

BALANCE: Hybrid Autoregressive-Speculative LLM Inference in Wireless Edge Networks

Guanqiao Qu, *Graduate Student Member, IEEE*, Shuo Chen, *Graduate Student Member, IEEE*,
Qian Chen, *Member, IEEE*, Kin K. Leung, *Fellow, IEEE*, Xianhao Chen, *Member, IEEE*

Abstract—Edge inference is a promising paradigm to provide large language model (LLM) inference services in next-generation mobile networks. LLM inference mainly relies on two approaches: Autoregressive decoding (AD) generates output tokens sequentially, resulting in long latency; Speculative decoding (SD) accelerates inference by using a small language model (SLM) to generate multiple draft tokens for LLM verification, but incurs extra memory costs. Due to this latency-memory tradeoff, neither approach alone can efficiently serve users with heterogeneous demands under limited edge computing resources. To address this challenge, we propose a hybrid autoregressive-speculative inference (BALANCE) framework for edge LLM inference. In BALANCE, an edge server hosts both an SLM and an LLM, assigns each user to AD or SD, and performs the two modes simultaneously. To maximize the number of served users, we formulate a task throughput maximization problem to jointly determine user scheduling and computing resource allocation between AD and SD under user latency requirements and server memory constraints. Since the problem is NP-hard, we develop a polynomial-time algorithm that transforms the original problem into two sub-problems and obtains a sub-optimal solution with a constant approximation guarantee. Experiments demonstrate that BALANCE consistently outperforms conventional AD and SD and significantly improves task throughput.

Index Terms—Large language models, edge inference, autoregressive decoding, speculative decoding, user scheduling, task throughput maximization.

I. INTRODUCTION

In next-generation mobile networks, large language model (LLM) inference at the network edge is emerging as an important paradigm for delivering LLM services to users [1]. In a typical edge LLM inference system, users upload their prompts to an edge server, which performs inference using the hosted LLM and returns the generated responses [2]. By placing LLMs on edge servers closer to end users, edge LLM inference can reduce transmission latency, compared with cloud-based serving [3], [4]. As such, edge LLM inference is becoming a key enabler for real-time LLM applications [5].

Autoregressive decoding (AD) and *speculative decoding (SD)* are two representative approaches in LLM inference.

Guanqiao Qu, Qian Chen, and Xianhao Chen are with the Department of Electrical and Computer Engineering, The University of Hong Kong, Hong Kong SAR, China (e-mail: gqqu@eee.hku.hk; qchen@eee.hku.hk; xchen@eee.hku.hk). Shuo Chen is with the Electrical and Electronic Engineering Department, Imperial College, London SW7 2BT, U.K. (e-mail: shuo.chen22@imperial.ac.uk). Kin K. Leung is with the Electrical and Electronic Engineering, and Computing Departments, Imperial College, London SW7 2BT, U.K. (e-mail: kin.leung@imperial.ac.uk). The work was supported in part by the Research Grants Council of Hong Kong under Grant 27213824, Grant 17207826, and Grant CRS HKU702/24. Kin K. Leung is supported by the EPSRC grant EP/Y037243/1. (Corresponding author: Xianhao Chen.)

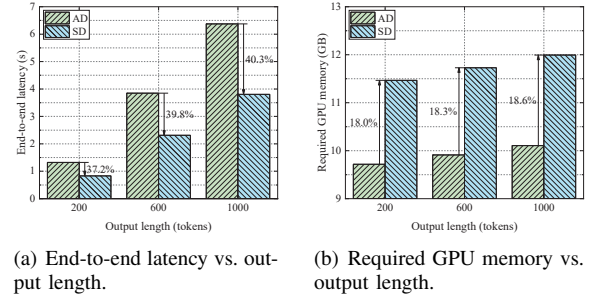


Fig. 1. Comparison between AD and SD when serving a single user using an NVIDIA GeForce RTX 4090 GPU, where the output length varies, and the input length is fixed at 500 tokens. In this experiment, the end-to-end latency denotes the time from feeding the prompt to the model to completing output generation. Llama-2-7B is used as the LLM in both AD and SD, and TinyLlama-1.1B is used as the SLM in SD.

In AD, an LLM generates the response token by token, producing one new token in each forward pass based on the input and previously generated tokens [6], [7]. However, the sequential decoding process of AD incurs high response latency [8]. This issue becomes more severe for tasks with long output responses and stringent latency requirements [8]. To reduce decoding latency, SD has been proposed [9]. In SD, a small language model (SLM) first generates multiple draft tokens via AD, and the target LLM verifies them in parallel within a single forward pass. By accepting multiple verified draft tokens at once, SD reduces the number of serial LLM forward passes. This reduction can significantly reduce response latency, since each forward pass of the SLM is typically faster than the target LLM [10]. Meanwhile, SD maintains output quality, since the verification preserves the target LLM's output distribution [11]. When applying SD to edge LLM inference, the SLM and LLM can be co-deployed on the edge server to serve users [12]. However, despite its latency advantage, SD introduces higher memory overhead due to the coexistence of the SLM and LLM as well as the memory for parallel draft verification [13].

Challenge. Neither AD nor SD alone can efficiently serve users with heterogeneous demands on resource-constrained edge servers. Serving users with AD alone is memory-efficient but may fail to satisfy stringent latency requirements, whereas relying solely on SD reduces latency but may exceed available memory resources [14], [15]. As illustrated in Fig. 1, when serving a single user, SD reduces the average end-to-end (E2E) latency by 39.1% compared with AD, but increases the GPU memory consumption by 18.3%. This latency-memory tradeoff raises a key challenge: *how to efficiently serve users with heterogeneous demands by jointly exploiting AD and SD under*

constrained computing and memory resources on edge servers.

Solution. To address the above challenge, we propose a hybrid autoregressive-speculative inference (BALANCE) framework. In BALANCE, an edge server hosts both an LLM and an SLM, enabling the simultaneous support of AD and SD for different users. To serve users with heterogeneous demands under limited computing resources, the edge server jointly selects users for service, assigns each selected user to AD or SD, and allocates computing resources between the two modes. Based on this framework, we formulate a joint user scheduling and computing resource allocation problem to maximize the number of served users in edge LLM inference while satisfying users' E2E latency requirements and the server's memory constraint. This problem is combinatorial because different assignments of users with heterogeneous demands between AD and SD incur different latency and memory costs, and we show that it is NP-hard. To address this problem, we further design a polynomial-time algorithm with a constant approximation guarantee. To the best of our knowledge, this is the first work to study hybrid autoregressive-speculative inference for edge LLM inference. The main contributions are as follows.

- 1) We propose a hybrid autoregressive-speculative inference framework where an edge server simultaneously supports AD and SD for multiple users. We formulate a joint user scheduling and computing resource allocation problem to maximize the number of served users under E2E latency requirements and resource constraints. We show that this problem is NP-hard.
- 2) To facilitate efficient solution finding, we decompose the original problem into AD and SD sub-problems and design a polynomial-time algorithm with a constant approximation guarantee.
- 3) Experimental results show that the proposed BALANCE framework significantly improves the system performance compared with conventional AD and SD.

The rest of this paper is structured as follows. Section II reviews the related work. Section III proposes the BALANCE framework and formulates the task throughput maximization problem. Section IV designs the algorithm and provides the theoretical analysis. Section V provides the experiment results, and Section VI concludes the paper.

II. RELATED WORK

Existing edge LLM inference systems commonly adopt AD to provide services to end users. To improve system throughput, prior studies have investigated batching techniques that group users requesting the same LLM into one inference batch [16]. Moreover, user scheduling and resource allocation for multi-user edge inference systems under heterogeneous user demands and constrained server resources have also been studied [17], [18]. Some works further explore *collaborative inference*, where an LLM is partitioned across end devices, edge servers, and cloud servers to exploit distributed computing resources [19]. These studies provide important system and scheduling foundations for edge LLM inference under AD.

To reduce response latency, SD has been introduced into edge LLM inference. Similar to AD, existing studies investigate batching, user scheduling, and resource allocation for SD-based edge inference [20], [21], as well as collaborative SD at the network edge, where the SLM and the LLM are hierarchically deployed across end devices, edge servers, and cloud servers [22]. To further improve the efficiency of SD, some techniques, such as tree-structured draft generation [23] and parallel draft generation and verification [24], have been proposed. Together, these studies advance the efficient deployment of SD in edge networks.

Most existing works focus on either AD or SD alone. Although several studies have explored hybrid LLM inference, they mainly consider distributed deployment of SLMs and LLMs [25], request routing between SLMs and LLMs [26], and selective LLM involvement during SLM token generation [27]. These hybrid inference methods coordinate SLMs and LLMs rather than jointly supporting AD and SD. Moreover, some works combine different decoding techniques, such as AD for early tokens followed by parallel decoding for subsequent tokens [28] and retrieval-based draft generation within SD [29]. However, such combinations represent a new inference paradigm rather than joint coordination of AD and SD under resource constraints. Therefore, user scheduling and resource allocation for hybrid autoregressive-speculative inference remain largely unexplored in edge LLM inference.

III. FRAMEWORK

A. Network Scenario

We consider a multi-user wireless edge network with a single edge server covering a set of users $\mathcal{K}$, as shown in Fig. 2. Each user $k \in \mathcal{K}$ generates an inference task with an E2E latency requirement T_k and requests the edge server to perform inference by uploading the task data to the edge server. The edge server hosts both an LLM and an SLM and simultaneously supports the **AD mode**, where the LLM generates output tokens autoregressively, and the **SD mode**, where the SLM generates draft tokens and the LLM verifies them. To serve heterogeneous user demands, the edge server selects users and schedules each selected user to AD or SD mode. To indicate the user scheduling, we define binary variables x_k and y_k , where $x_k = 1$ if and only if user k is scheduled to the AD mode, and $y_k = 1$ if and only if user k is scheduled to the SD mode. $x_k = y_k = 0$ indicates that user k is not served. To improve task throughput, users scheduled to the same mode are batched for inference. After inference, the generated outputs are returned to the users.

B. BALANCE Framework

The procedure of BALANCE is shown in Fig. 3. Before task execution, each user uploads its input length L_k^I and expected output length L_k^O , measured in tokens, to the edge server through a control signal¹. Based on the reported information,

¹Reporting input and expected output lengths is commonly adopted in scheduling for LLM inference [30]. L_k^O can be specified by the task requirements or obtained by efficient prediction methods, such as lightweight predictors [31] and distribution-based prediction [32].

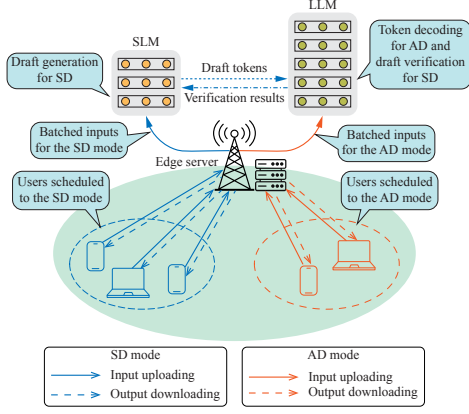


Fig. 2. Illustration of BALANCE, where an edge server simultaneously supports the AD and SD modes for different users. The blue and orange lines denote the data flows of the SD and AD modes, respectively.

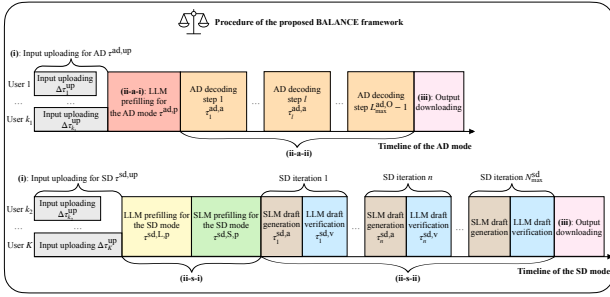


Fig. 3. The procedure of the proposed BALANCE framework.

the edge server determines the user scheduling and sends the scheduling results to the users. The scheduling problem will be formulated in Section III-G. (i) After receiving the scheduling results, the scheduled users upload their input prompts. Once the prompts of users scheduled for the same inference mode are received, the server processes them in a batch using that mode. (ii-a-i) In the AD mode, the prompts for AD are first prefilled by the LLM. (ii-a-ii) Then, the LLM generates the output tokens through multiple decoding steps. (ii-s-i) In the SD mode, the prompts for SD are first prefilled by the LLM and then by the SLM. (ii-s-ii) Then, the SLM and LLM generate output tokens over multiple iterations. (iii) Once one mode is completed, the corresponding outputs are returned to the users scheduled to that mode.

C. Communication Model

We consider orthogonal frequency-division multiple access for both uplink and downlink transmissions. The data uploading time of user k is $\Delta\tau_k^{\text{up}} = \frac{\gamma L_k^{\text{I}}}{B_k R_k}$, where B_k and R_k denote the uplink bandwidth and uplink spectral efficiency of user k , respectively. γ is the number of bits per token, and γL_k^{I} is the bit size of user k 's input prompt tokens. Thus, the data uploading times of the AD and SD modes are given by

$$\tau^{\text{ad,up}} = \max \{ \Delta\tau_k^{\text{up}} \mid x_k = 1 \}, \quad (1)$$

and

$$\tau^{\text{sd,up}} = \max \{ \Delta\tau_k^{\text{up}} \mid y_k = 1 \}, \quad (2)$$

respectively. The output downloading time of user k is

$$\tau_k^{\text{down}} = (x_k + y_k) \Delta\tau_k^{\text{down}}. \quad (3)$$

Here, $\Delta\tau_k^{\text{down}} = \frac{\gamma L_k^{\text{O}}}{\bar{B}_k \bar{R}_k}$, where $\bar{B}_k$ and $\bar{R}_k$ denote the downlink bandwidth and spectral efficiency of user k , respectively.

D. Computing Model

We model the computing procedures and workloads using a common Transformer architecture. Following previous works [33], we focus on matrix multiplication operations in workload analysis, including query, key, and value (QKV) projections, multi-head attention (MHA) computation, output projection, and feed-forward computation using a feed-forward network (FFN), while ignoring minor contributors.

1) *Computing procedures and workload analysis of the AD mode:* After all user input prompts of the AD mode arrive, the edge server starts the AD mode. These prompts are batched and prefilled by the LLM in one forward pass, which generates the first output token for each request. We define the LLM workload for prefilling a single input prompt of L^{I} tokens as $\Gamma^{\text{L,p}}(L^{\text{I}}) = M^{\text{L}} [8L^{\text{I}} D^{\text{L,m}} D^{\text{L,h}} H^{\text{L}} + 4(L^{\text{I}})^2 D^{\text{L,h}} H^{\text{L}} + 4L^{\text{I}} D^{\text{L,m}} D^{\text{L,f}}]$, where $D^{\text{L,m}}$, $D^{\text{L,h}}$, H^{L} , $D^{\text{L,f}}$, and M^{L} denote the LLM hidden dimension, the hidden dimension of an attention head, the number of attention heads, the FFN hidden dimension, and the number of layers of the LLM, respectively. The three terms inside the bracket represent the workloads of the QKV and output projections, attention computation, and feed-forward computation in each layer, respectively. Thus, the LLM prefilling workload in the AD mode is given by

$$W^{\text{ad,p}} = \Gamma^{\text{L,p}}(L_{\max}^{\text{ad,I}}) \sum_{k \in \mathcal{K}} x_k, \quad (4)$$

where $\sum_{k \in \mathcal{K}} x_k$ is the number of users scheduled to the AD mode, and $L_{\max}^{\text{ad,I}} = \max_{k \in \mathcal{K}} \{L_k^{\text{I}} \mid x_k = 1\}$ is the maximum input length of users scheduled to the AD mode².

After prefilling, the LLM generates one token for each unfinished user request at each decoding step. Since prefilling has generated the first token, user k requires $L_k^{\text{O}} - 1$ decoding steps for output generation. Therefore, the total number of decoding steps in the AD mode is $L_{\max}^{\text{ad,O}} - 1$, where

$$L_{\max}^{\text{ad,O}} = \max_k \{L_k^{\text{O}} \mid x_k = 1\} \quad (5)$$

is the maximum output length of users in the AD mode. We define the LLM workload for generating one token at the l -th decoding step for a single input prompt of L^{I} tokens as $\Gamma_l^{\text{L,a}}(L^{\text{I}}) = M^{\text{L}} [8D^{\text{L,m}} D^{\text{L,h}} H^{\text{L}} + 4(L^{\text{I}} + l) D^{\text{L,h}} H^{\text{L}} + 4D^{\text{L,m}} D^{\text{L,f}}]$, where the attention workload depends on the sequence length $L^{\text{I}} + l$, since the query of the current token attends to the KV cache of the previous $L^{\text{I}} + l - 1$ tokens as well as the key and value of the current token. Thus, the LLM workload at the l -th decoding step in the AD mode is given by

$$W_l^{\text{ad,a}} = \sum_{k \in \mathcal{K}} x_k \mathbb{I}_{\{l \leq L_k^{\text{O}} - 1\}} \Gamma_l^{\text{L,a}}(L_{\max}^{\text{ad,I}}), \quad (6)$$

where users with $l > L_k^{\text{O}} - 1$ are excluded at decoding step l .

²Following previous works [34], we consider padding for efficient GPU matrix multiplication, as supported and recommended by PyTorch [35].

2) *Computing procedures and workload analysis of the SD mode:* After all input prompts of users scheduled to the SD mode are received, the edge server starts the SD mode. These prompts are batched, first prefilled by the LLM, and then prefilled by the SLM. Following (4), the LLM prefilling workload in the SD mode is given by

$$W^{\text{sd,L,p}} = \Gamma^{\text{L,p}}(L_{\max}^{\text{sd,I}}) \sum_{k \in \mathcal{K}} y_k, \quad (7)$$

where $\sum_{k \in \mathcal{K}} y_k$ is the number of users scheduled to the SD mode, and $L_{\max}^{\text{sd,I}} = \max_{k \in \mathcal{K}} \{L_k^{\text{I}} \mid y_k = 1\}$ is the maximum input length of users scheduled to the SD mode. Additionally, following $\Gamma^{\text{L,p}}(L^{\text{I}})$, we define the SLM workload for prefilling a single input prompt of L^{I} tokens as $\Gamma^{\text{S,p}}(L^{\text{I}}) = M^{\text{S}} \left[8L^{\text{I}} D^{\text{S,m}} D^{\text{S,h}} H^{\text{S}} + 4(L^{\text{I}})^2 D^{\text{S,h}} H^{\text{S}} + 4L^{\text{I}} D^{\text{S,m}} D^{\text{S,f}} \right]$, where $D^{\text{S,m}}$, $D^{\text{S,h}}$, H^{S} , $D^{\text{S,f}}$, and M^{S} denote the SLM hidden dimension, the hidden dimension of an attention head, the number of attention heads, the FFN hidden dimension, and the number of layers of the SLM, respectively. Thus, the SLM prefilling workload in the SD mode is expressed as

$$W^{\text{sd,S,p}} = \Gamma^{\text{S,p}}(L_{\max}^{\text{sd,I}}) \sum_{k \in \mathcal{K}} y_k. \quad (8)$$

After prefilling, SD proceeds over multiple iterations. In each iteration, the SLM autoregressively generates draft tokens for unfinished requests, and the LLM verifies the batched draft sequences in one forward pass. After verification, the accepted draft tokens and one additional token are retained for each request, and the KV cache of rejected draft tokens is removed. The additional token is used as the input token for the next draft generation. Let A_k denote the expected acceptance rate that a draft token is accepted by the LLM for user k 's task. The number of accepted draft tokens for user k 's task is given by $L_k^{\text{A}} = \frac{A_k [1 - (A_k)^{L^{\text{D}}}]}{1 - A_k}$. User k requires $N_k = \lceil \frac{L_k^{\text{O}}}{L_k^{\text{A}} + 1} \rceil$ SD iterations, where term 1 represents the additional token. Therefore, the total number of SD iterations is denoted by

$$N_{\max}^{\text{sd}} = \max_{k \in \mathcal{K}} \{N_k \mid y_k = 1\}. \quad (9)$$

Moreover, let $L_n^{\text{sd,P}}$ be the prefix length of the retained tokens before draft generation in iteration n , which is given by

$$L_n^{\text{sd,P}} = \begin{cases} L_{\max}^{\text{sd,I}}, & n = 1, \\ L_{\max}^{\text{sd,I}} + (n-2) \max_{\substack{k \in \mathcal{K} \\ y_k = 1}} \{L_k^{\text{A}} + 1\} + \max_{\substack{k \in \mathcal{K} \\ y_k = 1}} \{L_k^{\text{A}}\}, & n \geq 2. \end{cases} \quad (10)$$

For $n = 1$, the retained prefix only consists of the input prompt tokens. For $n \geq 2$, it further includes the accepted draft tokens and additional tokens retained from previous iterations.

Following $\Gamma_l^{\text{L,a}}(L^{\text{I}})$, we define the SLM workload for generating one token for a single input prompt at the l -th decoding step of iteration n , given L_n^{P} prefix tokens available at the beginning of iteration n , as $\Gamma_{n,l}^{\text{S,a}}(L_n^{\text{P}}) = M^{\text{S}} \left[8D^{\text{S,m}} D^{\text{S,h}} H^{\text{S}} + 4(L_n^{\text{P}} + l) D^{\text{S,h}} H^{\text{S}} + 4D^{\text{S,m}} D^{\text{S,f}} \right]$. Thus, the SLM workload at decoding step l of iteration n is

$$W_{n,l}^{\text{sd,a}} = \sum_{k \in \mathcal{K}} y_k \mathbb{I}_{\{n \leq N_k\}} \Gamma_{n,l}^{\text{S,a}}(L_n^{\text{sd,P}}), \quad (11)$$

where users with $n > N_k$ are excluded in iteration n . We define the LLM workload for verifying L^{D} tokens in iteration n for a single input prompt, given L_n^{P} prefix tokens available at the beginning of iteration n , as $\Gamma_n^{\text{L,v}}(L_n^{\text{P}}, L^{\text{D}}) = M^{\text{L}} \left[8L^{\text{D}} D^{\text{L,m}} D^{\text{L,h}} H^{\text{L}} + 4 \sum_{l=1}^{L^{\text{D}}} (L_n^{\text{P}} + l) D^{\text{L,h}} H^{\text{L}} + 4L^{\text{D}} D^{\text{L,m}} D^{\text{L,f}} \right]$. Thus, the LLM workload for verification in iteration n is

$$W_n^{\text{sd,v}} = \begin{cases} \sum_{k \in \mathcal{K}} y_k \mathbb{I}_{\{n \leq N_k\}} \Gamma_1^{\text{L,v}}(L_1^{\text{sd,P}}, L^{\text{D}}), & n = 1 \\ \sum_{k \in \mathcal{K}} y_k \mathbb{I}_{\{n \leq N_k\}} \Gamma_n^{\text{L,v}}(L_n^{\text{sd,P}}, L^{\text{D}} + 1), & n \geq 2. \end{cases} \quad (12)$$

In the first iteration, the LLM verifies L^{D} draft tokens for each request. When $n \geq 2$, the verification further includes the additional token generated in the previous iteration.

E. Memory Model

Following previous works [36], we consider the memory required for storing model weights and the KV cache during inference when analyzing memory consumption.

1) *Memory for hosting the LLM and SLM:* The GPU memory required to store the LLM and SLM weights is

$$q^{\text{L}} = M^{\text{L}} \theta (4D^{\text{L,m}} D^{\text{L,h}} H^{\text{L}} + 2D^{\text{L,m}} D^{\text{L,f}}), \quad (13)$$

and

$$q^{\text{S}} = M^{\text{S}} \theta (4D^{\text{S,m}} D^{\text{S,h}} H^{\text{S}} + 2D^{\text{S,m}} D^{\text{S,f}}), \quad (14)$$

respectively, where θ denotes the number of bytes required to represent one floating-point number [36].

2) *Memory for the AD mode:* In the AD mode, the required KV cache memory for prefilling is $q^{\text{ad,p}} = M^{\text{L}} \theta \left(\sum_{k \in \mathcal{K}} x_k L_{\max}^{\text{ad,I}} \right) 2D^{\text{L,h}} H^{\text{L}}$, where $2D^{\text{L,h}} H^{\text{L}}$ denotes the number of KV cache elements required by one token across all attention heads in each layer. During decoding, the peak KV cache memory is denoted by $q^{\text{ad,a}} = M^{\text{L}} \theta \left[\sum_{k \in \mathcal{K}} x_k (L_{\max}^{\text{ad,I}} + L_k^{\text{O}}) \right] 2D^{\text{L,h}} H^{\text{L}}$. Therefore, the peak KV cache memory in the AD mode is given by

$$q^{\text{ad}} = \max \{q^{\text{ad,p}}, q^{\text{ad,a}}\} = q^{\text{ad,a}}. \quad (15)$$

3) *Memory for the SD mode:* In the SD mode, the required KV cache memory for prefilling is given by $q^{\text{sd,p}} = q^{\text{sd,L,p}} + q^{\text{sd,S,p}}$, where $q^{\text{sd,L,p}}$ and $q^{\text{sd,S,p}}$ denote the KV cache memory required by the LLM and SLM during prefilling, respectively.

Following $q^{\text{ad,p}}$, $q^{\text{sd,L,p}} = M^{\text{L}} \theta \left(\sum_{k \in \mathcal{K}} y_k L_{\max}^{\text{sd,I}} \right) 2D^{\text{L,h}} H^{\text{L}}$ and $q^{\text{sd,S,p}} = M^{\text{S}} \theta \left(\sum_{k \in \mathcal{K}} y_k L_{\max}^{\text{sd,I}} \right) 2D^{\text{S,h}} H^{\text{S}}$. Following $q^{\text{ad,a}}$, the peak SLM KV cache memory during draft generation is given by $q^{\text{sd,a}} = M^{\text{S}} \theta \left[\sum_{k \in \mathcal{K}} y_k (L_{N_k}^{\text{sd,P}} + L^{\text{D}}) \right] 2D^{\text{S,h}} H^{\text{S}}$. During verification, the peak LLM KV cache is given by $q^{\text{sd,v}} = M^{\text{L}} \theta \left[\sum_{k \in \mathcal{K}} y_k (L_{N_k}^{\text{sd,P}} + L^{\text{D}} + 1) \right] 2D^{\text{L,h}} H^{\text{L}}$. Therefore, the peak KV cache memory in the SD mode is

$$q^{\text{sd}} = \max \{q^{\text{sd,p}}, q^{\text{sd,a}} + q^{\text{sd,v}}\} = q^{\text{sd,a}} + q^{\text{sd,v}}. \quad (16)$$

F. User E2E latency

To model the inference latency under shared GPU computing resources, we consider the computing resource allocation between the AD and SD modes. Let z denote the fraction of GPU computing resources allocated to the AD mode, and thus $1 - z$ is allocated to the SD mode³.

1) *Latency of the AD mode*: Following previous studies [39], the LLM prefilling time in the AD mode is $\tau^{\text{ad,p}} = \frac{\alpha^{\text{L}} W^{\text{ad,p}}}{z} + \beta^{\text{L}}$, where α^{L} and β^{L} are model- and hardware-dependent constants measured offline. The time of decoding step l in the AD mode is $\tau_l^{\text{ad,a}} = \frac{\alpha^{\text{L}} W_{n,l}^{\text{ad,a}}}{z} + \beta^{\text{L}}$. Thus, the time from prefilling to the completion of step l in the AD mode is

$$\tau_l^{\text{ad}} = \tau^{\text{ad,p}} + \sum_{l'=1}^l \tau_{l'}^{\text{ad,a}}. \quad (17)$$

2) *Latency of the SD mode*: The prefilling times of the LLM and the SLM in the SD mode are given by $\tau^{\text{sd,L,p}} = \frac{\alpha^{\text{L}} W^{\text{sd,L,p}}}{1-z} + \beta^{\text{L}}$ and $\tau^{\text{sd,S,p}} = \frac{\alpha^{\text{S}} W^{\text{sd,S,p}}}{1-z} + \beta^{\text{S}}$, respectively. For iteration n , the time required by the SLM to generate L^{D} draft tokens for all users scheduled to the SD mode is

$$\tau_n^{\text{sd,a}} = \begin{cases} \sum_{l=1}^{L^{\text{D}}-1} \left(\frac{\alpha^{\text{S}} W_{n,l}^{\text{sd,a}}}{1-z} + \beta^{\text{S}} \right), & n = 1, \\ \sum_{l=1}^{L^{\text{D}}} \left(\frac{\alpha^{\text{S}} W_{n,l}^{\text{sd,a}}}{1-z} + \beta^{\text{S}} \right), & 2 \leq n \leq N_{\text{max}}^{\text{sd}}, \end{cases} \quad (18)$$

where the first iteration requires only $L^{\text{D}} - 1$ steps due to the prefilling. Moreover, the LLM verification time in iteration n is denoted by $\tau_n^{\text{sd,v}} = \frac{\alpha^{\text{L}} W_n^{\text{sd,v}}}{1-z} + \beta^{\text{L}}$. Thus, the time from prefilling to the completion of iteration n in the SD mode is

$$\tau_n^{\text{sd}} = \tau^{\text{sd,L,p}} + \tau^{\text{sd,S,p}} + \sum_{n'=1}^n \left(\tau_{n'}^{\text{sd,a}} + \tau_{n'}^{\text{sd,v}} \right). \quad (19)$$

3) *E2E latency*: Based on (1), (2), (3), (17), and (19), the E2E latency of user k is expressed as

$$t_k^{\text{E}} = x_k \left(\tau^{\text{ad,p}} + \tau_{L_{\text{max}}^{\text{ad}}-1}^{\text{ad,a}} \right) + y_k \left(\tau^{\text{sd,p}} + \tau_{N_{\text{max}}^{\text{sd}}}^{\text{sd,a}} \right) + \tau_k^{\text{down}}. \quad (20)$$

G. Problem Formulation

To maximize the task throughput of BALANCE, i.e., the number of served users satisfying their latency requirements, we formulate the following optimization problem.

$$\mathcal{P1} : \max_{\mathbf{X}, \mathbf{Y}, z} r(\mathbf{X} + \mathbf{Y}) = \sum_{k \in \mathcal{K}} x_k + y_k \quad (21a)$$

$$\text{s.t. } x_k + y_k \leq 1, \quad \forall k \in \mathcal{K}, \quad (21b)$$

$$t_k^{\text{E}} \leq T_k, \quad \forall k \in \mathcal{K}, \quad (21c)$$

$$\mathbb{I}_{\{z < 1\}} q^{\text{S}} + q^{\text{L}} + q^{\text{ad}} + q^{\text{sd}} \leq Q, \quad (21d)$$

$$x_k \in \{0, 1\}, \quad \forall k \in \mathcal{K}, \quad (21e)$$

³In practice, GPU computing resource allocation can be implemented by partitioning GPU resources, e.g., streaming multiprocessors and GPU engines, across multiple tasks, which has already been supported by NVIDIA [37]. For simplicity, we refer to it as ‘‘GPU computing resource allocation’’ in this paper. Such allocation has been adopted in existing studies [38].

$$y_k \in \{0, 1\}, \quad \forall k \in \mathcal{K}, \quad (21f)$$

$$z \in \mathbf{Z}. \quad (21g)$$

Here, $x_k \in \mathbf{X}$, $y_k \in \mathbf{Y}$, and z are the decision variables for AD-mode user scheduling, SD-mode user scheduling, and GPU computing resource allocation, respectively. In (21a), $\mathbf{X} + \mathbf{Y}$ denotes the element-wise sum of $\mathbf{X}$ and $\mathbf{Y}$. Constraint (21b) assigns each user to at most one inference mode. Constraints (21c) and (21d) enforce the user latency and GPU memory capacity constraints, respectively. Here, Q is the available GPU memory capacity, and the indicator function $\mathbb{I}_{\{z < 1\}}$ equals 1 if $z < 1$ and 0 otherwise. When $z = 1$, the SD mode is inactive, so the SLM weights are not loaded into GPU memory. Constraints (21e) and (21f) are binary constraints. Constraint (21g) restricts z to $\mathbf{Z} = \{0, z_1, \dots, z_I, 1\}$, where $0 < z_1 < \dots < z_I < 1$. The values in $\mathbf{Z}$ represent the GPU computing resource allocation ratios supported by the system.

IV. ALGORITHM DESIGN

This section proves the NP-hardness of $\mathcal{P1}$ and develops a polynomial-time constant-approximation algorithm.

A. Problem Mapping

The NP-hardness of $\mathcal{P1}$ is stated below.

Proposition 1. $\mathcal{P1}$ is an NP-hard problem.

Proof. The proof sketch is as follows. Consider a special case of $\mathcal{P1}$ with fixed z . The problem then reduces to user scheduling under latency requirements and memory resource constraints, which can be mapped to a multi-dimensional multiple-choice knapsack problem (MMKP). Since the MMKP is NP-hard [40], [41], $\mathcal{P1}$ is NP-hard, completing the proof. $\square$

B. Equivalent Problem Transformation

To facilitate the solution design, we introduce an auxiliary variable λ to represent the fraction of available GPU memory allocated to the AD mode. Since the memory consumption of the AD and SD modes can be characterized separately, $\mathcal{P1}$ is equivalently transformed into the following $\mathcal{P2}$.

$$\mathcal{P2} : \max_{\mathbf{X}, \mathbf{Y}, z, \lambda} r(\mathbf{X} + \mathbf{Y}) = \sum_{k \in \mathcal{K}} x_k + y_k \quad (22a)$$

$$\text{s.t. } (21b), (21c), (21e), (21f), (21g), \quad (22b)$$

$$q^{\text{ad}} \leq Q_1, \quad (22c)$$

$$q^{\text{sd}} \leq Q_2, \quad (22d)$$

$$\lambda \in \Lambda. \quad (22e)$$

Here, $Q_1 = \lambda(Q - q^{\text{L}} - \mathbb{I}_{\{z < 1\}} q^{\text{S}})$ and $Q_2 = (1 - \lambda)(Q - q^{\text{L}} - \mathbb{I}_{\{z < 1\}} q^{\text{S}})$ denote the memory reserved for the AD and SD modes, respectively. Constraint (22e) restricts λ to $\Lambda = \{0, \lambda_1, \dots, \lambda_J, 1\}$, where $0 < \lambda_1 < \dots < \lambda_J < 1$. The set Λ contains all supported ratios for partitioning GPU memory between the AD and SD modes, ensuring that every feasible memory split can be represented by some $\lambda \in \Lambda$.

Next, we state that solving $\mathcal{P2}$ is equivalent to solving $\mathcal{P1}$.

Proposition 2. Solving $\mathcal{P}1$ is equivalent to solving $\mathcal{P}2$.

Proof. After reserving memory for the model weights, Q_1 and Q_2 can be obtained according to λ . By the construction of Λ , any feasible solution to $\mathcal{P}1$ corresponds to a feasible solution to $\mathcal{P}2$. Conversely, any feasible solution to $\mathcal{P}2$ satisfies the memory constraint in $\mathcal{P}1$ since Q_1 and Q_2 are given. Thus, $\mathcal{P}1$ and $\mathcal{P}2$ are equivalent, completing the proof. $\square$

To solve $\mathcal{P}2$, we fix z and λ , and have the below proposition.

Proposition 3. Given z and λ , solving $\mathcal{P}2$ is equivalent to solving $\mathcal{P}3$.

$$\mathcal{P}3 : \max_{\mathbf{X}, \mathbf{Y}} r(\mathbf{X} + \mathbf{Y}) = \sum_{k \in \mathcal{K}} x_k + y_k \quad (23a)$$

$$\text{s.t. (21b), (21c), (21e), (21f), (22c), (22d).} \quad (23b)$$

Proof. For fixed z and λ , constraints (21g) and (22e) can be removed, reducing $\mathcal{P}2$ to $\mathcal{P}3$ and completing the proof. $\square$

C. Problem Decomposition

Given the NP-hardness of $\mathcal{P}3$, we decouple $\mathcal{P}3$ into the following two sub-problems, $\mathcal{P}3.1$ and $\mathcal{P}3.2$, which schedule users in the AD and SD modes, respectively. We have

$$\mathcal{P}3.1 : \max_{\mathbf{X}} r(\mathbf{X}) = \sum_{k \in \mathcal{K}} x_k \quad (24a)$$

$$\text{s.t. (21e), (22c)} \quad (24b)$$

$$x_k t^{\text{ad}} \leq T'_k, \forall k \in \mathcal{K}, \quad (24c)$$

where $t^{\text{ad}} = \tau^{\text{ad,up}} + \tau_{L_{\text{max}}^{\text{ad,O}} - 1}^{\text{ad}}$ denotes the latency of the AD mode excluding output downloading, and $T'_k = T_k - \Delta\tau_k^{\text{down}}$ denotes the remaining latency budget of user k after reserving the output downloading latency. Also, we have

$$\mathcal{P}3.2 : \max_{\mathbf{Y}} r'(\mathbf{Y}) = \sum_{k \in \mathcal{K} \setminus \mathcal{K}^{\text{ad}}} y_k \quad (25a)$$

$$\text{s.t. (22d),} \quad (25b)$$

$$y_k t^{\text{sd}} \leq T'_k, \forall k \in \mathcal{K} \setminus \mathcal{K}^{\text{ad}}, \quad (25c)$$

$$y_k \in \{0, 1\}, \forall k \in \mathcal{K} \setminus \mathcal{K}^{\text{ad}}, \quad (25d)$$

where $t^{\text{sd}} = \tau^{\text{sd,up}} + \tau_{N_{\text{max}}^{\text{sd}}}^{\text{sd}}$ denotes the latency of the SD mode excluding output downloading, and $\mathcal{K}^{\text{ad}} = \{k \mid x_k = 1\}$ is the set of users served by the AD mode.

The following subsections will sequentially solve $\mathcal{P}3.1$ and $\mathcal{P}3.2$ to determine the optimal user scheduling for the AD and SD modes, respectively. The performance gap between the sequential solution and the optimal solution to $\mathcal{P}1$ will be analyzed in Section IV-F.

D. Solution to $\mathcal{P}3.1$: User Scheduling for the AD Mode

We obtain an optimal solution to $\mathcal{P}3.1$ by enumerating the feasible values of $L_{\text{max}}^{\text{ad,I}}$, $L_{\text{max}}^{\text{ad,O}}$, t^{ad} , and $\tau^{\text{ad,up}}$, and identifying the served users under each setting. When $z = 0$ or $\lambda = 0$, the AD mode is inactive, and thus the optimal solution to $\mathcal{P}3.1$ is $\hat{\mathbf{X}} = \mathbf{0}$. The following analysis considers $z > 0$ and $\lambda > 0$.

First, given $L_{\text{max}}^{\text{ad,I}} = L_1$, $L_{\text{max}}^{\text{ad,O}} = L_2$, $t^{\text{ad}} = t_1$, and $\tau^{\text{ad,up}} = \tau_1$, the candidate user set for the AD mode is $\mathcal{K}_1 = \{k \mid k \in \mathcal{K}, L_k^{\text{I}} \leq L_1, L_k^{\text{O}} \leq L_2, t_1 \leq T'_k, \Delta\tau_k^{\text{up}} \leq \tau_1\}$. For each candidate user $k \in \mathcal{K}_1$, its latency contribution is

$$\tilde{t}_k^{\text{ad}} = \frac{\alpha^L \tilde{W}_k^{\text{ad,p}}}{z} + \sum_{l=1}^{L_2-1} \frac{\alpha^L \tilde{W}_{l,k}^{\text{ad,a}}}{z}. \quad (26)$$

Here, $\tilde{W}_k^{\text{ad,p}}$ and $\tilde{W}_{l,k}^{\text{ad,a}}$ denote the contributions of user k to the LLM workloads during prefilling and at the l -th decoding step in the AD mode, respectively, where $\tilde{W}_k^{\text{ad,p}} = \Gamma^{\text{L,p}}(L_1)$ and $\tilde{W}_{l,k}^{\text{ad,a}} = \mathbb{I}_{\{l \leq L_k^{\text{O}} - 1\}} \Gamma_l^{\text{L,a}}(L_1)$. Moreover, the contribution of user $k \in \mathcal{K}_1$ to the total memory consumption of the AD mode is given by

$$\tilde{q}_k^{\text{ad}} = \tilde{q}^{\text{ad}} = M^{\text{L}} \theta (L_1 + L_2) 2D^{\text{L,h}} H^{\text{L}}. \quad (27)$$

Second, let $\Pi_1 = \{1, \dots, d_1, \dots, |\mathcal{K}_1|\}$ denote a permutation of the users in $\mathcal{K}_1$ sorted in a non-decreasing order of $\tilde{t}_k^{\text{ad}}$, i.e., $\tilde{t}_{\Pi_1(d_1)}^{\text{ad}} \leq \tilde{t}_{\Pi_1(d_1+1)}^{\text{ad}}$. Therefore, the number of served users under L_1 , L_2 , t_1 , and τ_1 is given by

$$d_1^* = \max \left\{ d_1 \mid d_1 \leq \lfloor \frac{Q_1}{\tilde{q}^{\text{ad}}} \rfloor, \sum_{d'_1=1}^{d_1} \tilde{t}_{\Pi_1(d'_1)}^{\text{ad}} \leq \Delta t_1 \right\}, \quad (28)$$

where $\Delta t_1 = t_1 - \tau_1 - \Delta\tau_1$ and $\Delta\tau_1 = L_2 \beta^{\text{L}}$ denotes the fixed latency term of one forward pass in prefilling and $L_2 - 1$ forward passes in decoding. Based on (28), the served user set for the AD mode under L_1 , L_2 , t_1 , and τ_1 is $\hat{\mathcal{K}}^{\text{ad}} = \{\Pi_1(1), \dots, \Pi_1(d_1^*)\}$, and the corresponding user scheduling decision $\hat{x}_k \in \hat{\mathbf{X}}$ is

$$\hat{x}_k = \begin{cases} 1, & \text{if } k \in \hat{\mathcal{K}}^{\text{ad}}, \\ 0, & \text{otherwise.} \end{cases} \quad (29)$$

By enumerating all feasible L_1 , L_2 , t_1 , and τ_1 , we can obtain the maximum number of served users and the corresponding user scheduling decision under given z and λ for $\mathcal{P}3.1$. Algorithm 1 summarizes the above procedures, and its optimality is established in the following proposition.

Proposition 4. Algorithm 1 obtains an optimal solution to $\mathcal{P}3.1$ under given z and λ .

Proof. The proof sketch is below. For fixed z and λ , every feasible solution to $\mathcal{P}3.1$ corresponds to a tuple (L_1, L_2, t_1, τ_1) . For each tuple, all candidate users in $\mathcal{K}_1$ have unit profit, identical memory contributions, and fixed latency contributions. Thus, (29) leads to the optimal user scheduling for this tuple. Since Algorithm 1 enumerates all feasible tuples, it obtains an optimal solution to $\mathcal{P}3.1$, completing the proof. $\square$

Algorithm 1: AD-Mode User Scheduling Algorithm

Input: z and λ .
Output: $\hat{\mathbf{X}}$ and $\hat{\mathcal{K}}^{\text{ad}}$.

- 1 **Initialize:** $r_1 = 0$, $\hat{\mathbf{X}} = \mathbf{0}$, $\hat{\mathcal{K}}^{\text{ad}} = \emptyset$.
- 2 **if** $z = 0$ **or** $\lambda = 0$ **then**
- 3 Return $\hat{\mathbf{X}} = \mathbf{0}$ and $\hat{\mathcal{K}}^{\text{ad}} = \emptyset$.
- 4 **end**
- 5 **for** $L_1 \in \{0\} \cup \{L_k^I | k \in \mathcal{K}\}$,
 $L_2 \in \{0\} \cup \{L_k^O | k \in \mathcal{K}\}$, $t_1 \in \{0\} \cup \{T_k' | k \in \mathcal{K}\}$ **do**
- 6 **for** $\tau_1 \in \{0\} \cup \{\Delta\tau_k^{\text{up}} | k \in \mathcal{K}\}$ **do**
- 7 Calculate d_1^* with (28).
- 8 **if** $d_1^* > r_1$ **then**
- 9 $r_1 = d_1^*$ and $\hat{\mathcal{K}}^{\text{ad}} = \{\Pi_1(1), \dots, \Pi_1(d_1^*)\}$.
- 10 **end**
- 11 **end**
- 12 **end**
- 13 Calculate $\hat{\mathbf{X}}$ with (29).

E. Solution to P3.2: User Scheduling for the SD Mode

We solve P3.2 similarly by enumerating the feasible values of $L_{\max}^{\text{sd,I}}$, $L_{\max}^{\text{sd,A}}$, $N_{\max}^{\text{sd}}$, t^{sd} , and $\tau^{\text{sd,up}}$, and identifying the served users under each setting. When $z = 1$ or $\lambda = 1$, the SD mode is inactive, and thus the optimal solution to P3.2 is $\hat{\mathbf{Y}} = \mathbf{0}$. The following analysis considers $z < 1$ and $\lambda < 1$.

First, given $L_{\max}^{\text{sd,I}} = L_3$, $L_{\max}^{\text{sd,A}} = L_4$, $N_{\max}^{\text{sd}} = \tilde{N}_1$, $t^{\text{sd}} = t_2$, and $\tau^{\text{sd,up}} = \tau_2$, the candidate user set for the SD mode is $\mathcal{K}_2 = \{k \mid k \in \mathcal{K}', L_k^I \leq L_3, L_k^A \leq L_4, N_k \leq \tilde{N}_1, t_2 \leq T_k', \Delta\tau_k^{\text{up}} \leq \tau_2\}$, where $\mathcal{K}' = \mathcal{K} \setminus \mathcal{K}^{\text{ad}}$. The contribution of each candidate user $k \in \mathcal{K}_2$ to the total latency of the SD mode is

$$\begin{aligned} \tilde{t}_k^{\text{sd}} = & \frac{\alpha^L \tilde{W}_k^{\text{sd,L,p}}}{1-z} + \frac{\alpha^S \tilde{W}_k^{\text{sd,S,p}}}{1-z} \\ & + \sum_{l=1}^{L^D-1} \frac{\alpha^S \tilde{W}_{l,l,k}^{\text{sd,a}}}{1-z} + \sum_{n=2}^{\tilde{N}_1} \sum_{l=1}^{L^D} \frac{\alpha^S \tilde{W}_{n,l,k}^{\text{sd,a}}}{1-z} + \sum_{n=1}^{\tilde{N}_1} \frac{\alpha^L \tilde{W}_{n,k}^{\text{sd,v}}}{1-z}. \end{aligned} \quad (30)$$

Here, $\tilde{W}_k^{\text{sd,L,p}}$ and $\tilde{W}_k^{\text{sd,S,p}}$ are the contributions of user k to the total LLM and SLM workloads during prefilling in the SD mode, respectively, where $\tilde{W}_k^{\text{sd,L,p}} = \Gamma^{\text{L,p}}(L_3)$ and $\tilde{W}_k^{\text{sd,S,p}} = \Gamma^{\text{S,p}}(L_3)$. Additionally, $\tilde{W}_{n,l,k}^{\text{sd,a}}$ and $\tilde{W}_{n,k}^{\text{sd,v}}$ denote the contributions of user k to the total SLM workload at the l -th decoding step of iteration n and the total LLM workload during verification in iteration n , respectively. Let $\tilde{L}_n^{\text{sd,P}}$ denote the prefix length of the retained tokens before draft generation in the n -th iteration under L_3 and L_4 , where $\tilde{L}_n^{\text{sd,P}} = L_3$ when $n = 1$, and $\tilde{L}_n^{\text{sd,P}} = L_3 + (n-2)(L_4+1) + L_4$ when $n \geq 2$. Thus, $\tilde{W}_{n,l,k}^{\text{sd,a}} = \mathbb{I}_{\{n \leq N_k\}} \Gamma_l^{\text{S,a}}(\tilde{L}_n^{\text{sd,P}})$, $\tilde{W}_{n,k}^{\text{sd,v}} = \mathbb{I}_{\{n \leq N_k\}} \Gamma_1^{\text{L,v}}(\tilde{L}_1^{\text{sd,P}}, L^D)$ when $n = 1$, and $\tilde{W}_{n,k}^{\text{sd,v}} = \mathbb{I}_{\{n \leq N_k\}} \Gamma_n^{\text{L,v}}(\tilde{L}_n^{\text{sd,P}}, L^D + 1)$ when $n \geq 2$. Moreover, the contribution of user k to the total memory consumption of the SD mode is given by

$$\tilde{q}_k^{\text{sd}} = \tilde{q}_k^{\text{sd,a}} + \tilde{q}_k^{\text{sd,v}}. \quad (31)$$

Here, $\tilde{q}_k^{\text{sd,a}} = M^{\text{S}\theta}(\tilde{L}_{N_k}^{\text{sd,P}} + L^D) 2D^{\text{S,h}}H^{\text{S}}$ denotes the contri-

Algorithm 2: SD-Mode User Scheduling Algorithm

Input: z , λ , and $\hat{\mathcal{K}}^{\text{ad}}$.
Output: $\hat{\mathbf{Y}}$ and $\hat{\mathcal{K}}^{\text{sd}}$.

- 1 **Initialize:** $\mathcal{K}' = \mathcal{K} \setminus \hat{\mathcal{K}}^{\text{ad}}$, $r_2 = 0$, $\hat{\mathbf{Y}} = \mathbf{0}$, $\hat{\mathcal{K}}^{\text{sd}} = \emptyset$.
- 2 **if** $z = 1$ **or** $\lambda = 1$ **then**
- 3 Return $\hat{\mathbf{Y}} = \mathbf{0}$ and $\hat{\mathcal{K}}^{\text{sd}} = \emptyset$.
- 4 **end**
- 5 **for** $L_3 \in \{0\} \cup \{L_k^I | k \in \mathcal{K}'\}$,
 $L_4 \in \{0\} \cup \{L_k^A | k \in \mathcal{K}'\}$, $\tilde{N}_1 \in \{0\} \cup \{N_k | k \in \mathcal{K}'\}$,
 $t_2 \in \{0\} \cup \{T_k' | k \in \mathcal{K}'\}$ **do**
- 6 **for** $\tau_2 \in \{0\} \cup \{\Delta\tau_k^{\text{up}} | k \in \mathcal{K}'\}$ **do**
- 7 Calculate d_2^* with (32).
- 8 **if** $d_2^* > r_2$ **then**
- 9 $r_2 = d_2^*$ and $\hat{\mathcal{K}}^{\text{sd}} = \{\Pi_2(1), \dots, \Pi_2(d_2^*)\}$.
- 10 **end**
- 11 **end**
- 12 **end**
- 13 Calculate $\hat{\mathbf{Y}}$ with (33).

bution of user k to the total memory consumption during SLM draft generation. $\tilde{q}_k^{\text{sd,v}}$ denotes the contribution of user k to the total memory consumption during LLM draft verification, where $\tilde{q}_k^{\text{sd,v}} = M^{\text{L}\theta}(\tilde{L}_{N_k}^{\text{sd,P}} + L^D) 2D^{\text{L,h}}H^{\text{L}}$ when $N_k = 1$, and $\tilde{q}_k^{\text{sd,v}} = M^{\text{L}\theta}(\tilde{L}_{N_k}^{\text{sd,P}} + L^D + 1) 2D^{\text{L,h}}H^{\text{L}}$ when $N_k \geq 2$.

Second, let $\Pi_2 = \{1, \dots, d_2, \dots, |\mathcal{K}_2|\}$ denote a permutation of the users in $\mathcal{K}_2$ sorted in a non-decreasing order of $\tilde{t}_k^{\text{sd}}$, i.e., $\tilde{t}_{\Pi_2(d_2)}^{\text{sd}} \leq \tilde{t}_{\Pi_2(d_2+1)}^{\text{sd}}$. Since both $\tilde{t}_k^{\text{sd}}$ and $\tilde{q}_k^{\text{sd}}$ are monotonically non-decreasing with N_k , sorting users by $\tilde{t}_k^{\text{sd}}$ also preserves the non-decreasing order of $\tilde{q}_k^{\text{sd}}$. Therefore, the number of served users under L_3 , L_4 , $\tilde{N}_1$, t_2 , and τ_2 is

$$d_2^* = \max \left\{ d_2 \mid \sum_{d'_2=1}^{d_2} \tilde{q}_{\Pi_2(d'_2)}^{\text{sd}} \leq Q_2, \sum_{d'_2=1}^{d_2} \tilde{t}_{\Pi_2(d'_2)}^{\text{sd}} \leq \Delta t_2 \right\}, \quad (32)$$

where $\Delta t_2 = t_2 - \tau_2 - \Delta\tau_2$ and $\Delta\tau_2 = (\tilde{N}_1 + 1)\beta^{\text{L}} + \tilde{N}_1 L^D \beta^{\text{S}}$ denotes the fixed latency term in the SD mode. Based on (32), the served user set of the SD mode under L_3 , L_4 , $\tilde{N}_1$, t_2 , and τ_2 is $\hat{\mathcal{K}}^{\text{sd}} = \{\Pi_2(1), \dots, \Pi_2(d_2^*)\}$, and the corresponding user scheduling decision $\hat{\mathbf{y}}_k \in \hat{\mathbf{Y}}$ is

$$\hat{\mathbf{y}}_k = \begin{cases} 1, & \text{if } k \in \hat{\mathcal{K}}^{\text{sd}}, \\ 0, & \text{otherwise.} \end{cases} \quad (33)$$

By enumerating all feasible L_3 , L_4 , $\tilde{N}_1$, t_2 , and τ_2 , we can identify the maximum number of served users and the corresponding user scheduling decision under given z and λ for P3.2. Algorithm 2 summarizes the above procedures, and its optimality is established in the following proposition.

Proposition 5. Algorithm 2 obtains an optimal solution to P3.2 under given z , λ , and $\hat{\mathcal{K}}^{\text{ad}}$.

Proof. The proof is similar to that of Proposition 4. Since $\tilde{t}_k^{\text{sd}}$ and $\tilde{q}_k^{\text{sd}}$ are non-decreasing in N_k , (33) maximizes the number of served users for each tuple. Enumerating all feasible tuples yields an optimal solution to P3.2, completing the proof. $\square$

Algorithm 3: BALANCE User Scheduling Algorithm

Input: $\mathcal{K}$, Q , and T_k .

Output: $\hat{\mathbf{X}}$, $\hat{\mathbf{Y}}$, $\hat{z}$, and $\hat{\lambda}$.

```

1 Initialize:  $r_3 = 0$ ,  $\hat{\mathbf{X}} = \mathbf{0}$ ,  $\hat{\mathbf{Y}} = \mathbf{0}$ ,  $\hat{z} = 0$ ,  $\hat{\lambda} = 0$ .
2 for  $z \in \mathbf{Z}$ ,  $\lambda \in \Lambda$  do
3   Obtain  $\hat{\mathbf{X}}$  and  $\hat{\mathcal{K}}^{\text{ad}}$  using Algorithm 1 with  $z$  and  $\lambda$ .
4   Obtain  $\hat{\mathbf{Y}}$  and  $\hat{\mathcal{K}}^{\text{sd}}$  using Algorithm 2 with  $z$ ,  $\lambda$ ,
      and  $\hat{\mathcal{K}}^{\text{ad}}$ .
5   if  $|\hat{\mathcal{K}}^{\text{ad}}| + |\hat{\mathcal{K}}^{\text{sd}}| > r_3$  then
6      $r_3 = |\hat{\mathcal{K}}^{\text{ad}}| + |\hat{\mathcal{K}}^{\text{sd}}|$ ,  $\hat{\mathbf{X}} = \hat{\mathbf{X}}$ ,  $\hat{\mathbf{Y}} = \hat{\mathbf{Y}}$ ,  $\hat{z} = z$ ,
      and  $\hat{\lambda} = \lambda$ .
7   end
8 end

```

F. Algorithm Outline

Algorithm 3 summarizes the overall procedure for solving $\mathcal{P}1$. It enumerates all feasible pairs of (z, λ) . For each pair, it first solves $\mathcal{P}3.1$ using Algorithm 1 to schedule users for the AD mode, and then solves $\mathcal{P}3.2$ using Algorithm 2 to schedule the remaining users for the SD mode. The solution that serves the maximum number of users over all pairs gives the user scheduling decision $\hat{\mathbf{X}}$ for the AD mode, user scheduling decision $\hat{\mathbf{Y}}$ for the SD mode, and GPU computing resource allocation $\hat{z}$. The approximation guarantee of Algorithm 3 is characterized in the following theorem.

Theorem 1. *Algorithm 3 achieves a constant approximation guarantee. Specifically, the produced solution $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ satisfy $r(\hat{\mathbf{X}} + \hat{\mathbf{Y}}) \geq \frac{1}{2}r(\mathbf{X}^* + \mathbf{Y}^*)$, where $\mathbf{X}^*$ and $\mathbf{Y}^*$ denote the optimal user scheduling decisions of $\mathcal{P}1$.*

Proof. The proof sketch is as follows. Since $\mathbf{X}$ and $\mathbf{Y}$ are disjoint, we have $r(\mathbf{X} + \mathbf{Y}) = r(\mathbf{X}) + r(\mathbf{Y})$ and $r'(\mathbf{Y}) = r(\mathbf{Y})$. Let z^* and λ^* denote the computing and memory resource allocations in an optimal solution. Since Algorithm 3 enumerates all feasible z and λ , it also examines z^* and λ^* , and thus $r(\hat{\mathbf{X}} + \hat{\mathbf{Y}}) = r(\hat{\mathbf{X}} + \hat{\mathbf{Y}} \mid \lambda = \hat{\lambda}, z = \hat{z}) \geq r(\hat{\mathbf{X}} + \hat{\mathbf{Y}} \mid \lambda = \lambda^*, z = z^*)$. By Proposition 4, $r(\hat{\mathbf{X}} \mid \lambda = \lambda^*, z = z^*) \geq r(\mathbf{X}^*)$. We decompose $\mathbf{Y}^*$ into disjoint $\mathbf{Y}_1^*$ and $\mathbf{Y}_2^*$, where $\mathbf{Y}_1^*$ includes the users in $\mathbf{Y}^*$ that are also selected by $\hat{\mathcal{K}}^{\text{ad}}|_{\lambda=\lambda^*, z=z^*}$, and $\mathbf{Y}_2^*$ includes the remaining users. By Proposition 5, $r(\hat{\mathbf{Y}} \mid \lambda = \lambda^*, z = z^*) \geq r(\mathbf{Y}_2^*)$. Since $\mathbf{Y}_1^*$ only contains users already selected by $\hat{\mathbf{X}}$, we have $r(\mathbf{Y}_1^*) \leq r(\hat{\mathbf{X}} \mid \lambda = \lambda^*, z = z^*)$. Thus, $r(\mathbf{Y}^*) \leq r(\hat{\mathbf{X}} \mid \lambda = \lambda^*, z = z^*) + r(\hat{\mathbf{Y}} \mid \lambda = \lambda^*, z = z^*)$. Then, we have $r(\mathbf{X}^* + \mathbf{Y}^*) \leq 2r(\hat{\mathbf{X}} + \hat{\mathbf{Y}} \mid \lambda = \lambda^*, z = z^*) \leq 2r(\hat{\mathbf{X}} + \hat{\mathbf{Y}})$, completing the proof. $\square$

The time complexity of Algorithm 3 is stated below.

Theorem 2. *Algorithm 3 has a polynomial time complexity of $O(K^6 \log K)$, where K is the total number of users in $\mathcal{K}$.*

Proof. The proof sketch is as follows. Algorithm 1 enumerates $O(K^4)$ tuples (L_1, L_2, t_1, τ_1) . For each tuple, constructing Π_1 and computing d_1^* in Line 7 require $O(K \log K)$ time. Therefore, the time complexity of Algorithm 1 is $O(K^5 \log K)$. Similarly, the time complexity of Algorithm 2 is $O(K^6 \log K)$. Algorithm 3 enumerates $O(|\mathbf{Z}| |\Lambda|)$ pairs (z, λ) , where $|\mathbf{Z}|$ and $|\Lambda|$ denote the cardinalities of $\mathbf{Z}$ and Λ , respectively. For each pair, Lines 3 and 4 require $O(K^5 \log K)$ and $O(K^6 \log K)$ time, respectively. Since $|\mathbf{Z}|$ and $|\Lambda|$ are constants, Algorithm 3 has time complexity $O(K^6 \log K)$, completing the proof. $\square$

V. NUMERICAL RESULTS

This section evaluates the performance of BALANCE under different system settings, compares the algorithm running time, and conducts ablation studies.

A. Experimental Setup

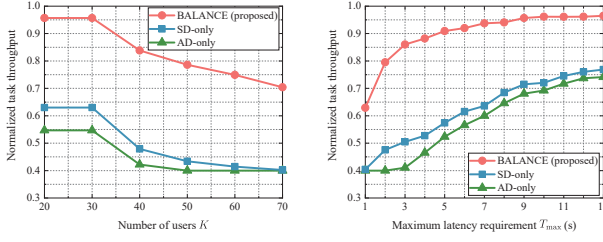
We adopt a hybrid testing method that combines device measurements and numerical simulations to evaluate the performance of the proposed BALANCE framework. Specifically, computing latencies are measured on a real server, while the communication latencies in multi-user scenarios are obtained through simulations. The inference is performed on an edge server using an AMD Ryzen Threadripper PRO 5975WX CPU, an NVIDIA GeForce RTX 4090 GPU with 24 GB of GPU memory, and 256 GB of DDR4 RAM. TinyLlama-1.1B and Llama-2-7B are used as the SLM and LLM, respectively, with FP16 precision. Accordingly, $Q = 24$ GB.

The coverage radius of the edge server is set to 250 m. The uplink and downlink bandwidths allocated to user k are set to $B_k = 1$ MHz and $\bar{B}_k = 2$ MHz, respectively. The transmit power spectral densities of the users for uplink transmission and of the edge server for downlink transmission are set to -50 dBm/Hz and -29 dBm/Hz, respectively. The number of users K ranges from 20 to 70. The user latency requirement satisfies $T_k \in [T_{\min}, T_{\max}]$, ranging from 0.5 s to 13 s. The input length L_k^I and output length L_k^O are set within $[L_{\min}^I, L_{\max}^I]$ and $[L_{\min}^O, L_{\max}^O]$, respectively, both ranging from 1 to 1000 tokens. For SD, the draft token length L^D is set to 1, and the acceptance rate A_k is set within $[0.6, 0.8]$.

We compare BALANCE with the following baselines.

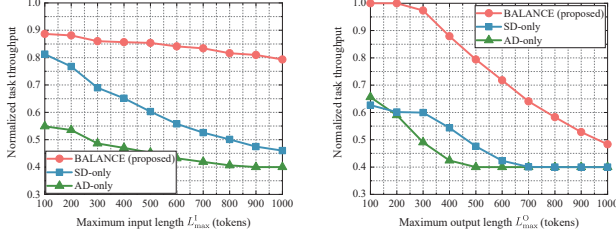
- **AD-only** [17]. In this baseline, all users are served only by the AD mode, and user scheduling is determined by Algorithm 1. By Proposition 4, this baseline achieves the maximum task throughput under the AD-only mode.
- **SD-only** [20]. In this baseline, all users are served only by the SD mode. User scheduling is determined by Algorithm 2. By Proposition 5, this baseline achieves the maximum task throughput under the SD-only mode.
- **Exhaustive search.** This baseline solves the original problem $\mathcal{P}1$ by exhaustively searching all feasible solutions, thereby incurring exponential time complexity. We use this baseline as an exact solution method to evaluate the running time efficiency of the proposed algorithm.

To evaluate system performance, we use normalized task throughput, defined as the ratio of the number of served users to the total number of users.



(a) Results vs. number of users K . (b) Results vs. maximum latency requirement $T_{\max}$.

Fig. 4. Performance of BALANCE under varying K and $T_{\max}$. The default values of K , $T_{\min}$, $T_{\max}$ are set to 30, 0.5, and 4, respectively. $L_{\min}^I$, $L_{\max}^I$, $L_{\min}^O$, and $L_{\max}^O$ are set to 1, 300, 1, and 100, respectively.



(a) Results vs. maximum input length $L_{\max}^I$. (b) Results vs. maximum output length $L_{\max}^O$.

Fig. 5. Performance of BALANCE under varying $L_{\max}^I$ and $L_{\max}^O$. The default values of $L_{\min}^I$, $L_{\max}^I$, $L_{\min}^O$, $L_{\max}^O$ are set to 1, 1000, 1, and 500, respectively. $T_{\min}$ and $T_{\max}$ are set to 2 s and 12 s, respectively. The default value of K is same as Fig. 4.

B. Performance Evaluation

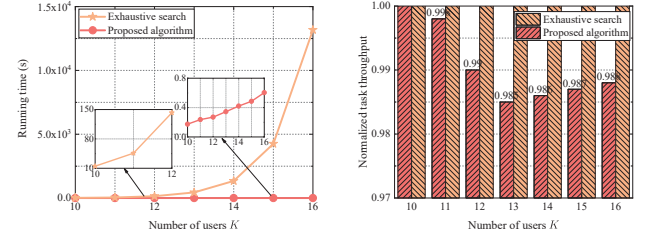
Figs. 4 and 5 evaluate the normalized task throughput of BALANCE under varying system settings. BALANCE consistently outperforms both AD-only and SD-only baselines. In Fig. 4, the throughput decreases with the number of users and increases under looser latency requirements. On average, BALANCE improves throughput over AD-only and SD-only baselines by 37.9% and 33.4%, respectively, under varying K , and by 31.5% and 27.3%, respectively, under varying $T_{\max}$. In Fig. 5, BALANCE achieves average gains over AD-only and SD-only baselines of 38.8% and 23.9%, respectively, under varying $L_{\max}^I$, and 30.4% and 27.3%, respectively, under varying $L_{\max}^O$.

C. Algorithm Running Time Comparisons

Fig. 6 compares the algorithm running time and task throughput of the proposed algorithm with exhaustive search under varying K . To run the exhaustive search efficiently, K is varied from 10 to 16. As shown in Fig. 6(a), the running time of the proposed algorithm grows nearly linearly with K , while exhaustive search increases nearly exponentially. The proposed algorithm achieves an average acceleration of about 5,100 times and up to 21,800 times at $K = 16$. Moreover, Fig. 6(b) shows that the proposed algorithm achieves near-optimal task throughput, with only about 0.9% average degradation.

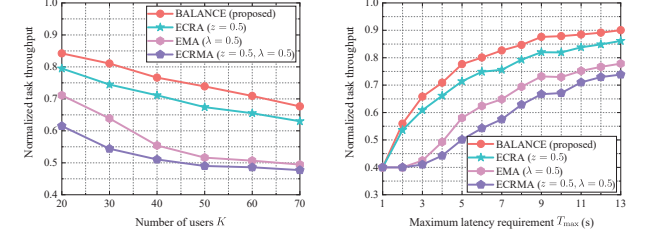
D. Ablation Study

We conduct ablation studies to evaluate the contribution of each component in the proposed algorithm. We compare the proposed algorithm with three variants, including equal computing resource allocation (ECRA), equal memory allocation



(a) Running time vs. number of users K . (b) Normalized task throughput vs. number of users K .

Fig. 6. Comparisons of algorithm running time and task throughput between the proposed algorithm and the exhaustive search under varying K . The system parameters are the same as those in Fig. 4(a).



(a) Ablation results vs. number of users K . (b) Ablation results vs. maximum latency requirement $T_{\max}$.

Fig. 7. Ablation study results for the proposed algorithm. The system parameters are the same as those in Fig. 4.

(EMA), and equal computing resource and memory allocation (ECRMA). In ECRA, $z = 0.5$, while λ and user scheduling are optimized using Algorithm 3. In EMA, $\lambda = 0.5$, while z and user scheduling are optimized using Algorithm 3. In ECRMA, both z and λ are 0.5, and only user scheduling is optimized using Algorithm 3. Fig. 7 illustrates the performance gains of the proposed algorithm over ECRA, EMA, and ECRMA under varying K and $T_{\max}$. Specifically, in Fig. 7(a), the proposed algorithm improves the task throughput by 5.6%, 18.7%, and 23.7% on average compared with ECRA, EMA, and ECRMA, respectively. The corresponding average performance improvements are 9.0%, 4.9%, and 18.4% in Fig. 7(b).

VI. CONCLUSION

In this paper, we investigated a hybrid autoregressive-speculative inference (BALANCE) framework for edge LLM inference to balance the latency-memory tradeoff between conventional autoregressive decoding (AD) and speculative decoding (SD). In BALANCE, an edge server hosts both a small language model and a large language model, and simultaneously supports AD and SD for different users with heterogeneous demands. To maximize the number of served users, we formulated a joint user scheduling and computing resource allocation problem under end-to-end latency requirements and server memory constraints. Since this problem is an NP-hard problem, we transformed it into two sub-problems and developed a polynomial-time algorithm with a constant approximation guarantee. The experiment results showed that BALANCE consistently outperforms the conventional AD and SD schemes and achieves significant improvements in task throughput. Overall, BALANCE provides an effective and practical solution for resource-constrained edge LLM inference systems.

REFERENCES

- [1] Z. Chen, B. Zhu, J. Wang, H. Shin, A. Nallanathan, and D. T. Niyato, "Network edge inference for large language models: Principles, techniques, and opportunities," *ACM Comput. Surv.*, vol. 58, no. 12, pp. 1–35, May 2026.
- [2] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile edge intelligence for large language models: A contemporary survey," *IEEE Commun. Surveys Tuts.*, vol. 27, no. 6, pp. 3820–3860, Dec. 2025.
- [3] M. Hu, Q. He, and D. Wu, "QLLMS: Quantization-adaptive LLM scheduling for partially informed edge serving systems," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, May 2025, pp. 1–10.
- [4] S. Jang and R. Morabito, "Edge-first language model inference: Models, metrics, and tradeoffs," in *Proc. IEEE 45th Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Jul. 2025, pp. 309–314.
- [5] D. Kafetzis, R. Khalili, and I. Koutsopoulos, "Large language model partitioning for low-latency inference at the edge," in *Proc. 23rd Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, Aug. 2025, pp. 1–8.
- [6] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee, "Taming throughput-latency tradeoff in LLM inference with sarathi-serve," in *Proc. 18th USENIX Conf. Oper. Syst. Des. Implement. (OSDI)*, Jul. 2024.
- [7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inform. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.
- [8] A. K. Jaiswal, B. Hu, L. Yin, Y. Ro, T. Chen, S. Liu, and A. Akella, "FFN-SkipLLM: A hidden gem for autoregressive decoding with adaptive feed forward skipping," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Miami, Florida, USA, Nov. 2024, pp. 16943–16956.
- [9] H. Xia, Z. Yang, Q. Dong, P. Wang, Y. Li, T. Ge, T. Liu, W. Li, and Z. Sui, "Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding," in *Proc. Findings Assoc. Comput. Linguistics: ACL 2024*, Aug. 2024, pp. 7655–7671.
- [10] M. Yin, M. Chen, K. Huang, and M. Wang, "A theoretical perspective for speculative decoding algorithm," in *Proc. Adv. Neural Inform. Process. Syst. (NeurIPS)*, vol. 37, Dec. 2024, pp. 128 082–128 117.
- [11] M. Yan, S. Agarwal, and S. Venkataraman, "Decoding speculative decoding," in *Proc. Conf. Nations Americas Chapter Assoc. Comput. Linguistics: Human Lang. Technol.*, Albuquerque, New Mexico, Apr. 2025, pp. 6460–6473.
- [12] R. Svirschevski, A. May, Z. Chen, B. Chen, Z. Jia, and M. Ryabinin, "SpecExec: Massively parallel speculative decoding for interactive LLM inference on consumer devices," in *Proc. Adv. Neural Inform. Process. Syst. (NeurIPS)*, vol. 37, Dec. 2024, pp. 16 342–16 368.
- [13] R. Tiwari, H. Xi, A. Tomar, C. R. C. Hooper, S. Kim, M. Horton, M. Najibi, M. W. Mahoney, K. Keutzer, and A. Gholami, "QuantSpec: Self-speculative decoding with hierarchical quantized KV cache," in *Proc. 42nd Int. Conf. Mach. Learn. (ICML)*, Jul. 2025, pp. 59 668–59 686.
- [14] J. Zhang, J. Wang, H. Li, L. Shou, K. Chen, G. Chen, and S. Mehrotra, "Draft & verify: Lossless large language model acceleration via self-speculative decoding," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Bangkok, Thailand, Aug. 2024, pp. 11 263–11 282.
- [15] H. Xia, Y. Li, J. Zhang, C. Du, and W. Li, "SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2025, pp. 1–24.
- [16] J. Jiang, Z. Chen, H. Shin, and A. Nallanathan, "Edge inference for large language models with pipeline parallelism and batching," *IEEE Trans. Commun.*, vol. 74, pp. 8390–8406, Apr. 2026.
- [17] X. Zhang, J. Nie, Y. Huang, G. Xie, Z. Xiong, J. Liu, D. Niyato, and X. Shen, "Beyond the cloud: Edge inference for generative large language models in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 24, no. 1, pp. 643–658, Jan. 2025.
- [18] C. Liu and J. Zhao, "Resource allocation in large language model integrated 6G vehicular networks," in *Proc. IEEE 99th Veh. Technol. Conf. (VTC)*, Sep. 2024, pp. 1–6.
- [19] K. Zhang, H. He, S. Song, J. Zhang, and K. B. Letaief, "Communication-efficient distributed on-device LLM inference over wireless networks," *IEEE J. Sel. Topics Signal Process.*, vol. 19, no. 7, pp. 1301–1317, Oct. 2025.
- [20] B. Zhu, Z. Chen, L. Zhao, H. Shin, and A. Nallanathan, "Efficient LLM inference over heterogeneous edge networks with speculative decoding," *arXiv preprint arXiv:2510.11331*, 2025.
- [21] Y. Xu, S. Zhou, and Z. Niu, "DiP-SD: Distributed pipelined speculative decoding for efficient LLM inference at the edge," *arXiv preprint arXiv:2604.20919*, 2026.
- [22] X. Li, D. Spatharakis, S. Ghafouri, J. Fan, H. Vandierendonck, D. John, B. Ji, and D. S. Nikolopoulos, "SLED: A speculative LLM decoding framework for efficient edge serving," in *Proc. 10th ACM/IEEE Symp. Edge Comput. (SEC)*, Dec. 2025.
- [23] J. Wang, Y. Su, J. Li, Q. Xia, Z. Ye, X. Duan, Z. Wang, and M. Zhang, "OPT-tree: Speculative decoding with adaptive draft tree structure," *Transactions of the Association for Computational Linguistics*, vol. 13, pp. 188–199, Feb. 2025.
- [24] T. Liu, Y. Li, Q. Lv, K. Liu, J. Zhu, W. Hu, and X. Sun, "PEARL: Parallel speculative decoding with adaptive draft length," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, vol. 2025, May 2025, pp. 1085–1104.
- [25] A. Albaseer, E. Bentafat, M. Hamood, M. Abdallah, A. Al-Fuqaha, and M. Hamdi, "Think fast, infer smart: A hybrid distributed LLMs inference at the wireless edge," in *Proc. IEEE 36th Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2025, pp. 1–16.
- [26] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Rühle, L. V. Lakshmanan, and A. H. Awadallah, "Hybrid LLM: Cost-efficient and quality-aware query routing," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Vienna Austria, May 2024, pp. 1–19.
- [27] S. Oh, J. Kim, J. Park, S.-W. Ko, J. Choi, T. Q. Quek, and S.-L. Kim, "Communication-efficient hybrid language model via uncertainty-aware opportunistic and compressed transmission," *IEEE Trans. Commun.*, early access 2026.
- [28] H. Huang, F. Yang, Z. Liu, Y. Xu, J. Li, Y. Liu, X. Yin, D. Li, P. Ren, and E. Barsoum, "Jakiro: Boosting speculative decoding with decoupled multi-head via MoE," *arXiv preprint arXiv:2502.06282*, 2025.
- [29] M. Fang, Z. Fu, Q. Zhao, and J. Wang, "When, what, and how: Rethinking retrieval-enhanced speculative decoding," *arXiv preprint arXiv:2511.01282*, 2025.
- [30] H. Oh, K. Kim, J. Kim, S. Kim, J. Lee, D.-s. Chang, and J. Seo, "ExeGPT: Constraint-aware resource scheduling for LLM inference," in *Proc. 29th ACM Int. Conf. Archit. Support Program. Lang. Operating Syst. (ASPLOS)*, Apr. 2024, p. 369–384.
- [31] H. Qiu, W. Mao, A. Patke, S. Cui, S. Jha, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "Efficient interactive LLM serving with proxy model-based sequence length prediction," in *Proc. 5th Int. Workshop Cloud Intell./AIOps*, vol. 5, San Diego, CA, USA, Apr. 2024, pp. 1–7.
- [32] R. Gong, S. Bai, S. Wu, Y. Fan, Z. Wang, X. Li, H. Yang, and X. Liu, "Past-future scheduler for LLM serving under SLA guarantees," in *Proc. 30th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Mar. 2025, pp. 798–813.
- [33] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *Proc. 16th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Jul. 2022, pp. 521–538.
- [34] O. Khattab and M. Zaharia, "ColBERT: Efficient and effective passage search via contextualized late interaction over BERT," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, Mar. 2020, pp. 39–48.
- [35] PyTorch, "Reference API." [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pad_sequence.html
- [36] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, Honolulu, Hawaii, USA, Jul. 2023, pp. 31 094–31 116.
- [37] NVIDIA, "NVIDIA multi-instance GPU user guide release r580," 2025. [Online]. Available: https://docs.nvidia.com/datacenter/tesla/pdf/MIG_User_Guide.pdf
- [38] F. Xu, J. Xu, J. Chen, L. Chen, R. Shang, Z. Zhou, and F. Liu, "iGniter: Interference-aware GPU resource provisioning for predictable DNN inference in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 812–827, Mar. 2023.
- [39] W. Shi, S. Zhou, Z. Niu, M. Jiang, and L. Geng, "Multiuser co-inference with batch processing capable edge server," *IEEE Trans. Wireless Commun.*, vol. 22, no. 1, pp. 286–300, Jan. 2023.
- [40] H. Shojaei, T. Basten, M. Geilen, and A. Davoodi, "A fast and scalable multidimensional multiple-choice knapsack heuristic," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 4, pp. 1–32, Oct. 2013.
- [41] M. Hifi, M. Michrafy, and A. Sbihi, "Heuristic algorithms for the multiple-choice multidimensional knapsack problem," *J. Oper. Res. Soc.*, vol. 55, no. 12, pp. 1323–1332, Jul. 2004.