HALO: Semantic-Aware Distributed LLM Inference in Lossy Edge Network

Peirong Zheng[1], Wenchao Xu*[2], Haozhao Wang[3], Jinyu Chen[1], and Xuemin (Sherman) Shen[4], *Fellow, IEEE*
[1]Department of Computing, The Hong Kong Polytechnic University
[2]Division of Integrative Systems and Design, The Hong Kong University of Science and Technology
[3]School of Computer Science and Technology, Huazhong University of Science and Technology
[4]Department of Electrical and Computer Engineering, University of Waterloo
peirong.zheng@connect.polyu.hk, wenchaoxu@ust.hk, hz_wang@hust.edu.cn
jinyu.chen@connect.polyu.hk, sshen@uwaterloo.ca
*Corresponding author: Wenchao Xu

***Abstract*—The deployment of large language models' (LLMs) inference at the edge can facilitate prompt service responsiveness while protecting user privacy. However, it is critically challenged by the resource constraints of a single edge node. Distributed inference has emerged to aggregate and leverage computational resources across multiple devices. Yet, existing methods typically require strict synchronization, which is often infeasible due to the unreliable network conditions. In this paper, we propose `HALO` , a novel framework that can boost the distributed LLM inference in lossy edge network. The core idea is to enable a relaxed yet effective synchronization by strategically allocating less critical neuron groups to unstable devices, thus avoiding the excessive waiting time incurred by delayed packets. `HALO` introduces three key mechanisms: (1) a semantic-aware predictor to assess the significance of neuron groups prior to activation. (2) a parallel execution scheme of neuron group loading during the model inference. (3) a load-balancing scheduler that efficiently orchestrates multiple devices with heterogeneous resources. Experimental results from a Raspberry Pi cluster demonstrate that HALO achieves a 3.41x end-to-end speedup for LLaMA-series LLMs under unreliable network conditions. It maintains performance comparable to optimal conditions and significantly outperforms the state-of-the-art in various scenarios.**

***Index Terms*—Large Language Models, Tensor Parallelism, Edge Computing, Heterogeneity, Semantics, Packet Loss**

I. Introduction

Recently, massive natural language processing (NLP) applications, such as voice assistants and chatbots, machine translation, etc., are emerging in edge environments with a focus on preserving the privacy of users and providing prompt responses [1], [2]. To implement these applications, various transformer models are deployed on edge devices. However, as transformer models, particularly large language models (LLMs), continue to grow in scale, their deployment has become increasingly challenging due to the inherent computational and memory constraints of edge devices.

To tackle this challenge, many approaches propose deploying LLMs on multiple edge devices in a distributed manner via parallel computing. For instance, trusted edge devices in smart home scenario can contribute their computing resources for collaborative LLM inference within a local area network [3] [4], as shown in Figure 1. Existing methods include intra-layer tensor parallelism (TP) [5], inter-layer pipeline parallelism (PP) [6], and sequence parallelism (SP) [7]. For instance, to deal with the resource heterogeneity of multiple edge devices, *LinguaLinked* employs inter-layer PP to increase throughput by considering both the constraints of each device and the inter-dependencies between the layers of the LLMs, which requires accurate synchronization among multiple nodes for an overlapping scheme [6]. Further, considering the one-shot (referring to a single request) inference characteristic of LLM applications in edge, intra-layer TP is gaining broader adoption to reduce the inference latency. For example, *EdgeShard* combines intra-layer TP and inter-layer PP to optimize latency and throughput by considering the heterogeneities in both edge node and servers [8]. *Galaxy* combines SP and intra-layer TP, and employs a tile-wise overlapping method to parallel the communication and model execution processes [3].

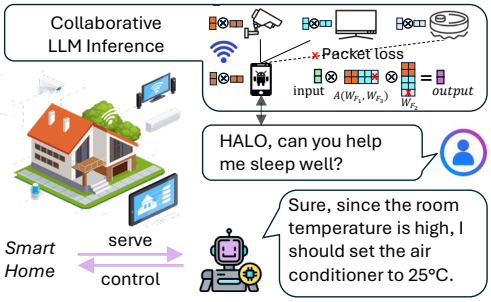


Fig. 1: Smart home scenario empowered by `HALO`: trusted home devices collaboratively serve LLM inference with high accuracy and efficiency in lossy network.

While existing distributed inference mechanisms have achieved considerable success, yet they generally presuppose a stable network environment—an assumption that is rarely feasible in most edge computing scenarios [9], [10]. For instance, the end-to-end packet loss rate (PLR) can be up to 5% even under in-door conditions [11]. For cellar networks, the PLR can be up to 10% and even worse in poor channel areas [12]. The synchronization overhead for distributed inference can be greatly increased due to a relatively high PLR, and thus compromise the LLM performance, e.g, 4.26x slower for the time per output token (TPOT) under 5% PLR in our preliminary experiment.

To enable efficient distributed LLM inference in lossy edge network, we propose HALO, a semantic-aware algorithm-system co-design to aggregate and leverage computational resources from multiple edge devices with unreliable network connections. The core idea of HALO is to allocate less critical neuron groups of LLM to unreliable edge devices while facilitating relaxed yet effective synchronization. HALO enables the inference process to continue without waiting for overly delayed packets while maintaining the inference accuracy. Specifically, the key innovations of HALO are:

- Runtime semantic-aware predictors that identify neuron groups importance for mapping and loading, hence ensuring the effectiveness even for relaxed synchronization.
- An overlapping scheme that parallel the neuron group mapping & loading process with communication and computation, to overcome the system bottleneck and accelerate the LLM inference.
- A PLR-aware scheduler that efficiently adapts to the edge heterogeneity in memory budget and computing power for load balancing and straggler minimization.

To the best of our knowledge, HALO represents the first comprehensive framework to accelerate LLM inference specifically designed for lossy edge network. We address the challenges of slow collaborative inference under packet loss between heterogeneous edge devices by proposing semantic-aware methods to achieve effective relaxed synchronization, combined with meticulous overlapping scheme and PLR-aware load-balancing scheduler to maximize resource efficiency. We implement HALO on a real-world testbed, composed of 8 typical edge devices, Raspberry Pi 4 Model B (RPi), with LLaMA-series LLMs. It achieves 3.41x end-to-end speedup under 5% PLR while preserving comparable accuracy, demonstrating significant improvements compared to existing approaches.

II. Background and Motivation

A. LLM Inference over Edge Devices

Transformer decoder-based LLMs. Modern LLMs follow the generative pretrained transformer (GPT) paradigm, composed of the token embedding table, multiple Transformer decoder layers, and the language modeling head (LM Head). The token embedding table resides solely in memory without requiring computational resources. In the Transformer decoder layer, the multi-head attention (MHA) and multi-layer perceptron (MLP) blocks are the primary computational and memory components suitable for distribution. In addition, the LM Head is also a distributable linear layer. The remainder is element-wise operations connecting these layers, including dropout, normalization, and residual connection.

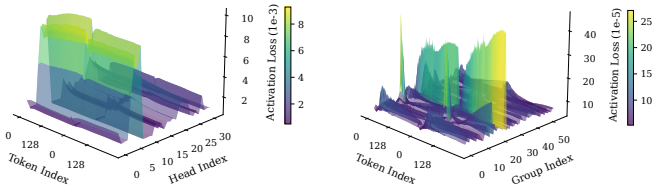
Distributed LLM inference over edge. In the edge scenario, the one-shot inference, such as the chatbot, is dominantly popular. Intra-layer TP [5] is designed for LLM training in the data center, and evenly assigns the workload to all GPUs. It is suitable for reducing latency in one-shot inference [3], [13], [14]. Hence, we propose TP-based methods to utilize each device's computing power and memory simultaneously. TP is composed of consecutive block-wise computation and activation synchronization. The algorithm performance is heavily determined by the straggler (i.e., the one that finishes work last), which can leave other devices idle and increase the overall inference latency. Hence, load balance is of vital importance. Second, the synchronization overhead becomes the bottleneck due to unstable links according to the preliminary experiment on the relation of inference time and PLR.

TABLE I: Comparison of Average TPOT of 64 tokens.

PLR (%)	TPOT (ms)	
	TCP	UDP / 10ms timeout
0	899.45	793.10
1	1531.53	765.66
2	2068.10	818.48
5	3983.52	934.95

Table I shows the relation between different PLRs (0-5%) and TPOT to validate the influence of unstable links and efficiency of unreliable transmission. Eight RPis are involved and benchmarked by TP framework dllama [14] and model TinyLlama [15]. TCP is the default communication protocol providing reliable transmission. UDP with timeout is the protocol for efficiency and provides unreliable transmission. A timeout mechanism is essential because, in its absence, a receiver would wait indefinitely for lost packets, thereby stalling the entire inference process. Hence, we limit the waiting time within the timeout of 10ms empirically. For TCP, as the PLR increases by 1%, the TPOT can increase by about 617.2ms. When the PLR=1%, its overhead nearly accounts for 70% of the TPOT without PLR. Obviously, current communication optimization approaches, such as overlap, will lose effectiveness and cause downgraded system performance in a lossy network. For UDP, as the PLR increases by 1%, the TPOT increases slightly by about 28.4ms. It achieves at most 4.26x speedup than TCP under 5% PLR. Using UDP for uploading packets also allows for faster synchronization, without the overhead of TCP mechanisms under 0% PLR. However, simply adopting UDP transmission could suffer from packet loss. From the application view, model accuracy is severely degraded by activation loss (encapsulated in packets). To preserve model accuracy and exploit the efficiency brought by sacrificed transmission reliability, we need information about packets' importance prior to upload and reserve the transmission reliability for important packets.

Neuron group. TP distributes model weights to each device, with dependent neurons grouped in one device. In the MHA, one head and its output projection weight are an independent neuron group, while in the MLP, each up-projection, gate function, and down-projection can be viewed as one independent neuron group; the LM Head only contains up-projection, which is also a collection of neuron groups. For hardware-friendly implementation and compatibility with quantization, we fix the group size to 256. For instance, the MLP with dimension 6,144 can be divided into 24 distributable neuron groups. We adopt the term neuron groups to refer to the dependent neurons in one block (MHA/MLP/LM Head).



(a) Activation sparsity of neuron groups in the MHA block.

(b) Activation sparsity of neuron groups in the MLP block.

Fig. 2: Activation Distribution of Neuron Groups.

B. Insight: Semantic-Aware Synchronization

> *Insight 1:* Semantic imbalance (activation sparsity) enables relaxed synchronization to be effective.

To optimize activation synchronization of TP in lossy edge network, we delve into the activation distribution during LLM inference. Figure 2 presents the skewed activation distribution among neuron groups. The semantic features are the merging result of all neuron groups' activation. The input is two random samples from GSM8K [16], and the output is layer 3 of Llama-3.1-8B-Instruct-int4 model. The loss of activations refers to the L2 norm of an activation, which indicates its importance. Both (a) and (b) demonstrate that activation patterns exhibit sparsity, where certain heads/groups dominate the merged semantic features at the token level while others maintain minimal contribution. The semantic features, are largely determined by top activations, a subset of total activations. Besides, the two samples' activation distribution patterns are different, especially in the MLP block. This dynamic characteristic limits the effectiveness of neuron group importance statistics.

We conduct a small-scale experiment to verify the imbalanced impact of the neuron groups' activation. In Table II, all layers exhibit the same activation loss method, under 10% PLR and 25% of neuron groups. The perturbed semantic features, which are influenced by the activation loss, change the inference accuracy. It shows that loss of high-norm activation causes a significant accuracy degradation of 47.2%, while loss of low-norm activation degrades merely 1.6% accuracy. The performance of random activation loss is roughly in the middle of the other two methods. This proves that a subset of activations can still preserve comparable inference accuracy. The low-norm activation loss method is not realistically useful because this importance can only be known after the neuron groups have already been computed. Given that semantic features change slowly across layers [17], it is possible to train highly-accurate neural networks to learn the output activation distribution using input semantic features from layers ahead of the current layer.

Opportunities of semantic-aware synchronization. In a lossy network, with SAPs, we could discard strict slow synchronization and enjoy the efficiency of relaxed yet effective synchronization, as shown in the Table I. The benefits are: 1) It reduces synchronization overhead under lossy network; 2) It preserves critical activations, thus maintaining accuracy.

TABLE II: TinyGSM8K [18] Accuracy of Llama-3.1-8B-Instruct-int4 with Different Activation Loss Methods.

Method	Accuracy (%)	Δ (%)
Baseline (No Loss)	71.6	-
Random activation loss	44.4	-27.2
High-norm activation loss	24.4	-47.2
Low-norm activation loss	70.0	-1.6

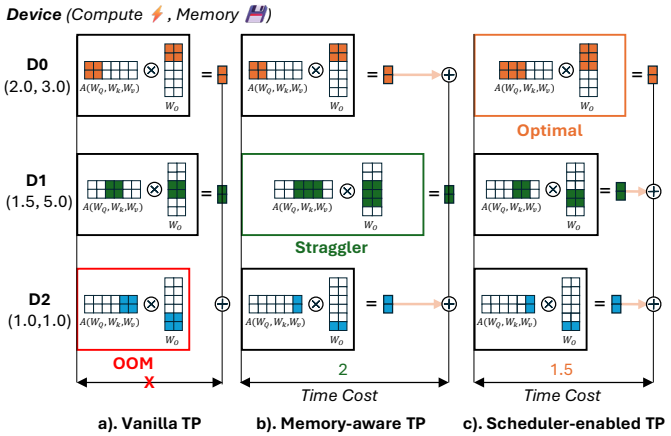


Fig. 3: Time Cost of Different Scheduling Strategies.

C. Insight: Comprehensive Heterogeneity-aware Scheduler

> *Insight 2:* PLR-awareness is fully compatible with load balancing algorithms. The scheduler determines the runtime mapping priority to optimize communication, while load balancing algorithms optimize computation.

Edge environments are characterized by profound heterogeneity in both memory budgets and computing power, posing significant challenges for LLM inference. To tackle these challenges, we start by delving into the characteristics of LLMs' layers, essentially matrix parameters, which contribute linearly to both memory and computing requirements. This attribute makes it possible to allocate the workload linearly by the number of neuron groups. We use one attention block as an example to illustrate the critical need for a comprehensive scheduler in Figure 3. The attention block is distributed along the head dimension to three devices. The number of filled squares represents the workload, and the vertical line with addition represents the merging operation. A naive approach like Vanilla TP (a), which ignores resource limits, immediately fails with an out-of-memory (OOM) error on the memory-constrained device D2. An improved Memory-aware TP (b) avoids OOM but creates a new performance bottleneck by overloading the memory-sufficient & compute-slow device D1, turning it into a straggler that dictates the overall latency while other devices sit idle. This highlights that heuristic optimization for memory and compute is suboptimal. Scheduler-enabled TP (c) demonstrates the power of a holistic approach. By jointly considering both memory budgets and computing power, it achieves load balancing, mitigates straggler effect, and minimizes inference time. This insight drives us to utilize

all available resources for optimal performance.

As shown in Table I, packet loss can severely increase the communication time. To reduce this latency, an intuitive method is to alter the load distribution to evict devices associated with high PLR. The rationale would be to reduce communication overhead by reducing the participating devices. However, such a strategy decreases the total computing power, increases computation time, and introduces the need for a complex trade-off optimization. The presence of PLR needs an informative scheduler that preserves critical PLR information without interfering with load balance. Accordingly, our scheduler is designed to integrate the PLR order of each participant directly into its schedule, while strictly upholding the optimal load balance. The core of our approach is a simple yet effective technique: we transform neuron group indices into *neuron group priority indices* (see details in Algorithm 3). These priority indices, in conjunction with real-time results from our SAPs, guide the dynamic updates of neuron groups (see Pred process in Figure 5). This mechanism enables us to obtain the communication time savings afforded by relaxed synchronization, without incurring increased computation time.

Opportunities of comprehensive scheduling. A comprehensive scheduler offers a dual advantage: 1). It is designed to guarantee optimal load balance across heterogeneous devices. 2). It embeds PLR-awareness directly into the scheduling indices, enabling highly efficient, dynamic updates at runtime.

III. System Design

A. Overview

Figure 4 presents HALO, which features three phases: *Profiling Phase, Assignment Phase, and Generation Phase*. *Profiling* phase profiles both the model and edge devices. HALO profiler utilizes a prior dataset to gather the activation of neuron groups and train the SAPs accordingly, which is an offline one-time operation. We exploit the temporal stability of edge environments by conducting pre-inference profiling, assuming conditions remain almost constant during execution. Before each inference, HALO profiler receives user-assigned memory budgets and utilizes Linux tools to obtain the devices' information, including computing power and PLR. This profiling information is then used in the following phases.

Assignment phase is the generation of the scheduler for each device. It aims to map devices with neuron group priority indices. Based on profiling the model and devices, the scheduler can determine the workload ratio and then map the neuron group priority indices to each device to ensure load balance. The indices are a list of integer sequence numbers.

Generation starts with the master node's command. Then, each worker reads the assigned neuron groups according to the scheduler. After the full model has been loaded, the generation process begins. During the Generation, Computation (Comp) and communication (Comm) of TP are executed in the system until all layers have been forwarded. The predicting (Pred) and loading (Load) processes are executed in parallel with the original TP inference. Finally, the master node gathers the logits for token decoding and continues this auto-regressive process to generate the next token. This parallel execution scheme is detailed in Figure 5 and elaborated in Section III-B3.

TABLE III: Notations for Formulas and Algorithms

Notation	Definition
Device-Related Notations	
n	The total number of available devices.
i	The index for a device, where $i \in \{1, \ldots, n\}$.
m_i	The memory budget (e.g., in MB) of device i.
c_i	The computing power (e.g., in TFLOPS) of device i.
$\boldsymbol{r}$	The vector of memory allocation ratios for n devices, $\boldsymbol{r} = (r_1, \ldots, r_n)$, where $\sum_{i=1}^{n} r_i = 1$.
$\boldsymbol{\rho}$	The vector of PLRs for n devices, $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$.
Model-Related Notations	
$\mathcal{M}_t$	The total memory demand of the model.
L	Number of Transformer decoder layers in the model.
D	The dimension of the hidden states.
N_h	Number of attention heads in one MHA block.
N_{kv}	Number of key-value (KV) heads.
N_g	Number of neuron groups in one MLP block.
N_v	Number of neuron groups in the vocabulary of LM Head.

B. Workflow and Algorithms

We consider 3 aspects: memory budget, computing power, and PLR. The top priority is the total memory budget, which determines whether the model and KV cache can be accommodated. Table III includes the notation for model and device informations. We denote the workload for the MHA, MLP, and LM Head in the same form of neuron groups in the algorithms.

1) Problem Formulation: Since the memory budget is the primary constraint determining inference feasibility, and the optimization goal is to minimize the heterogeneous cluster inference time, we formulate this as an integer linear programming (ILP) problem. The solution to this problem is the optimal workload schedule, which determines the number of neuron groups for each device. In summary, it can be formulated as an ILP problem with an objective function under constraint of each device's resources and LLM demands:

$$\begin{aligned} \underset{W}{\text{argmin}} \quad & \sum_{l=1}^{L} \sum_{k \in \{h,g\}} \max_{i=1,\ldots,n} \left\{ \frac{\tau_k \cdot W_{l,i,k}}{c_i} \right\} + 2L(n-1)T_c \\ \text{s.t.} \quad & \sum_{i=1}^{n} W_{l,i,k} = N_k, \qquad \forall l \in \{1, ..., L\}, \qquad k \in \{h, g\} \\ & \sum_{l=1}^{L} \sum_{k \in \{h,g\}} (W_{l,i,k} \cdot \text{mem}_k) \leq m_i, \qquad \forall i \in \{1, ..., n\}. \end{aligned} \tag{1}$$

The two terms are computation and communication latency. The LLM is composed of L Transformer layers. Each layer contains N_h attention heads in the MHA and N_g neuron groups in the MLP. We aim to distribute this model across n heterogeneous devices. The core of our optimization is the variable $W_{l,i,k}$, which is a non-negative integer representing the number of units of type k (where $k \in \{h, g\}$) from layer l that are assigned to device i. The computational cost to process a single attention head or a single MLP group is given by τ_h and τ_g, respectively. Similarly, the memory required to store them is denoted by mem_g and mem_h. In the formula, we use the generic terms τ_k and mem_k to represent these costs depending on the component type k.

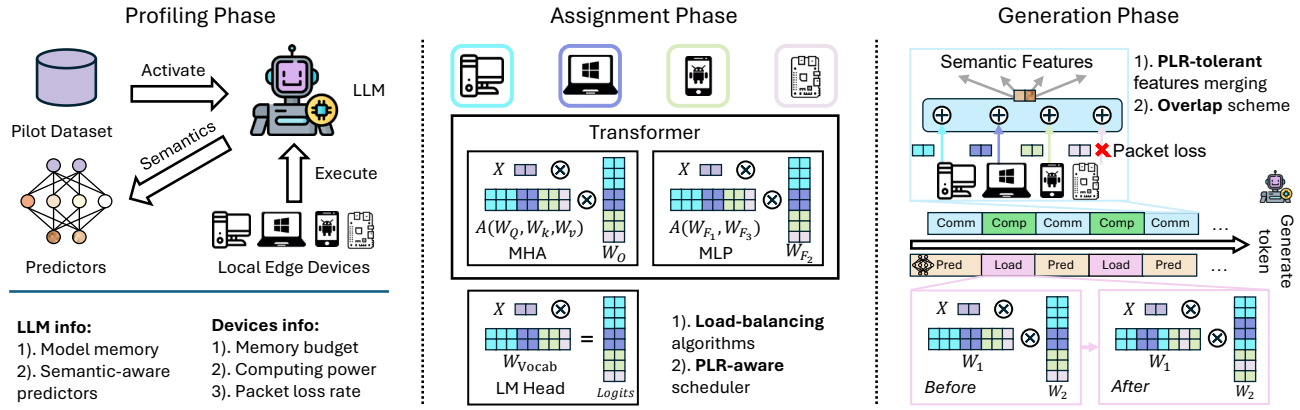


Fig. 4: Overview of the System Design. Three phases are executed in sequence. Four edge devices collaboratively inference one LLM. In the *Assignment* phase, their four colors and the number of filled squares indicate the assigned workload. $A()$ refers to the attention computation in the MHA block and activation function computation in the MLP block.

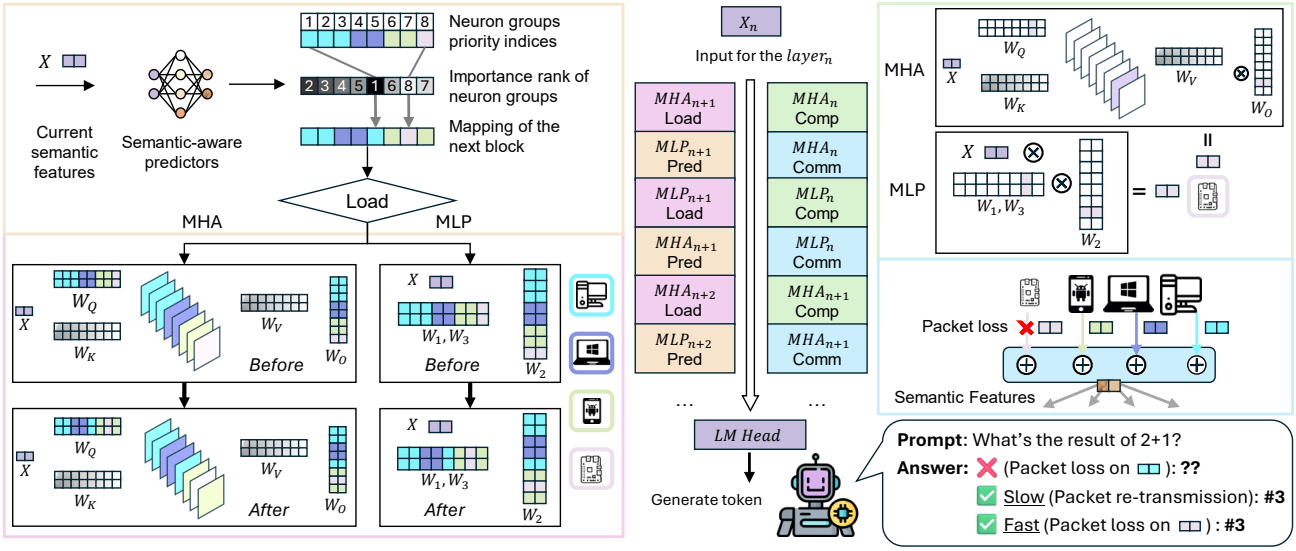


Fig. 5: *Generation* Phase with Overlapped Processes. Load&Comp and Pred&Comm are overlapped across layers during runtime. Four processes have four colors, and the block lines, containing detailed illustrations, are in the same color. For illustration simplicity, we ignore the LM Head, which is essentially similar to the MLP block.

For the communication latency, $2L(n-1)$ refers to the times of synchronization from participants to the master. T_c represents the communication time required for synchronization between devices after each block's computation. Note that it is primarily determined by the hidden dimension D and the link attributes, and thus is treated as a constant with respect to the workload distribution W. For instance, to calculate T_c, when $D = 4096$ and link bandwidth is $1000Mbps$, the activation is $16KB$, then T_c is $0.262ms$. The objective function in Equation (1) is designed to minimize the overall latency.

2) Semantic-Aware Predictors (SAPs): To reduce the communication overhead, we delve into the activation of neuron groups. Leveraging the insight on semantic imbalance and activation sparsity discussed in Section II-B, we design SAPs to predict critical neuron groups dynamically during runtime. Inspired by DEJAVU [17], we gather the LLM activation and train our predictors.

Our SAPs differ from DEJAVU [17]'s predictors. 1). We use the activation norm as training data for fine-grained importance metric instead of processing it using sigmoid for two-class classification. 2). We adopt one layer ahead instead of the one block (MHA/MLP) ahead for overlap efficiency in Figure 5. 3). We decrease the hidden dimension of predictors to 256 and 512 for 8B and 13B models instead of 1000. However, our modified predictors still have high performance; the SAPs training result achieves an average validation loss of 2.1% and 2.4% for MHA and MLP. Furthermore, we demonstrate the ability to maintain model accuracy in Section V-B.

3) Packet-Loss-Tolerant Fast Generation: This phase represents the actual inference stage. There are two designs to achieve end-to-end acceleration. Firstly, to eliminate the packet retransmission overhead, our relaxed synchronization design tolerates packet loss. The master node proceeds to merge available activations after a timeout period, rather than waiting

for lost packets indefinitely and causing system deadlock. This timeout is set as $10ms$ empirically. By proactively assigning neuron groups predicted to be less important to devices with higher PLR, our method obviates costly retransmissions.

Second, to hide the latency of Pred and Load process, we design this overlap scheme as shown in Figure 5. 1). In the Pred process, each device's SAPs predict the next layer's importance rank of neuron groups locally using current semantic features as input. And then the importance ranks serve as a reference list for the neuron group priority indices, given in the *Assignment* phase, to further determine mapping of local neuron group indices of the next block. 2). In the Load and Comp process, each device will execute the assigned neuron groups and output its activation, which is two squares in the same color. Our method utilizes the linear layer's characteristics: Load time and Comp time both exhibit a linear relation with the number of neuron groups (i.e, parameter count). MLP has about 2 times the parameter count of MHA, which hinders the simplicity of the overlap scheme when the Load is one block ahead of Comp. In this case, the slower MLP Load will execute during the faster MHA Comp process. Similarly, the faster MHA Load will execute during the slower MLP Comp, causing a waste of waiting time. In contrast, when the Load is one layer ahead of the Comp, the latency of both processes for one block has a linear relation with the same number of neuron groups, thus they can be overlapped by easily allocating CPU threads for the two processes. The limitation of this design is that each device should reserve a full KV cache locally in case needed by runtime-assigned neuron groups in MHA. But this memory overhead is minimal when the sequence length is not extremely long. For example, for Llama 8B model with sequence length 1024, KV cache requires $1024 * L * 2 * D$ elements, only 3.3% of the model size. 3). In the Comm process, for this example, the RPi has the highest PLR and is assigned the least important neuron groups. Our method schedules it to upload the least important activation, shown in the lightest color. As a result, the semantic features are not severely influenced even when the activation loss occurs. For the first layer, we use TCP because no semantic information can be given by SAPs. For the LM Head, we use TCP to ensure the logits are accurate. Finally, the LLM's answer is a correct #3 in Figure 5, showing that HALO delivers end-to-end acceleration while preserving accuracy.

4) PLR-Aware Load-balancing Scheduler (PLS): PLS contains 2 steps to distribute workload by finding ratios and mapping indices. First, to solve the NP-hard ILP problem, we approximate it as a linear programming problem. Algorithm 1 is a greedy approach that first allocates tasks to the fastest devices until their full capacity is utilized, before assigning the remaining load to the next fastest devices in sequence. Algorithm 2 minimizes the maximum computation time among all devices to mitigate the straggler impact. It achieves this by maximizing the utilization of the entire parallel computing power. Second, Algorithm 3 then maps the resulting workload ratios to neuron group priority indices, prioritizing network-stable devices for more critical activation. For instance, to illustrate the function of Algorithm 3, consider partitioning 8 neuron groups among three devices, and the prioritized indices are $[1, ..., 8]$. Let the PLRs $\boldsymbol{\rho} = [0.0, 0.5, 0.1]$ and workload ratios $\boldsymbol{r} = [0.4, 0.4, 0.2]$. Then the result would be $[1, 2, 3], [6, 7, 8], [4, 5]$, which makes sure that the one with the highest PLR gets the least prioritized indices.

Algorithm 1: Computation-Greedy Scheduler

Input: $\boldsymbol{m} = \{m_1, \ldots, m_n\}$: Memory budgets of n devices. $\boldsymbol{c} = \{c_1, \ldots, c_n\}$: Computing power of n devices. $\mathcal{M}_t$: Model's total memory demand.

Output: $\boldsymbol{r}$: Memory allocation ratios vector.

Assert $\sum_{i=1}^{n} m_i \geq \mathcal{M}_t$;

$\boldsymbol{\sigma} \leftarrow$ argsort($\mathcal{C}$, direction='descending') ;

$k \leftarrow \min\{j \mid \sum_{i=1}^{j} m_{\sigma_i} \geq \mathcal{M}_t\}$;

// Minimum number of devices needed

$\mathcal{M}_k \leftarrow \mathcal{M}_t - \sum_{i=1}^{k-1} m_{\sigma_i}$;

// Memory needed for the last device

$$\boldsymbol{u}_i = \begin{cases} m_i & \text{if } i \in \{\sigma_1, \ldots, \sigma_{k-1}\} \\ \mathcal{M}_k & \text{if } i = \sigma_k \\ 0 & \text{otherwise} \end{cases}, \text{for } i = 1, \ldots, n.$$

$\boldsymbol{r} \leftarrow \{\boldsymbol{u}_i / \sum_{i=1}^{n} \boldsymbol{u}_i \mid i \in \{1, \ldots, n\}\}$;

Return $\boldsymbol{r}$;

Algorithm 2: Min-Max Scheduler

Input: $\boldsymbol{m}$. $\boldsymbol{c}$. M_t. ϵ: Precision threshold (10^{-3}).

Output: $\boldsymbol{r}$: Memory allocation ratios vector.

Function SearchThreshold($\boldsymbol{m}, \boldsymbol{c}, M_t, \epsilon$)**:**

  $left \leftarrow 0, \quad right \leftarrow \max(\boldsymbol{m})/\min(\boldsymbol{c})$;

  $\mathcal{T}_{\text{opt}} \leftarrow right$;

  // Optimal utilization threshold

  while $right - left > \epsilon$ **do**

    $mid \leftarrow (left + right)/2$;

    $Used\_Memory \leftarrow \sum_{i=1}^{n} \min(m_i, mid \cdot c_i)$;

    if $Used\_Memory \geq M_t$ **then**

      $\mathcal{T}_{\text{opt}} \leftarrow mid, \quad right \leftarrow mid$;

      // Found valid threshold

    else

      $left \leftarrow mid$; // Insufficient memory

  end

  Return $\mathcal{T}_{\text{opt}}$;

Function AllocateMemoryRatio($\boldsymbol{m}, \boldsymbol{c}, \mathcal{T}$)**:**

  $\boldsymbol{u} \leftarrow \{\min(m_i, \mathcal{T} \cdot c_i) \quad \text{for } i \in \{1, 2, \ldots, n\}\}$;

    // Calculate memory allocation

  $\boldsymbol{r} \leftarrow \{(u_i / \sum_{i=1}^{n} u_i) \quad \text{for } i \in \{1, 2, \ldots, n\}\}$;

    // Normalize to get ratios

  Return $\boldsymbol{r}$;

Assert $\sum_{i=1}^{n} m_i \geq \mathcal{M}_t$;

$\mathcal{T}_{\text{opt}} \leftarrow$ SearchThreshold($\boldsymbol{m}, \boldsymbol{c}, M_t, \epsilon$) ;

Return AllocateMemoryRatio($\boldsymbol{m}, \boldsymbol{c}, \mathcal{T}_{opt}$);

IV. Implementation

We implemented HALO using 3.5k+ LOC in C++ atop [14] and 1.4k+ LOC in Python atop [19]. The C++ implementation

Algorithm 3: PLR-Aware Ratio-Guided Scheduler

Input: Model dims N_h, N_{kv}, N_g, N_v, Device count n, ratios $\boldsymbol{r}$, PLRs $\boldsymbol{\rho}$.

Output: Scheduler $\mathcal{S}$, which contains n devices' partitioning plan for all neuron groups.

$\boldsymbol{q} \leftarrow \text{argsort}(\boldsymbol{\rho}, \text{direction='ascending'})$;

$\boldsymbol{r}_{\text{sorted}} \leftarrow \boldsymbol{r}[\boldsymbol{q}]$; `// Prioritized ratios`

$\mathcal{S}_{\text{temp}} \leftarrow \text{Partition}(\{N_h, N_{kv}, N_g, N_v\}, \boldsymbol{r}_{\text{sorted}})$;
`// Partition the list of neuron groups indices according to` $\boldsymbol{r}_{\text{sorted}}$

$\mathcal{S}[\boldsymbol{q}_k] \leftarrow \mathcal{S}_{\text{temp}}[k] \quad \forall k \in \{1, \ldots, n\}$;
`// Map the k-th list of indices to the k-th prioritized device`

Return $\mathcal{S}$;

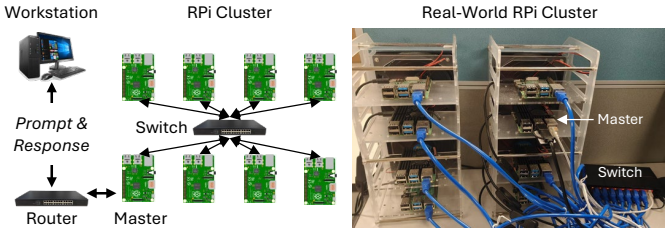


Fig. 6: The Testbed Diagram and Real-World RPi Cluster.

extends `dllama` (version 0.11.0) [14]'s tensor parallelism with indexed model weight loading, hybrid communication backend, overlap functions, and timing statistics. The Python implementation modifies [19] with drop strategies and SAPs execution in the forward function and adds wrapper compatibility for the `lm-eval` framework. `HALO` is a standalone extensible framework and can be implemented using other DL frameworks such as TF-Lite, PyTorch. As shown in Figure 6, the testbed consists of 8 RPis and one consumer-grade switch (Mercusys MS108G with Gigabit). One RPi is the master node responsible for the interactions with the workstation and coordinating the cluster. It also takes computing workload.

V. Evaluation

A. Experimental Setup

1) LLM and Dataset: We evaluate our method on two open-source LLMs, Llama-3.1-8B-Instruct and Llama-2-13B-Chat-Hf, both quantized to 4-bit integer precision. For brevity, their abbreviations are LlaMA3 and LlaMA2. Our goal is to accelerate model inference while maintaining task accuracy.

We used three benchmark datasets to evaluate both prompt evaluation and text generation capabilities. ARC-Easy and ARC-Challenge [20] are multiple-choice datasets, evaluated via the `loglikelihood` function. GSM8K test set [16], a challenging math task, requires reasoning (free-form answer generation) via the `generate_until` function.

2) Edge Computing Scenario: In the edge scenario, we evaluate the end-to-end LLM inference time. We formulate a subset of samples from the GSM8K [16] dataset of sequence length 256. The prompt length is 128, and the generated token length is also 128. To emulate the network instability, we use the Linux tool `tc` to control the master node's link attribute. To emulate the heterogeneity, we adjust the CPU frequency to control the computing power precisely. Table IV shows the specifications for the heterogeneity configurations. Each device CPU frequency ranges in 0.6-1.8 GHz with a step size of 0.1GHz; Each device memory budget ranges in 300-6300 MB with a step size of 200MB. These configurations are randomly generated under only two limitations: 1). the total memory budget is more than the LLM; 2). the average frequency is 1.2 GHz. For homogeneous configurations, all devices are identical in 1.8GHz frequency.

TABLE IV: RPi Specifications and Configurations.

Hardware	Specifications
CPU	1.8 GHz Quad-core 64-bit SoC
RAM	8GB LPDDR4
CPU Computing Power	Range: 0.6-1.8 GHz; Step: 0.1 GHz
Memory Budget	Range: 300-6300 MB; Step: 200 MB

3) Baselines: Vanilla TP [5] is implemented by [14] for all participated edge devices with evenly distribution. *Galaxy* [3] is a two-step scheduler that first allocates by computing power, then redistributes from OOM devices.

4) Metrics: Inference speed: We measure the end-to-end generation time of LLM. Since there's little performance difference between prompt processing and decoding of the `dllama` [14] framework, we gather the overall token count and inference time, then average as time per token (TPT). *Model accuracy:* For evaluation, we employ the `lm-eval` library [21] to evaluate model accuracy (ACC) on text tasks. Strict-match is adopted for answer correctness evaluation. We conduct experiments with four seeds to mitigate randomness and reported the average results.

B. Accuracy Result

Stable performance across tasks and PLR settings. We evaluate ACC performance under varying PLR, where 25% of model neuron groups may experience packet loss in Table V. `HALO` maintains high ACC under different settings. Our method achieves state-of-the-art performance with an average 0.067% ACC loss, compared with Random's 2.91%. Firstly, `HALO` performs excellently across different tasks, showing stable performance not only on knowledge-intensive tasks, ARC Easy and ARC Challenge, but also demonstrating clear advantages on mathematical reasoning problems GSM8K. This cross-task consistency indicates that `HALO` has good generalization capabilities and adaptability to out-of-domain application requirements. Second, `HALO` demonstrates more stable performance across different PLR settings, compared to the Random method. Notably, as the PLR increases (from 0.01 to 0.05), the accuracy of the Random method generally shows a downward trend, while our method maintains relatively stable performance. For example, on the ARC Easy dataset using the LLaMA2 model, the accuracy of the Random method drastically drops 4.59% at PLR=0.05 (compared to 77.82%

TABLE V: Comparison of Model Accuracy across Different Models, Datasets and PLR Settings. The results unit is (ACC/%) and the best results are bold. The upbound ACC of the dataset is in the parentheses after the model.

Method	ARC Easy						ARC Challenge						GSM8K					
Model	LLaMA2 (77.82%)			LLaMA3 (80.68%)			LLaMA2 (46.08%)			LLaMA3 (50.17%)			LLaMA2 (34.87%)			LLaMA3 (68.16%)		
PLR	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05
Random	75.46	74.33	73.23	77.69	75.93	75.08	45.73	44.37	44.11	48.04	48.46	45.73	33.21	33.43	31.24	62.17	67.85	64.82
HALO	**77.36**	**77.40**	**76.94**	**80.26**	**80.05**	**79.12**	**46.25**	**45.99**	**45.99**	**50.85**	**50.60**	**50.85**	**35.33**	**35.10**	**34.57**	**68.92**	**68.54**	**68.01**

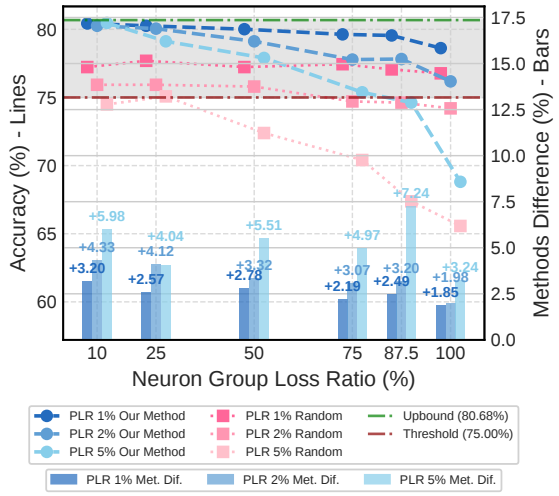


Fig. 7: Comparison of LLaMA3 Accuracy on Arc Easy Dataset with HALO vs. Random Under Different PLR and Neuron Group Loss Ratios (NGLR).

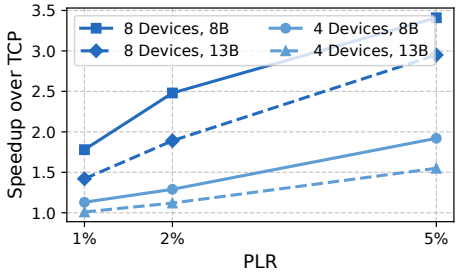


Fig. 8: Speedup (HALO / TCP) of End-to-End Inference Time under Different Configurations. The legend refers to the number of participants and the model size.

upbound), while our method still maintains 76.94%, slightly dropping 0.88% from the upbound.

Potential reasoning performance enhancement. Our method demonstrates interesting performance improvement over upbound. Remarkably, for both LLaMA2 and LLaMA3, HALO at low PLR (0.01 and 0.02) surpasses the upbound on Arc Challenge and GSM8K. The improvement is significant for LLaMA3 on GSM8K, where HALO outperforms the upbound by 0.76% at a 0.01 PLR. We speculate that this improvement stems from a regularization effect that mitigates overfitting. Analogous to Dropout, our method induces more streamlined and focused sub-networks by preserving only the most critical activations, eventually leading to enhanced accuracy.

Robustness under extreme conditions. To investigate the limit of our method's PLR tolerance, we report model performance variations under extreme loss scenarios, where PLR and NGLR gradually increase in this experiment Figure 7. NGLR corresponds to $1 - 1/n$ when there are n devices, $n - 1$ of which experience packet loss. Scenarios of 100% or less than 50% NGLR is not practical because n is a natural number but their theoretical accuracies are valuable. As the lines show, HALO demonstrates remarkable robustness, maintaining an accuracy of around 75%, equivalent to **93%** of the unbound performance 80.68%, across the vast majority of tested scenarios. Only one scenario leads to significant performance degradation: 100% NGLR and 5% PLR, indicating that one block (all neuron groups) could be skipped in 5% probability. This scenario is not practical, as a distributed cluster would at minimum retain the activations from the master node, preventing a total 100% loss. Therefore, our method ensures high accuracy in all plausible packet loss situations. In contrast, the Random method's performance begins to falter much earlier, dipping below 75% accuracy with a PLR of just 2% at higher NGLR. As shown in the bars of Figure 7, most obviously at 5% PLR and 87.5% NGLR, HALO outperforms the Random method with the performance gap peaking at 7.24%. This clearly showcases robustness and the superiority of HALO.

C. Speed Result

Communication acceleration under various settings. Figure 8 demonstrates the speedup of end-to-end inference time (summation of computation time and communication time) across different configurations. Note that HALO only reduces communication time in this homogeneous experiment. The inference accuracy corresponds to the situation of 75% and 87.5% NGLR for 4 and 8 devices. 1). Our method consistently outperforms TCP across all configurations, with speedups ranging from 1.01× to 3.41×. As the PLR increases from 1% to 5%, the speedup becomes more pronounced, reaching up to **3.41×** with (8, 5%) configuration, highlighting the scalability of our approach. 2). A positive correlation exists between the number of participants and speedup. 8-device configurations, with higher communication time ratios, consistently yield superior acceleration compared to their 4-device counterparts. 3). 8B model obtains greater speedups compared to 13B model under the same settings. This is because larger models require higher computation time ratios, so the end-to-end speedup achieved by reducing communication time is less pronounced.
Stable acceleration over heterogeneous devices. In Figure 9, we evaluate n random heterogeneous scenarios for each device count $n \in (4, 6, 8)$, and report the average speedups over

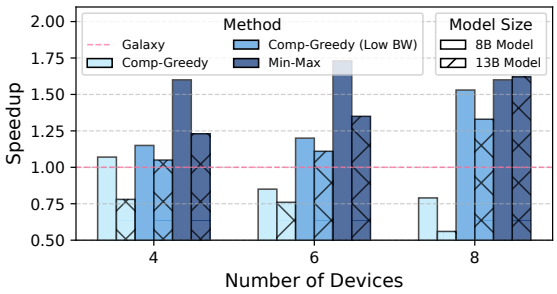


Fig. 9: Average Speedup Across Multiple Heterogeneous Scenarios. Vanilla TP causes the OOM error and is not shown.

TABLE VI: Impact of Overlap Designs.

Overlap Design	TPT Speedup 8B	TPT Speedup 13B	Average
Baseline (No Overlap)	1.00x		
Pred-Comm Overlap Only	1.10	1.25	1.18x
Load-Comp Overlap Only	1.14	1.11	1.13x
Both Overlap Enabled	1.22	1.38	**1.30x**
Upbound (No Pred&Load)	1.47	1.77	1.62x

Galaxy. The results demonstrate the consistent superiority of Min-Max across all scenarios, which achieves at most **1.73x** speedup when 6 devices run 8B model. For Comp-Greedy, it only performs better than Galaxy under low bandwidth (BW) 100Mbps situation by reducing the number of participants (0 in line 5 of Algorithm 1), hence decreasing n in the communication term in Equation (1). Its worse performance under normal bandwidth originates from the lack of full utilization of slower devices. For Min-Max, its advantage becomes more pronounced, from 1.23x to 1.35x to 1.62x with 13B model, as the number of devices increases. This indicates that our Algorithm 2 (by finding $\mathcal{T}_{\text{opt}}$) becomes more effective with larger heterogeneity, particularly valuable for scaling.

D. System Overhead and Ablation

This subsection investigates the overhead and/or function of each component of HALO.

a) Overhead of SAPs and PLS: 1). SAPs Training Overhead. We utilize a small calibration set from SVAMP [22]. The whole training time for a 13B model is 10min22s, negligible as a one-time operation. 2). SAPs Inference Overhead. During inference, SAPs consistently occupies negligible memory and computation resources. For instance, the memory requirement for a quantized 8B model, SAPs's 37MB, is merely **0.82%** of the model's 4.5GB when not considering the KV cache. This proportion is 0.32% with a quantized 7.3GB 13B model. 3). Overhead of PLS. Our PLS execution consumes about **11ms** on an RPi. This scheduling is performed as a one-off preprocessing task before each inference request. It is more efficient than solving the IIP problem.

b) Overlap Scheme Ablation: In Table VI, we configure 3 threads for Comp/Pred and only 1 thread for Load/Comm. We conduct a detailed ablation study on the two overlap designs. The baseline is without any overlap. When implementing Pred-Comm overlap only, we observe a moderate speedup of 1.18x. Similarly, the Load-Comp overlap alone provides a speedup of 1.13x. The most significant performance gains are achieved by employing both overlap designs, delivering 1.22x speedup for the 8B model and 1.38x speedup for the 13B model. Notably, our combined overlap approach achieves **81.25%** of the upper bound 1.62x. Due to the hardware limitation of RPi's quad-core, the configuration zoom is limited. But it could benefit from CPU with more threads and get better overlap efficiency. These results demonstrate that our overlap designs effectively minimize system-inherent overheads.

VI. Related Work

A. LLM on Single Device

To tackle the challenge of massive memory footprint, LLM quantization can reduce the numerical precision of weights at the cost of accuracy. Frantar et al. [23] focuses on preserving accuracy through finding important weights and reducing their quantization degree. Shao et al. [24] aims to preserve activation outliers by learnable weight-activation quantization. Note that quantization methods are orthogonal to and compatible with HALO. Sparsity [17], [25] enables skipping less salient weights without compromising accuracy. Two studies [26], [27] both adopt predictors from Liu et al. [17] to reduce memory footprint during inference by pre-fetching weights to be activated from external storage. However, these methods often do not suffice to accommodate large models inside a resource-constrained edge device. Sparsity can enhance robustness in our method's relaxed synchronization.

B. Packet Loss Optimization

In data centers, the TCP incast [28] is the main challenge for DNN training. Packet loss occurs in the congested switch buffer and motivates in-switch computing works [29], [30]. Specifically, Fei et al. propose *OmniReduce* [31], similar to ours, by exploiting the gradient sparsity to eliminate the transmission overhead for zero-value packets. However, our method differs from theirs, which are solutions for DNN training and not directly applicable to edge LLM inference. More fundamentally, the edge packet loss challenge stems not only from congestion but also from the unreliability of links [10]. Lastly, our goal is to utilize sparsity for effective relaxed synchronization instead of reducing transmission packet count.

VII. Conclusion

In this paper, we have proposed HALO, a novel distributed LLM inference framework specifically tailored to unreliable edge networks. From the semantic perspective, HALO assigns less critical neuron groups to unreliable edge nodes and introduces a parallel scheme to overlap the runtime prediction and the model loading processes. HALO adopts a PLR-aware scheduler incorporating load-balancing algorithms to optimize the utility of edge resources. Experiments on real-world testbed have demonstrated that HALO can maintain accuracy while accelerating the distributed LLM inference up to 3.41x in lossy edge network.

REFERENCES

[1] E. King, H. Yu, S. Lee, and C. Julien, "Sasha: creative goal-oriented reasoning in smart homes with large language models," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 8, no. 1, pp. 1–38, 2024.

[2] Y. Li, H. Wen, W. Wang, X. Li, Y. Yuan, G. Liu, J. Liu, W. Xu, X. Wang, Y. Sun *et al.*, "Personal llm agents: Insights and survey about the capability, efficiency and security," *arXiv preprint arXiv:2401.05459*, 2024.

[3] S. Ye, J. Du, L. Zeng, W. Ou, X. Chu, Y. Lu, and X. Chen, "Galaxy: A resource-efficient collaborative edge ai system for in-situ transformer inference," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024.

[4] W. Xu, J. Chen, P. Zheng, X. Yi, T. Tian, W. Zhu, Q. Wan, H. Wang, Y. Fan, Q. Su *et al.*, "Deploying foundation model powered agent services: A survey," *arXiv preprint arXiv:2412.13437*, 2024.

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

[6] J. Zhao, Y. Song, S. Liu, I. G. Harris, and S. Abdu Jyothi, "LinguaLinked: Distributed large language model inference on mobile devices," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Y. Cao, Y. Feng, and D. Xiong, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 160–171. [Online]. Available: https://aclanthology.org/2024.acl-demos.16/

[7] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 341–353, 2023.

[8] M. Zhang, X. Shen, J. Cao, Z. Cui, and S. Jiang, "Edgeshard: Efficient llm inference via collaborative edge computing," *IEEE Internet of Things Journal*, 2024.

[9] D. R. Bhadra, C. A. Joshi, P. R. Soni, N. P. Vyas, and R. H. Jhaveri, "Packet loss probability in wireless networks: A survey," in *2015 International Conference on Communications and Signal Processing (ICCSP)*. IEEE, 2015, pp. 1348–1354.

[10] I. Ali, S. Hong, and T. Cheung, "Congestion or no congestion: Packet loss identification and prediction using machine learning," in *2024 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2024, pp. 72–76.

[11] R. K. Sheshadri and D. Koutsonikolas, "On packet loss rates in modern 802.11 networks," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[12] J. Rischke, P. Sossalla, S. Itting, F. H. P. Fitzek, and M. Reisslein, "5g campus networks: A first measurement study," *IEEE Access*, vol. 9, pp. 121 786–121 803, 2021.

[13] Z. Li, W. Feng, M. Guizani, and H. Yu, "Tpi-llm: Serving 70b-scale llms efficiently on low-resource edge devices," *arXiv preprint arXiv:2410.00531*, 2024.

[14] B. Tadych, "Distributed llama," https://github.com/b4rtaz/distributed-llama, 2024.

[15] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," *arXiv preprint arXiv:2401.02385*, 2024.

[16] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[17] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re *et al.*, "Deja vu: Contextual sparsity for efficient llms at inference time," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 137–22 176.

[18] F. M. Polo, L. Weber, L. Choshen, Y. Sun, G. Xu, and M. Yurochkin, "tinybenchmarks: evaluating llms with fewer examples," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.

[19] pytorch-labs, "gpt-fast: Simple and efficient pytorch-native transformer text generation in ¡1000 loc of python," https://github.com/pytorch-labs/gpt-fast, 2024.

[20] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[21] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 07 2024. [Online]. Available: https://zenodo.org/records/12608602

[22] A. Patel, S. Bhattamishra, and N. Goyal, "Are nlp models really able to solve simple math word problems?" in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 2080–2094.

[23] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=tcbBPnfwxS

[24] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "Omniquant: Omnidirectionally calibrated quantization for large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=8Wuvhh0LYW

[25] J. Liu, P. Ponnusamy, T. Cai, H. Guo, Y. Kim, and B. Athiwaratkun, "Training-free activation sparsity in large language models," in *The Thirteenth International Conference on Learning Representations*, 2024.

[26] K. Alizadeh, S. I. Mirzadeh, D. Belenko, S. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, and M. Farajtabar, "Llm in a flash: Efficient large language model inference with limited memory," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 12 562–12 584.

[27] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024, pp. 590–606.

[28] J. Zhang, F. Ren, L. Tang, and C. Lin, "Modeling and solving tcp incast problem in data center networks," *IEEE Transactions on Parallel and Distributed systems*, vol. 26, no. 2, pp. 478–491, 2014.

[29] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "Atp: In-network aggregation for multi-tenant learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 741–761.

[30] J. Liu, Y. Zhai, G. Zhao, H. Xu, J. Fang, Z. Zeng, and Y. Zhu, "Inart: In-network aggregation with route selection for accelerating distributed training," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 2879–2889.

[31] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 676–691.