_h\}_{h=1}^{H} = \text{SplitHeads}(Q/K/V^l). \tag{19}$$

2) Cross-Device MHA Computation: D_k chooses the Q/K/V of multiple heads $\mathcal{H}_{k'}$ to send to a neighbor $D_{k'}$ (detailed in Section 3). For each head $h_{k'} \in \mathcal{H}_{k'}$, $D_{k'}$ computes:

$$\text{Attn}^l_{h_{k'}} = \text{Softmax}\left(\frac{Q_{h_{k'}} K^\top_{h_{k'}}}{\sqrt{d_k}}\right) V_{h_{k'}}. \tag{20}$$

After computation, the device $D_{k'}$ will send back $\text{Attn}^l_{h_{k'}}$ to D_k.

3) Local Post-Processing: When D_k receives all $\text{Attn}^l_{h_{k'}}$ of the heads it has sent out to neighbor devices for cross-device attention computation, it will perform concatenation, along with the output projection and Add&Normalize operations:

$$\text{Attn}^l = \text{Concat}(\text{Attn}^l_1, ..., \text{Attn}^l_H), \tag{21}$$

$$h^l = \text{LayerNorm}(\text{Attn}^l W^l_o + h^{l-1}). \tag{22}$$

Blocks are strictly processed sequentially from $l = 1$ to L.

C Appendix for Method

C.1 Head Importance Update

For a head b' that fails to match any existing cluster within threshold τ, we recalculate its importance score $I_{b'}$ according to Eqs. 12,13, and update $I_{b'}$ into the mapping table $\mathcal{M}$:

- Compute the similarity of b' to all existing cluster centroids

$$\text{Sim}(b', C_j) = \left\| \mathbf{f}^{\text{sep}}_{b'} - \mathbf{f}^{\text{sep}}_{\mathbf{c}_j} \right\|_2 + \zeta \left(1 - \cos\left(\mathbf{f}^{\text{inter}}_{b'}, \mathbf{f}^{\text{inter}}_{\mathbf{c}_j}\right)\right).$$

- If there is j that satisfies $\text{Sim}(b', C_j) \leq \tau$, assign b' to C_j with the smallest $\text{Sim}(b', C_j)$ and update the mapping table

$$\mathcal{M} \leftarrow \mathcal{M} \cup \left\{ \left(C_j, \left\{\mathbf{f}^{\text{sep}}_{b'}, \mathbf{f}^{\text{inter}}_{b'}\right\}, I_{b'}\right) \right\}.$$

- Else, we create a new cluster C_{new} with b' as the centroid, and update the mapping table

$$\mathcal{M} \leftarrow \mathcal{M} \cup \left\{ \left(C_{\text{new}}, \left\{\mathbf{f}^{\text{sep}}_{b'}, \mathbf{f}^{\text{inter}}_{b'}\right\}, I_{b'}\right) \right\}.$$

In addition, we perform full reclustering (as the above head-conditioned clustering) and reconstruct the mapping table during idle periods, to maintain cluster quality and adapt to evolving head patterns. This two-fold update mechanism ensures robust adaptation to both transient head variations and long-term pattern shifts, critical for sustaining high-performance collaborative inference in dynamic edge environments.

C.2 Dynamic Head Allocation

This section formulates the allocation problem, followed by a heuristic solution.

To minimize end-to-end latency while adhering to inference accuracy, we formulate head retention and device assignment as a unified optimization problem that dynamically allocates attention heads to devices based on their semantic importance, execution latency, and real-time resource availability.

To maximize the total composite importance of retained heads, we formulate the dynamic head allocation for MHA of block l in device k as an

optimization problem:

$$\begin{aligned} \max \quad & \textstyle\sum_{b=1}^{H} \mathcal{S}_b \cdot \mathbb{I}(x_b \neq 0), \qquad (23) \\ \text{s.t.} \quad \text{C1}: \; & \textstyle\sum_{b=1}^{H} \mathbb{I}(x_b = k') \leq s'_{k'}, \\ & \forall k' \in \{1, 2, ..., K\} \\ \text{C2}: \; & T_{\text{collab-att}} \leq T_{\max} \end{aligned}$$

Here, C1 ensures that the allocated heads are within the left storage capacity $s'_{k'}$ (measured by the maximum number of heads $D_{k'}$ can store) of collaborator device k', and C2 limits the maximum tolerable latency of collaborative attention.

C.3 Heuristic Algorithm

Given the NP-hard nature of the problem, we propose a greedy heuristic algorithm to efficiently allocate attention heads while balancing semantic importance and latency constraints.

- Head Prioritization. Estimate I_b for each head b; sort heads in descending order of I_b.
- Feasible Assignment. For each b, choose devices with sufficient storage ($s_{k'} > 1$); assign b to k' that minimizes $T(b)$.
- Latency-Bounded Pruning. After allocation, if C2 is violated, iteratively prune the lowest $\mathcal{S}_B$ head until it is satisfied.

Algorithm 1 details dynamic allocation, retaining $O(H\log H + H \cdot K)$ time complexity as each head evaluates K devices once.

C.4 Theorem

Theorem 1. *For any non-zero cluster centroid c, the hierarchical clustering process uniquely determines the relationship between a head b and c. If two distinct head b and b' satisfy:*

$$\begin{cases} \|f_b^{\text{sep}} - f_c^{\text{sep}}\|_2 = \|f_{b'}^{\text{sep}} - f_c^{\text{sep}}\|_2 & (\textit{Fold 1}) \\ \cos(f_b^{\text{inter}}, f_c^{\text{inter}}) = \cos(f_{b'}^{\text{inter}}, f_c^{\text{inter}}) & (\textit{Fold 2}) \end{cases}$$

then $b = b'$, provided $c \neq 0$.

Proof. Clustering via Euclidean distance on separable features:

$$\|f_b^{\text{sep}} - f_c^{\text{sep}}\|_2^2 = \|f_{b'}^{\text{sep}} - f_c^{\text{sep}}\|_2^2. \qquad \text{(Fold 1)}$$

Simplifying yields:

$$\|f_b^{\text{sep}}\|_2^2 - 2f_b^{\text{sep}} \cdot f_c^{\text{sep}} = \|f_{b'}^{\text{sep}}\|_2^2 - 2f_{b'}^{\text{sep}} \cdot f_c^{\text{sep}}.$$

Algorithm 1 Heuristic Head Allocation & Pruning

Require: Heads $\mathcal{H}$, Devices $\mathcal{D}$, Storage $s'_{k'}$, Max latency $T_{\max}$
Ensure: Assignment $x_b \in \{0, 1, \ldots, K\}$
Stage 1: Prioritize Heads
Compute I_b for all $b \in \mathcal{H}$ (Eq. 15).
Sort $\mathcal{H}$ as $\mathcal{B}$ (descending I_b).
Stage 2: Assign Heads to Devices
Initialize $s'_{k'} \leftarrow$ remaining storage, $T_{\text{collab-att}} \leftarrow 0$.
For each $b \in \mathcal{B}$:
  Generate candidates $C = \{k \cup \mathcal{D}_{\text{neigh}}\}$.
  For $k'' \in C$:
    Compute $\Delta T = \begin{cases} T_{\text{local-att}} & \text{if } k'' = k \\ T_{\text{remote-att}}(k'') & \text{else} \end{cases}$.
  Rank C by ΔT (ascending).
  For ranked k'':
    If $s'_{k''} \geq 1$ and $T_{\text{collab-att}} + \Delta T \leq T_{\max}$:
      Assign $x_b \leftarrow k''$, decrement $s'_{k''}$.
      $T_{\text{collab-att}} \leftarrow \max(T_{\text{local-att}}, \max_{k'} T_{\text{remote-att}}(k'))$.
      Break.
  If no valid k'': Prune $x_b \leftarrow 0$.
Stage 3: Enforce Latency Constraint
If $T_{\text{collab-att}} > T_{\max}$:
  Sort assigned heads by ascending $\mathcal{S}_b$.
  While $T_{\text{collab-att}} > T_{\max}$:
    Prune lowest-score head b', set $x_{b'} \leftarrow 0$.
    Recompute $T_{\text{collab-att}}$.
Return assignments $\{x_b\}$.

Within a cluster, use cosine similarity on interaction features:

$$\frac{f_b^{\text{inter}} \cdot f_c^{\text{inter}}}{\|f_b^{\text{inter}}\|_2 \|f_c^{\text{inter}}\|_2} = \frac{f_{b'}^{\text{inter}} \cdot f_c^{\text{inter}}}{\|f_{b'}^{\text{inter}}\|_2 \|f_c^{\text{inter}}\|_2}. \qquad \text{(Fold 2)}$$

Let $\alpha = \|f_b^{\text{inter}}\|_2$, $\beta = \|f_{b'}^{\text{inter}}\|_2$, then $f_{b'}^{\text{inter}} \cdot f_c^{\text{inter}} = \frac{\beta}{\alpha}(f_b^{\text{inter}} \cdot f_c^{\text{inter}})$.

Substituting Fold 2 results into Fold 1, we derive $f_b^{\text{sep}} = f_{b'}^{\text{sep}}$ and $f_b^{\text{inter}} = f_{b'}^{\text{inter}}$.

Since f^{sep} and f^{inter} are jointly injective, b and b' must be identical. The hierarchical design ensures uniqueness: 1) Fold 1 isolates separable patterns. 2) Fold 2 distinguishes interaction patterns within clusters.

Thus, $b = b'$ if both Fold 1 and Fold 2 constraints hold. □

Theorem 2. *The optimization problem in Eq. 23 is NP-hard.*

Proof. We prove the NP-hard of Eq. 23 by reducing it into a multiple knap problem (MKP) as follows:

- Mapping Devices to Knapsacks. Each device $D_{k'}$ could be seen as a knapsack with a capacity $s_{k'}$.

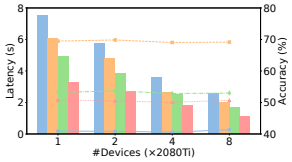


Figure 11: Device Counts.

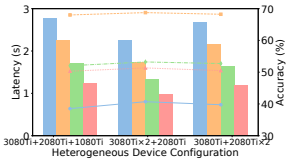


Figure 12: Heterogeneous Devices.

- Mapping Heads to Items. Each attention head b corresponds to an item with the weight $w_b = 1$ representing one unit of storage per head and value $v_b = \mathcal{S}_b$ denoting the composite score in Eq. 15.

- Constraint Simplification. By relaxing C2 (setting $T_{\max} \to \infty$, the problem reduces into MKP, where the goal is to maximize total value without exceeding knapsack capacities.

MKP is a well-known NP-hard problem. Since the relaxation of Eq. 23 reduces to MKP, the original problem is **at least NP-hard**. Adding the latency constraint $C2$ further increases complexity, as verifying $T_{\text{collab-att}} \leq T_{\max}$ entails solving a potentially NP-hard subproblem (e.g., nonlinear latency aggregation).

Thus, unless $P = NP$, Eq. 23 admits no polynomial-time solution, requiring heuristics or approximation methods. □

D Appendix for Experiment

D.1 System Parameter Analysis

Diverse Device Counts. We evaluate the impact of device count (13 TFLOPS) on EdgeFormer's performance using OPT-1.3B across four datasets in Fig. 11, with a single-device setup as the baseline. As the number of devices increases, latency consistently decreases. For example, on ReCoRD, latency drops from 7.53s with one device to 2.56s with eight devices, achieving a 2.94× speedup. Similarly, on BoolQ, latency improves from 6.07s to 1.99s (3.05× speedup), on MultiRC from 4.93s to 1.66s (2.97× speedup), and on WiC from 3.26s to 1.11s (2.93× speedup). These results highlight EdgeFormer's effective parallelization of computations across devices, reducing latency while maintaining stable accuracy. However, as the number of devices grows, the rate of improvement diminishes due to increased communication overhead. Despite this, accuracy remains stable across datasets, with a maximum drop of only 0.83%. This confirms that EdgeFormer's dynamic scheduling and collaborative execution preserve model performance while improving efficiency in edge environments.

Heterogeneous Devices. We further evaluate EdgeFormer's performance under heterogeneous device configurations in Fig. 12. Among the three configurations, the system with two powerful devices (20 TFLOPS ×2) and one weaker device (13 TFLOPS) achieves the best latency across all datasets, emphasizing the benefit of using more capable devices to handle the majority of the workload. Latency improves across all tasks, with the largest reduction observed on BoolQ (from 2.26s to 1.73s). In contrast, the configuration with one strong and two weaker devices results in slightly higher latency, especially for compute-heavy tasks like MultiRC, where bottlenecks from the less capable devices become more evident. Despite these latency differences, accuracy remains nearly unchanged across all configurations, with fluctuations of only $\pm 0.85\%$, confirming that EdgeFormer's dynamic scheduling can effectively manage heterogeneous device environments without compromising model performance.

D.2 Method Parameter Analysis

Parameter Analysis for λ. Fig. 13 shows the effect of varying the trade-off parameter λ in Eq. 15, which balances semantic importance and execution latency in EdgeFormer. As λ increases from 0.2 to 0.8, accuracy improves across all datasets: ReCoRD increases from 34.5% to 38.3%, BoolQ from 56.9% to 67.8%, MultiRC from 46.8% to 52.2%, and WiC from 47.8% to 50.2%. This indicates that incorporating more semantic importance into the head allocation process yields moderate performance gains. However, when λ increases from 0.8 to 1.0, accuracy gains begin to plateau, suggesting diminishing returns. Throughout all λ values, latency remains relatively stable, around 2.8s on ReCoRD, 2.2s on BoolQ, 1.8s on MultiRC, and 1.2s on WiC. Thus, a moderate $\lambda = 0.8$ strikes the best balance, delivering near-maximum accuracy while maintaining acceptable latency.

Parameter Analysis for ζ. Fig. 14 shows the effect of the threshold parameter ζ in Eq. 17, which weights interaction importance of training samples. As ζ increases from 0.2 to 0.6, accuracy improves across all datasets: ReCoRD from 39.1% to 39.8%, BoolQ from 67.1% to 67.8%, MultiRC from 51.5% to 52.2%, and WiC from 49.3% to 50.2%. This suggests that moderate ζ values enhance the inclusion of informative heads. However, further increasing ζ to 0.8 and 1.0 leads to accuracy drops, especially on ReCoRD (38.3%) and WiC (48.1%), likely due

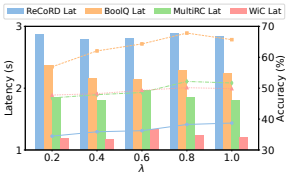


Figure 13: Analysis on λ.

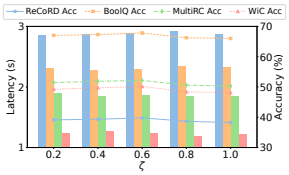


Figure 14: Analysis on ζ.

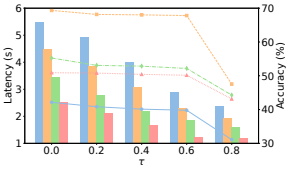


Figure 15: Analysis on τ.

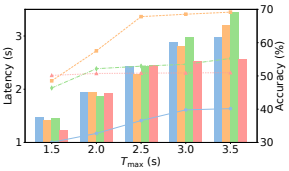


Figure 16: Analysis on $T_{\max}$.

to the inclusion of less relevant heads. Latency remains stable across ζ values, with minor variations: around 2.9s for ReCoRD, 2.3s for BoolQ, 1.85s for MultiRC, and 1.2s for WiC. These indicate that $\zeta = 0.6$ achieves the best balance between accuracy and latency.

Parameter Analysis for τ. Fig. 15 shows the impact of the threshold τ, which governs the reuse of importance scores between similar Transformer blocks. As τ increases from 0.0 to 0.8, latency decreases across all datasets, indicating that reusing importance scores for similar blocks effectively reduces computational overhead. For example, on the ReCoRD dataset, latency drops from 5.48s at $\tau = 0.0$ to 2.37s at $\tau = 0.8$. Similar reductions are observed on BoolQ, MultiRC, and WiC. However, this latency reduction comes at the cost of accuracy. On ReCoRD, for example, accuracy falls from 42.2% at $\tau = 0.0$ to 31.1% at $\tau = 0.8$, with similar declines on other datasets. This suggests that while reusing importance scores improves efficiency, it may reduce accuracy as the threshold for reuse becomes more lenient. Overall, $\tau = 0.6$ offers the best trade-off, balancing low latency with acceptable accuracy.

Parameter Analysis for $T_{\max}$. In Fig. 16, we analynize the impact of the collaborative attention latency thresholds $T_{\max}$ (Eq. 23) on system performance. As $T_{\max}$ increases from 1.5s to 3.5s, latency steadily grows across all datasets, reflecting the system's allowance for more remote attention operations. For instance, on ReCoRD, latency rises from 1.47s at $T_{\max} = 1.5$s to 2.97s at $T_{\max} = 3.5$s. Similar trends are observed for BoolQ, MultiRC, and WiC. Simultaneously, accuracy improves with higher $T_{\max}$, especially on ReCoRD (30.1% $\rightarrow$ 40.2%), BoolQ (48.5% $\rightarrow$ 69.2%), and MultiRC (46.4% $\rightarrow$ 55.3%). Accuracy stabilizes around $T_{\max} = 3.0$s, suggesting that relaxing the latency constraint allows EdgeFormer to better leverage high-quality remote attention heads, boosting prediction quality without significantly impacting inference delay.

D.3 Case Study

We present a case study for a representative query in Table 4, showing the impact scores, execution device of each attention head, current GPU usage rate of the device and real-time bandwidth.

First, we can observe a clear layer-wise trend in head pruning. Specifically, layers 4, 8, 12, 16, and 20 exhibit progressively fewer pruned heads as the layer depth increases. This behavior is consistent with the intuition that deeper layers tend to preserve more semantically rich and task-critical information, making aggressive head pruning less favorable in later stages of the model.

Second, It can be observed that the allocation of attention heads across devices is strongly influenced by the available network bandwidth. When the bandwidth is sufficient, attention heads with higher impact scores are preferentially offloaded to devices with lower GPU utilization and higher computing capacity (Device 1 having the highest capability and Device 3 the lowest). In these layers with sufficient bandwidth (e.g., layers 4, 20), Device 1 and Device 2 execute a larger number of high-impact heads while maintaining relatively lower GPU usage rate, whereas Device 3 handles fewer heads. In contrast, under bandwidth-constrained settings (e.g., layer 8), more attention heads are computed locally on the device hosting the current submodel (The bold numbers indicate the devices where the submodels are located), while cross-device head transfers are reduced.

Third, we observe that GPU utilization further modulates head allocation when bandwidth is abundant. In layers where the current device exhibits high GPU usage (e.g., layer 16), the framework proactively redistributes more attention heads to other devices with lower utilization, effectively balancing the computational load.

Table 4: Layer-wise head pruning and cross-device execution analysis. (IS: Impact Score, ED: Execution Device, GU: GPU Usage (%), RB: Real-time Bandwidth (Mbps))

Head	Layer4				Layer8				Layer12				Layer16				Layer20			
	IS	ED	DC	RB	IS	ED	DC	RB	IS	ED	DC	RB	IS	ED	DC	RB	IS	ED	DC	RB
1	0.639	**1**	76%	774	0.156	3	68%	314	0.367	**2**	86%	556	0.932	**2**	82%	481	0.335	1	47%	924
2	0.301	2	46%	–	0.809	**1**	74%	–	0.259	3	58%	–	0.789	**2**	82%	–	0.928	1	47%	–
3	0.142	–	–	–	0.432	**1**	74%	–	0.499	**2**	86%	–	0.150	–	–	–	0.558	1	47%	–
4	0.058	–	–	–	0.121	3	68%	–	0.897	**2**	86%	–	0.748	**2**	82%	–	0.461	1	47%	–
5	0.662	**1**	76%	–	0.197	2	51%	–	0.425	**2**	86%	–	0.287	–	–	–	0.926	1	47%	–
6	0.374	2	46%	–	0.511	**1**	74%	–	0.827	**2**	86%	–	0.225	3	62%	–	0.208	2	65%	–
7	0.109	–	–	–	0.402	**1**	74%	–	0.265	1	42%	–	0.126	–	–	–	0.489	1	47%	–
8	0.225	2	46%	–	0.537	**1**	74%	–	0.509	**2**	86%	–	0.045	–	–	–	0.242	2	65%	–
9	0.128	–	–	–	0.475	**1**	74%	–	0.297	1	42%	–	0.583	**2**	82%	–	0.766	1	47%	–
10	0.967	**1**	76%	–	0.291	**1**	74%	–	0.519	**2**	86%	–	0.663	**2**	82%	–	0.995	1	47%	–
11	0.748	**1**	76%	–	0.275	**1**	74%	–	0.254	1	42%	–	0.457	1	49%	–	0.803	1	47%	–
12	0.635	**1**	76%	–	0.155	2	51%	–	0.455	**2**	86%	–	0.801	**2**	82%	–	0.261	2	65%	–
13	0.631	**1**	76%	–	0.851	**1**	74%	–	0.764	**2**	86%	–	0.525	**2**	82%	–	0.693	1	47%	–
14	0.584	**1**	76%	–	0.276	**1**	74%	–	0.972	**2**	86%	–	0.064	–	–	–	0.946	1	47%	–
15	0.408	**1**	76%	–	0.397	**1**	74%	–	0.359	1	42%	–	0.594	**2**	82%	–	0.147	**3**	87%	–
16	0.366	2	46%	–	0.768	**1**	74%	–	0.256	1	42%	–	0.835	**2**	82%	–	0.256	2	65%	–
17	0.426	2	46%	–	0.344	**1**	74%	–	0.206	–	–	–	0.445	1	49%	–	0.117	**3**	87%	–
18	0.102	–	–	–	0.174	2	51%	–	0.993	**2**	86%	–	0.575	**2**	82%	–	0.353	2	65%	–
19	0.141	–	–	–	0.031	–	–	–	0.352	**2**	86%	–	0.941	**2**	82%	–	0.984	1	47%	–
20	0.549	**1**	76%	–	0.039	–	–	–	0.499	**2**	86%	–	0.486	1	49%	–	0.141	**3**	87%	–
21	0.529	**1**	76%	–	0.621	**1**	74%	–	0.659	**2**	86%	–	0.415	1	49%	–	0.081	–	–	–
22	0.779	**1**	76%	–	0.641	**1**	74%	–	0.229	3	58%	–	0.377	1	49%	–	0.322	2	65%	–
23	0.601	**1**	76%	–	0.026	–	–	–	0.299	1	42%	–	0.115	3	62%	–	0.812	1	47%	–
24	0.075	–	–	–	0.374	**1**	74%	–	0.192	–	–	–	0.905	**2**	82%	–	0.358	1	47%	–
25	0.367	3	59%	–	0.477	**1**	74%	–	0.858	**2**	86%	–	0.355	1	49%	–	0.265	2	65%	–
26	0.487	**1**	74%	–	0.727	**1**	74%	–	0.723	**2**	86%	–	0.845	**2**	82%	–	0.575	1	47%	–
27	0.158	–	–	–	0.215	2	51%	–	0.468	**2**	86%	–	0.671	**2**	82%	–	0.524	1	47%	–
28	0.415	2	46%	–	0.225	**1**	74%	–	0.378	**2**	86%	–	0.532	**2**	82%	–	0.176	**3**	87%	–
29	0.585	**1**	76%	–	0.291	**1**	74%	–	0.569	**2**	86%	–	0.732	**2**	82%	–	0.471	1	47%	–
30	0.161	3	59%	–	0.194	2	51%	–	0.356	1	42%	–	0.545	**2**	82%	–	0.939	1	47%	–
31	0.709	**1**	76%	–	0.719	**1**	74%	–	0.251	3	58%	–	0.834	**2**	82%	–	0.661	1	47%	–
32	0.522	**1**	76%	–	0.528	**1**	74%	–	0.248	3	58%	–	0.258	3	62%	–	0.301	1	47%	–