Beyond the Cloud: Edge Inference for Generative Large Language Models in Wireless Networks

Xinyuan Zhang, *Graduate Student Member, IEEE*, Jiangtian Nie, Yudong Huang, *Graduate Student Member, IEEE*, Gaochang Xie, *Graduate Student Member, IEEE*, Zehui Xiong, *Senior Member, IEEE*, Jiang Liu, Dusit Niyato, *Fellow, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*— Generative Artificial Intelligenge (GAI) is revolutionizing the world with its unprecedented content creation ability. Large Language Model (LLM) is one of its most embraced branches. However, due to LLM's substantial size and resource-intensive nature, it is cloud-hosted, raising concerns about privacy, usage limitations, and latency. In this paper, we propose to utilize ubiquitous distributed wireless edge computing resources for real-time LLM inference. Specifically, we introduce a novel LLM edge inference framework, incorporating batching and model quantization to ensure high throughput inference on resource-limited edge devices. Then, based on the architecture of transformer decoder-based LLMs, we formulate an edge inference optimization problem which is NP-hard, considering batch scheduling and joint allocation of communication and computation resources. The solution is the optimal throughput under edge resource constraints and heterogeneous user requirements on latency and accuracy. To solve this NP-hard problem, we develop an OT-GAH (Optimal Tree-search with Generalized Assignment Heuristics) algorithm with reasonable complexity and $\frac{1}{2}$-approximation ratio. We first design the OT algorithm with online tree-pruning for single-edge-node multi-user case, which navigates the inference request selection within the tree structure to miximize throughput. We then consider the multi-edge-node case and propose the GAH algorithm, which recrusively invokes the OT in each node's inference scheduling iteration. Simulation results demonstrate the superiority of OT-GAH batching over other benchmarks, revealing an over 45% time complexity reduction compared to brute-force searching.**

***Index Terms*— Generative AI, edge inference, wireless networks, multiuser edge computing.**

Received 13 October 2023; revised 1 April 2024, 31 May 2024, and 31 August 2024; accepted 3 November 2024. Date of publication 20 November 2024; date of current version 10 January 2025. This work was supported in part by the State Scholarship Fund of China Scholarship Council under Grant 202106470020; in part by the National Research Foundation, Singapore, and the Infocomm Media Development Authority under its Future Communications Research and Development Program; in part by the Defence Science Organization (DSO) National Laboratories under the Artificial Intelligence (AI) Singapore Program; in part by Singapore Ministry of Education (MOE) Tier 1 under Grant RG87/22; in part by the Nanyang Technological University (NTU) Centre for Computational Technologies in Finance (NTU-CCTF); in part by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 under Grant MOE-T2EP20221-0017; and in part by the Singapore University of Technology and Design (SUTD) Kickstarter Initiative under Grant SKI 20210204. The associate editor coordinating the review of this article and approving it for publication was X. Gong. *(Corresponding author: Jiangtian Nie.)*

Xinyuan Zhang, Yudong Huang, and Gaochang Xie are with the State Key Laboratory of Networking and Switching Technology, BUPT, Beijing 100876, China (e-mail: zhangxy@mail.zgclab.edu.cn; hyduni@bupt.edu.cn; xiegaochang@bupt.edu.cn).

Jiangtian Nie is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: jnie001@e.ntu.edu.sg).

Zehui Xiong is with the Information Systems Technology and Design (ISTD) Pillar, Singapore University of Technology and Design, Singapore 487372 (e-mail: zehui_xiong@sutd.edu.sg).

Jiang Liu is with the State Key Laboratory of Networking and Switching Technology, BUPT, Beijing 100876, China, and also with Purple Mountain Laboratories, Nanjing 211111, China (e-mail: liujiang@bupt.edu.cn).

Dusit Niyato is with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798 (e-mail: dniyato@ntu.edu.sg).

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: sshen@uwaterloo.ca).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TWC.2024.3497923.

Digital Object Identifier 10.1109/TWC.2024.3497923

I. Introduction

JUST as electricity transformed the world a century ago, Artificial Intelligence (AI) is now revolutionizing our daily life [1]. A significant milestone in this evolution is Generative AI (GAI), which exceeds mere analytics to enable machine creativity [2], [3], [4], [5]. Among its prominent subfields, Large Language Models (LLMs) have excelled in diverse applications over the past years, from code generation and customer service chatbots to the interpretation of academic literature [6]. The extraordinary generative potential stems from LLM's sophisticated architecture with billions of neurons, leading to intensive resource consumption during inference. Hence, LLMs primarily operate in clouds nowadays. This centralized inference approach, however, introduces challenges in privacy, usage limitations, and latency [7], [8], [9]. These issues pose significant obstacles to LLM's businesses expansion and broad adoption, necessitating careful consideration and innovative solutions.

Wireless edge networks offer a promising complement to cloud-based LLM inference. By harnessing ubiquitous edge computing power, LLM service capacity is amplified. Besides, deploying LLMs on nearby edge nodes circumvents long propagation. Furthermore, the risk of data leakage from single-point vulnerabilities is reduced as user prompts are spread across distributed edge nodes over time. While many studies [7], [8], [9] have proposed solutions for collaborative edge-cloud GAI service architectures and edge-enhanced GAI computations, none of them has explicitly tackled the challenges related to LLM inference in wireless edge networks.

Exploring LLM inference in wireless edge networks is crucial due to LLM's distinct attributes compared with other

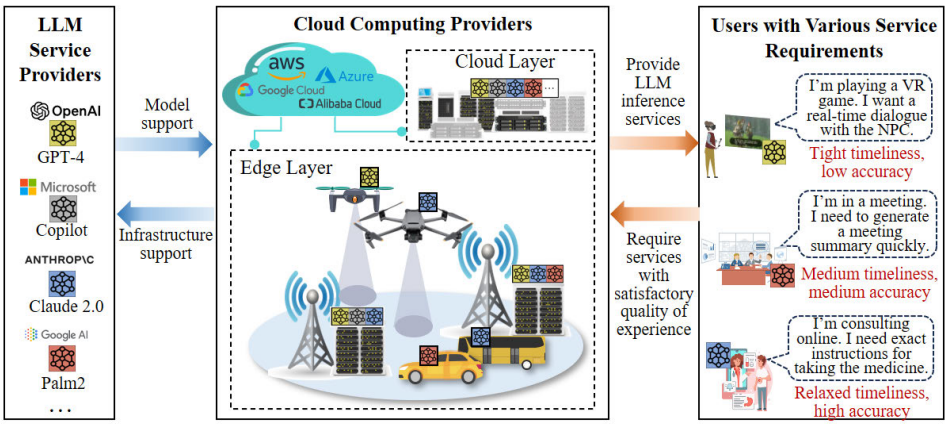


Fig. 1. Edge-enhanced large language model service provisioning framework.

GAI models. First, transformer-based LLMs are notably larger. Specifically, the advanced GPT-4 model has around 1.8 trillion parameters [10], whereas diffusion models like Stable Diffusion XL have only 3.5 billion [11]. This magnitude demands extensive memory footprint during inference. Additionally, transformer decoder-based LLMs sequentially generate tokens in an autoregressive manner [12]. This means that the computation of every new output token must go through all layers of the model, exacerbating the computational burden [13]. Moreover, the self-attention module requires each output token to be calculated using the cache of all its preceding tokens [14]. The cache is retained until the output sequence ends, thereby further intensifying memory requirements. In summary, the inherent complexities of LLMs make their inference on resource-constrained edge nodes particularly challenging. Although [15] delved into joint LLM caching and inference in wireless edge networks, it overlooked LLMs' significant size, extensive memory footprint, and computational intensity.

Therefore, in this paper, we introduce a novel transformer-based LLM inference framework for mobile users in wireless edge networks. As shown in Fig. 1, LLM service providers (LSPs) cooperate with cloud computing providers (CCPs) for LLM deployment and inference. CCPs employ edge computing power, such as UAVs over hotspot zones or vehicles in dense areas, to augment their services and ensure instant delivery via wireless networks. Mobile users can conveniently access LLM services in a pay-as-you-go manner. However, two significant problems arise in this framework. First, is it feasible to run resource-intensive LLM inference on resource-limited wireless edge nodes? Second, how can we assess the LLM service quality and ensure that it meets user requirements? To address the first challenge, we adopt model quantization in the framework to conserve memory footprint during LLM edge inference by storing model weights and activations with reduced bit precision [6], [16]. To address the second challenge, we recognize the variability in user accuracy demands. For example, online medical prescriptions prioritize output text accuracy unlike poetry writing. Hence, we introduce the perplexity differential [17] as a metric to balance the trade-off between quantization-induced accuracy loss and user accuracy demands. Moreover, we notice that while Metaverse dialogue and other social network live interactions demand stringent timeliness, blog creation is less time-sensitive. Thus, latency is also considered for service quality evaluation. Furthermore, our framework utilize batching to compute multiple user requests parallelly [18], [19], boosting the inference capability of wireless edge networks.

Based on the above framework, we propose an LLM edge inference optimization problem, encompassing batching scheduling and resource allocation. The problem is NP-hard due to the following three facts. Firstly, batching scheduling and resource allocation are intertwined. A request's prompt size and expected output length affect both the latency of batched inference and the consumption of memory and communication resources. Secondly, users' scheduled requests are inter-dependent. An inference output cannot be released until the entire batch of requests is fully processed, making each request's batched inference latency reliant on others in the same batch [20]. Lastly, there is a tight interplay between communication and computation, requiring joint consideration of channel states and user needs. To efficiently solve this complex problem, we propose the Optimal Tree-search with Generalized Assignment Heuristics (OT-GAH) algorithm. Our contributions can be summarized as follows:

- We introduce the LLM edge inference framework tailored for wireless networks, incorporating model quantization and batching techniques to enhance inference throughput on resource-constrained edge nodes. The framework ensures real-time, privacy-protected LLM services universally accessible to any device, at anytime, from anywhere.
- We propose the LLM edge inference optimization problem that aims at maximizing the throughput by scheduling batching in granularity of requests and jointly allocating communication and computation resources. The constraints include edge resources of communication and memory, as well as user-specific latency and accuracy requirements. The problem is a variant of the multiple multi-dimensional knapsack problem, which is NP-hard.
- We design an OT-GAH algorithm to solve the problem. First, we develop a depth-first Optimal Tree-search (OT)

algorithm with online pruning to determine the optimal solution for a single edge node serving multiple users. Then, for the multi-node, multi-user case, we adopt the generalized assignment problem model and propose a Generalized Assignment Heuristic (GAH) approximation algorithm. We theoretically prove that the OT-GAH algorithm is of polynomial time and $\frac{1}{2}$-approximation ratio.

- We demonstrate the effectiveness of the proposed OT-GAH batching against other batching schemes by simulations. We also illustrate the varied effect of quantization approaches on LLM edge inference performance. The superiority of OT-GAH algorithm is confirmed compared to the brute-force approach.

The rest of the paper is structured as follows: Section II introduces the ralated works. Section III represents the LLM edge inference model. The optimization problem formulation and solution are proposed in Section IV. Simulation results are given in Section V, followed by the conclusions in Section VI.

II. Related Work

In this section, we provide a brief review of related work, i.e., collabrative cloud-edge-end GAI service provisioning, batching inference, and LLM quantization.

A. Collaborative Cloud-Edge-End GAI Service Provisioning

GAI has revolutionized content creation with its unparalleled ability to automate diverse outputs like text, images, and videos [8], [21]. It offers significant time and resource savings compared to manual methods, transitioning us from professional and user-generated content to a new era. GAI's remarkable generative capacity originates from its complex architecture, which boasts a vast number of parameters. This design allows the model to discern subtle data nuances, grasp underlying structures, dependencies, and statistical distributions of the training data, and produce sophisticated, realistic outputs [12], [22]. Yet, such large-size models demand rigorous computational operations and substantial memory resources during inference, leading to their deploymet on cloud platforms. Specifically, users upload their GAI tasks to the cloud via core networks. The cloud servers execute the GAI models and return the generation outputs to the users [7].

However, such cloud-centric approach poses issues as follows. Firstly, privacy concerns arise from the centralized storage of GAI service prompts on the cloud platform, as seen with instances such as Samsung's ban on ChatGPT usage due to data exposure [23] and misalignment glitches causing information leakage in ChatGPT [24]. Additionally, sensitive information is vulnerable to attacks during long-distance transmission to remote clouds. Secondly, GAI service providers pose strict usage limits to users. For example, a ChatGPT Plus user can only initiate 50 chats within 3 hours [25]. Claude 2.0 can only be used in US and UK [26]. Lastly, the current centralized framework suffers from high latency. For example, in the game Mount&Blade 2, typing-based dialogues with GPT-empowered NPCs introduce 3 to 5 seconds of latency, notably affecting player experience [27].

To offer privacy-preserving, no-restriction, low-latency GAI service, the concept of collabrative cloud-edge-end GAI service provisioning is recently proposed. The authors in [7] delineated the architecture designs, benefits, and significant technical problems. Meanwhile, the authors in [8] introduced metrics for evaluating GAI service quality within collabrative cloud-edge networks and proposed a mechanism to select GAI service providers for users demanding image generation. The authors in [9] presented a device-to-device distributed offloading approach for diffusion-based GAI image-generation services. Besides, the authors in [15] explored joint LLM caching and inference in wireless edge networks, and introduced a novel metric "age of context" to evaluate the freshness of GAI-produced text. While these studies shed a light on collabrative cloud-edge-end GAI computations, they overlooked the challenges of LLM inference in resource-limited wireless edge networks, which arises from LLMs' super large model size, autoregressive generation process, and resource-demanding self-attention operations.

B. Batching Inference

Batching is a key technique to optimize accelerator utilization when using accelerators like GPUs, by exploiting their parallel computation capabilities [18]. Its process is as follows: A batch assembles a set of input requests into a single, large input tensor. This tensor is then fed as a whole into the model for parallel inference. During the inference phase, the model parameters, once loaded from off-chip memory to GPUs, are reused for multiple requests within a batch. This significantly diminishes memory access frequency and amortizes the memory access time, thereby boosting the resource utilization.

Numerous studies have delved into batching techniques for LLM inference. In [14], the authors enhanced the throughput of latency-insensitive LLM tasks on a single GPU, incorporating dynamic batch size adjustments, optimized tensor storage and retrieval, and a unique execution sequence across batches, model layers, and text iterations. In [20], the authors noted that requests within a batch require varied iteration counts in the autoregressive generation. They proposed allowing early-completing requests exit the batch early to accommodate new ones, reducing latency and boosting throughput. However, although these works focused on accelerating GAI computation on distributed computing systems by optimizing batching, our work lies in enhancing the access availability of GAI services in the wireless network domain. [19] bears a close resemblance to our research, but it was based on traditional deep neural network models and does not adapt to our work.

C. Model Quantization

Model quantization has been one of the most impactful ways to reduce computational time and memory use of neural networks by representing model weights and activations in lower bits [16]. This technique is crucial for making LLMs manageable and efficient on wireless edge nodes, given their large size and computational demands. Generally, there are two primary model quantization techniques: quantization-aware training (QAT), which necessitates full model retraining,

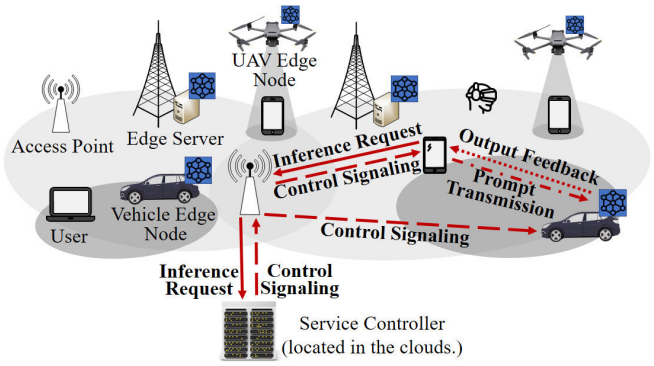


Fig. 2. The LLM edge inference network and workflows. The blue squares with different sizes represent LLMs with various quantization methods.

and post-training quantization (PTQ), which doesn't require any model retraining. Due to LLMs' extensive parameters and complex activation patterns, PTQ is often preferred, as discussed in [6]. The authors in [17] provided a comprehensive examination of various LLM PTQ schemes. Meanwhile, the authors in [28] proposed a mixed-precision quantization scheme, using 8-bit matrix multiplication excluding outlier features. This approach efficiently ran OPT-175B on a consumer GPU server without performance losses. In our study, we adopt some outstanding LLM quantization schemes in wireless edge networks and design two metrics to represent the effect of quantization on memory savings and computational time. We propose to select quantization strategies based on these metrics to match LLM services with edge node resources.

III. Models and Metrics

Consider a wireless edge network that comprises multiple users, multiple edge nodes, multiple access points, and a service controller, as illustrated in Fig. 2. An edge node, denoted as $j \in \mathcal{J} = \{1, 2, \ldots, J\}$, can be an edge server, a UAV, or a vehicle equipped with LLMs. While we demonstrate the edge inference optimizating process for only one LLM, the proposed method can be easily extended to other LLMs. Besides, although all edge nodes in Fig. 2 are deployed with the same LLM, the model quantization methods vary across these nodes. A more in-depth discussion on this is provided in Section III-B. Let B_j^U and B_j^D be the total uplink and downlink bandwidth of j, respectively. C_j denotes the computing speed and M_j represents the memory capacity.

As depicted in Fig. 2, when a user requests an LLM service, it first sends its inference request to a nearby access point, which then forwards it to the service controller via wired links. After deciding on edge inference scheduling, the controller dispatches control signaling messages to user devices and edge nodes through access points. These messages inform user devices of edge nodes assigned for prompt transmission and processing, and also instruct edge nodes to reserve the necessary bandwidth, computation, and memory resources for the inference tasks they manage. Subsequently, the user device sends its inference prompt to the assigned edge node and receives output feedback. Here, the access point serves as a stationary intermediate node, forwarding inference requests and control signaling messages between the controller and mobile user devices and edge nodes. The edge node is responsible for receiving the actual inference prompts and processing them.

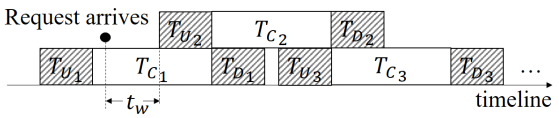


Fig. 3. Timeline for LLM edge inference.

A user request $i \in \mathcal{I} = \{1, 2, \ldots, I\}$ containing its request information $\langle s_i, n_i, \tau_i, a_i \rangle$ is forwarded to its nearby access point through the application API, just like ChatGPT playground [29]. Here, s_i is i's input prompt length, n_i is i's required maximum text output length, categorized into multiple levels of $\{N_1, N_2, \ldots, N_{max}\}$. The output length depends on the input length and LLM service type. For example, in translation, the output length matches the input. For text summary, the output is much shorter than the input, while in article writing, the output is much longer. We assume that the output length is defined by users or the API. The latency constraint of i is symbolized by τ_i, which measures the maximum duration from the request's dispatch to the inference output's receipt. a_i is the required text output accuracy, further detailed in Section III-B.

The following protocol for edge inference is considered [19]. As shown in Fig. 3, time is divided into epochs, each of which is further divided into an uplink communication slot T_U, a computation slot T_C, and a downlink communication slot T_D. To maximize resource efficiency, each T_C starts immediately after the prior one ends, overlapping with the preceding epoch's T_D and the subsequent epoch's T_U. Slot durations adjust periodically based on long-term service observations. User requests arrive at access points within an arbitrary epoch, with the wait time until next epoch's start denoted by t_w. By next epoch's T_U, the controller aggregates requests from all access points and schedules edge inference for the next epoch. In next epoch's T_U, users send input prompts to designated edge nodes. By T_U's end, edge nodes assemble the prompts into a single batch, completing inference on the batch within T_C. During next epoch's T_D, the inference outputs are fed back to users. Models and metrics are described as follows.

A. Communication Model

As illustrated in Fig. 2, the data sizes for inference requests and control signaling are negligible. Therefore, the communication model focuses on prompt transmission and output feedback. The system employs orthogonal frequency-division multiple access (OFDMA) to allocate broadband spectrum to scheduled users. With the number of sub-carriers assumed to be sufficiently large (e.g., several thousands in 5G), bandwidth partitioning can be approximated as being continuous [19]. Let ρ_{ij}^U (for uplink) and ρ_{ij}^D (for downlink) be the fractions of edge node j's bandwidth allocated to i, with both ρ_{ij}^U and ρ_{ij}^D between 0 and 1. For any scheduled user request i, the associated channel is assumed frequency non-selective

and the channel gain can be represented by scalars h_{ij}^U and h_{ij}^D, remaining consistent within one epoch. The uplink and downlink transmission power are given by p_{ij}^U and p_j^D, respectively. Using measurement techniques such as CSI-RS, edge nodes can estimate the channel gain and uplink transmission power. This data is then forwarded to the server controller for bandwidth allocation. However, if the information is not available to the controller, bandwidth planning should be carried out at the distributed edge nodes, which will be further explored in future work. The uplink communication rate, r_{ij}^U, is given by:

$$r_{ij}^U = \rho_{ij}^U B_j^U \log_2\left(1 + \frac{p_{ij}^U {h_{ij}^U}^2}{N_0}\right),$$

where N_0 is the white Gaussian channel noise power. The downlink communication rate r_{ij}^D can be modeled similarly.

Given that the prompt must be uploaded within one T_U, which means $r_{ij}^U T_U \geq s_i$ mathematically, the minimum fraction of allocated uplink bandwidth, denoted as $\rho_{ij,min}^U$, is:

$$\rho_{ij,min}^U \triangleq \frac{s_i}{T_U B_j^U \log_2\left(1 + \frac{p_{ij}^U {h_{ij}^U}^2}{N_0}\right)}, \quad (1)$$

where $\rho_{ij}^U \geq \rho_{ij,min}^U$ if request i is scheduled. Similary, since the duration of output downloading must be within T_D, $\rho_{ij,min}^D$ is defined:

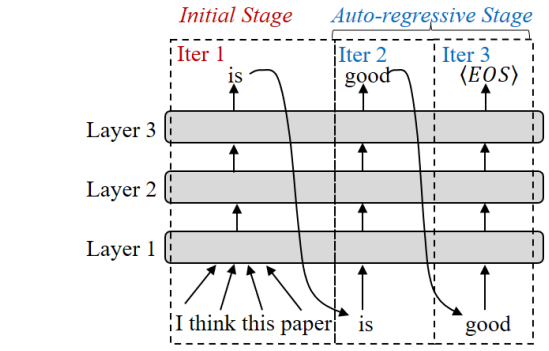
$$\rho_{ij,min}^D \triangleq \frac{n_i}{T_D B_j^D \log_2\left(1 + \frac{p_j^D {h_{ij}^D}^2}{N_0}\right)}. \quad (2)$$

B. Inference Model

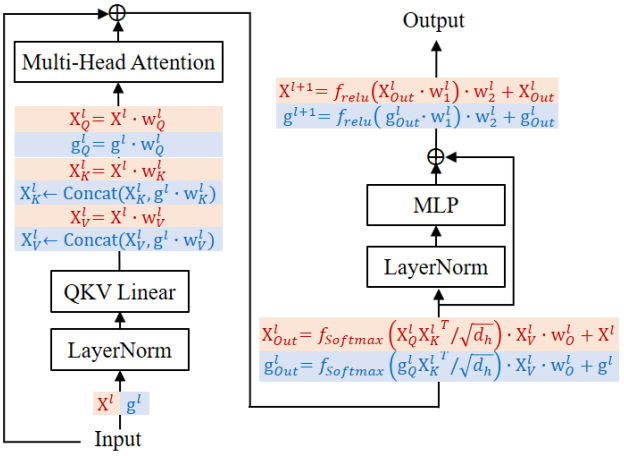
1) Basics of Transformer Decoder-Based LLM

Generative LLMs are autoregressive language models composed of stacked transformer decoder layers. Fig. 4(a) illustrates a simplified inference procedure of a three-layer GPT-3. Here, we take GPT-3 as an example because it typifies the architecture of transformer decoder-based LLMs and GPT-4's architecture details are not released. In Fig. 4(a), the model receives a sequence of prompt tokens and completes the sequence iteratively until the end-of-sequence terminator, $\langle EOS \rangle$, is produced. We define the run of all layers as an iteration. The first iteration, termed the *Initial Stage*, takes all input prompt tokens and generates a new token. The generated token becomes input for the next iteration. Subsequent iterations, termed the *Autoregressive Stage*, input the previously generated token and produce the next, composing a sequential, one-by-one inference procedure [20]. Since each transformer layer has the same architecture, we focus on a single transformer layer in the following.

Fig. 4(b) zooms in on a GPT-3 transformer decoder layer. Among the operations that compose the layer, *Attention* is the essence that distinguishes transformer from other architectures. It takes three inputs (i.e., query, key, and value) and computes a weighted average of tokens so that each token is aware of the other. Since *Attention* relies on all preceding token keys and values, a key-value cache (KV cache) is maintained



(a) Inference procedure of a transformer decoder-based LLM.



(b) The computation procedure of l-th transformer decoder layer. The colored blocks represent the outputs of intermediate steps. Red blocks denote operations during the *Initial Stage*, while blue blocks signify operations during the *Autoregressive Stage*.

Fig. 4. Architecture of a transformer decoder-based LLM.

across iterations to avoid redundant calculations. During the *Initial Stage*, the KV cache for all prompt tokens is computed and retained per transformer layer. During the *Autoregressive Stage*, the KV cache updates with each new token until generation concludes [14].

The computation precedure of the l-th transformer decoder layer is illustrated in Fig. 4(b). While we ignore minor details like LayerNorm, Mask, Dropout, and Residual Connection for simplicity, we focus on matrix multiplication [20]. Let d_m represent the transformer's hidden dimension, n_h the number of attention heads, d_h the hidden dimension of one attention head, and d_f the hidden dimension of the FFN. The weight parameters of the l-th layer are specified by $\mathbf{w}_Q^l, \mathbf{w}_K^l, \mathbf{w}_V^l \in \mathcal{R}^{d_m \times d_h n_h}$, $\mathbf{w}_O^l \in \mathcal{R}^{d_h n_h \times d_m}$, $\mathbf{w}_1^l \in \mathcal{R}^{d_m \times d_f}$, and $\mathbf{w}_2^l \in \mathcal{R}^{d_f \times d_m}$. During the *Initial Stage*, the input of the l-th layer is specified by $\mathbf{X}^l \in \mathcal{R}^{s \times d_m}$. During the *Autoregressive Stage*, the l-th layer's input is $\mathbf{g}^l \in \mathcal{R}^{1 \times d_m}$.

2) Batching Model

Now, we focus on the LLM inference with batching during a given computation slot and analyze its peak footprint memory and inference latency.

Batching is a key to enhance the throughput, measured by the number of executed tasks per second [19]. The batch size is defined as the number of inference prompts processed together within a batch. Given any T_C, we define a 0-1 variable x_{ij} to represent whether user request i is allocated to edge node j for inference. If so, $x_{ij} = 1$ and $x_{ij} = 0$ otherwise. We assume that each edge node only processes one batch in any T_C. Hence, the batch size of any j can be represented by $\sum_{i\in\mathcal{I}} x_{ij}$. Furthermore, at the beginning of T_C, the edge node assembles all received input prompt tensors into a single input tensor. In this step, all input prompts must be extended to the maximum token length for parallel execution. We set the maximum token length to not exceed s'. The outputs for all batched requests are released once the batch completes.

Although batching can greatly reduce memory access frequency and amortizes memory access time, a large batch size can result in excessive memory demands during inference and extreme latency for users, which instead leads to throughput reduction. This necessitates intelligent batching scheduling to balance the tradeoff. Hence, in the following, we analyze the peak footprint memory and inference latency with batching.

1) Memory Analysis: During LLM inference, the memory footprint mainly comes from two components: weights and KV cache. The model weight parameters, including $\mathbf{w}_Q^l, \mathbf{w}_K^l, \mathbf{w}_V^l, \mathbf{w}_O^l, \mathbf{w}_1^l$, and $\mathbf{w}_2^l$ for each transformer decoder layer in Fig. 4(b), need to be loaded from off-chip memory and then maintained in GPUs. Given the FP16 storage precision, which utilizes 16 bits to encode a floating-point number, a matrix $\mathbf{A} \in \mathcal{R}^{m\times n}$ requires $2mn$ bytes of memory. Hence, the memory for maintaining weights on an edge node j during LLM inference is:

$$m_j^1 = L\left(8d_m d_h n_h + 4d_m d_f\right), \tag{3}$$

where L is the number of model layers, $8d_m d_h n_h$ denotes memory for $\mathbf{w}_Q^l, \mathbf{w}_K^l, \mathbf{w}_V^l, \mathbf{w}_O^l$ in Fig. 4(b), and $4d_m d_f$ represents memory for $\mathbf{w}_1^l, \mathbf{w}_2^l$ in Fig. 4(b) [14].

Another component contributing to memory footprint is the KV cache. For subsequent token generations, the KV cache components should be retained for reuse. Hence, the footprint memory of KV cache during batch inference is:

$$m_j^2 = \sum_{i\in\mathcal{I}} x_{ij}\left(4Ls' d_h n_h + 4Ln_i d_h n_h\right), \tag{4}$$

where $4s' d_h n_h$ is the cache of $\mathbf{X}_K^l, \mathbf{X}_V^l$ during *Initial Stage*, and $4n_i d_h n_h$ is the cache of $\mathbf{g}_K^l = \mathbf{g}^l \cdot \mathbf{w}_K^l$ and $\mathbf{g}_V^l = \mathbf{g}^l \cdot \mathbf{w}_V^l$ per prompt during *Auto-regressive Stage*.

Therefore, the total peak footprint memory of an edge node j during a LLM batched inference is:

$$m_j = m_j^1 + m_j^2 = k_1 + k_2 \sum_{i\in\mathcal{I}} x_{ij}\left(s' + n_i\right), \tag{5}$$

where k_1 and k_2 are constants related to specific LLM model parameters.

2) Latency Analysis: The inference latency is defined as the total delay spent on computing a batch of prompts and generate all output tokens. We adopt the inference acceleration method detailed in [14], which facilitates parallel execution of model weight reading and computation. Consequently, the overall latency is governed by the longer of the two durations: weight reading or output computation. Evidence from [14] indicates that the time for weight reading is less than that for computing. Hence, the latency mainly comes from matrix multiplication computations, as shown in Fig. 4(b). Floating point operations (FLOPs) quantify computational work. For a matrix-vector multiplication with $\mathbf{A} \in \mathcal{R}^{m\times n}, \mathbf{b} \in \mathcal{R}^n$, the computation is $2mn$ FLOPs. The matrix-matrix multiplication invoving $\mathbf{A} \in \mathcal{R}^{m\times n}, \mathbf{B} \in \mathcal{R}^{n\times p}$ is $2mnp$ FLOPs. Hence, the batched inference latency of *Initial Stage* on edge node j is:

$$t^1 = \frac{L\sum_{i\in\mathcal{I}} x_{ij}}{C_j} \left(6s' d_m d_h n_h + \left(4s'^2 d_h n_h + 2s' d_m d_h n_h\right) + 4s' d_m d_f\right), \tag{6}$$

where $6s' d_m d_h n_h$ denotes the computation to obtain $\mathbf{X}_Q^l, \mathbf{X}_K^l, \mathbf{X}_V^l$, $4s'^2 d_h n_h + 2s' d_m d_h n_h$ denotes the calculation of $\mathbf{X}_{Out}^l$, and $4s' d_m d_f$ is related to $\mathbf{X}^{l+1}$.

Similarly, the batched inference latency to get $\mathbf{g}_Q^l, \mathbf{X}_K^l, \mathbf{X}_V^l, \mathbf{g}_{Out}^l, \mathbf{g}^{l+1}$ in *Auto-regression Stage* is:

$$t_j^2 = \frac{L}{C_j} \sum_{i\in\mathcal{I}} x_{ij}(n_i - 1)\left(6d_m d_h n_h + \left(4(s' + \frac{n_i}{2}) d_h n_h + 2d_m d_h n_h\right) + 4d_m d_f\right), \tag{7}$$

where the latency increases with output iteration rounds. Here, $(s' + \frac{n_i}{2})$ arises because the dimensions of $\mathbf{X}_K^l$ and $\mathbf{X}_V^l$ grow from $\mathcal{R}^{s'\times d_h n_h}$ to $\mathcal{R}^{(s'+n_i)\times d_h n_h}$ with each output iteration round, and we take the average dimension.

Therefore, the total latency of LLM batched inference on an edge node j is:

$$t_j = t_j^1 + t_j^2 = \frac{1}{C_j} \sum_{i\in\mathcal{I}} x_{ij}(k_3 + k_4 n_i + k_5 n_i^2), \tag{8}$$

where k_3, k_4, and k_5 are constants related to s' and specific LLM model parameters, and t_j should be shorter than T_C.

3) Quantization Model

Given the resource-intensive nature of LLMs, PTQ encodes the LLM weights and activation tensors at a precision typically lower than FP16, often using INT8 or INT4 (using 8-bit or 4-bit integers to encode data) [6]. By transitioning from FP16 to INT8, the memory needed for tensor storage is reduced by half. The computational cost for matrix multiplication decreases by a factor of 4 [16]. However, various PTQ approaches employ distinct quantization strategies. Some only quantize weights, while others quantize both weights and activations. The precision can also vary, encompassing INT8, INT4, or even a mix, leading to differential impacts on memory saving and computation latency. To quantify these effects, we define α_j to represent the memory reduction effect of the quantization method on edge node j. Another variable β_j is defined to denote the computational time reduction. It is worth noting that β_j is not always the square of α_j, as some PTQ methods only compress specific tensor types. Importantly, both α_j and β_j are obtained in advance through extensive testing across multiple datasets [17].

Thus, accounting for model quantization on edge nodes, the footprint memory and inference latency can be expressed as:

$$m_j \leftarrow \alpha_j m_j, \quad t_j \leftarrow \beta_j t_j. \tag{9}$$

C. QoS Metrics

1) End-to-End Latency

An arbitrary LLM inference for request i is considered successful only if the end-to-end latency does not exceed τ_i. The end-to-end latency consists of four parts. Firstly, the t_w in Fig. 3. Since edge nodes consistently broadcast their inference epoch information to users, $t_{w,ij}$ is fixed for any i in edge node j's coverage. Secondly, the time for prompt transmission, which does not exceed T_U. Thirdly, the computation time at the edge node, t_j. Lastly, the time for output downloading, which is less than T_D. Hence, if $x_{ij} = 1$, the following condition must be met:

$$t_{w,ij} + T_U + \beta_j t_j + T_D \leq \tau_i. \tag{10}$$

2) Inference Accuracy

While model quantization effectively diminishes the memory footprint and reduces inference latency, it introduces additional noise that can degrade accuracy. To address this, the *perplexity* (PPL) differential [17] is employed to measure the change in LLM accuracy pre- and post-quantization. A larger PPL differential indicates a greater accuracy loss. We define ΔPPL_j as the PPL differential of the LLM deployed on edge j. Since the PPL differential of a specific quantization method is assessed across a wide range of test datasets, ΔPPL_j is assumed to be known [17]. Furthermore, we introduce f as a function that maps PPL differential to accuracy [17], which is monotonically decreasing. Consequently, for users prioritizing high-accuracy LLM services, if $x_{ij} = 1$, the following condition must hold true:

$$a_i \leq f(\Delta PPL_j), \tag{11}$$

which means the accuracy of provided generation text must be higher than required value a_i.

IV. Problem Formulation and Solution

In this section, we first introduce the LLM edge inference optimization problem in wireless networks, aimed at maximizing throughput while adhering to constraints of bandwidth, memory, and heterogeneous user requirements on latency and accuracy. For this NP-hard problem, we develop the OT-GAH algorithm. Starting with the most basic version of this issue involving one edge node and multiple users without bandwidth constraints, we offer foundational insights for addressing more intricate scenarios. We craft an optimal algorithm that greedily selects the requests with minimal memory consumption and the most accommodating latency in a sequential search for the maximum throughput. As we progress to a version with bandwidth constraints, the complexity grows with the added bandwidth dimension. We propose the Optimal Tree-search (OT) algorithm, which constructs a search tree to represent all potential solutions and modifies the preceding algorithm by incorporating the new bandwidth dimension in its greedy search. We also introduce a tree-pruning method to bypass exploring redundant branches. Lastly, we return to the original problem. As it can be seen as a Generalized Assignment Problem (GAP), we put forth the OT-GAH algorithm, which approximates the solution heuristically by sequentially scheduling edge nodes, where each derives its local optimal solution using the OT algorithm. Fig. 5 vividly depicts the connections between problems across various versions and their corresponding algorithms.

A. Problem Formulation

Recall that edge nodes have limited communication and computation resources, and user demands vary in latency and accuracy. Let $\boldsymbol{X} = \{x_{ij} | \forall i \in \mathcal{I}, j \in \mathcal{J}\}$ be the decision variable, the LLM edge inference optimization problem for the whole wireless network is formulated as follows:

$$\text{P1:} \max_{\boldsymbol{X}} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} x_{ij} \tag{12}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{I}} \rho^U_{ij,min} x_{ij} \leq 1, \quad \forall j \in \mathcal{J} \tag{12a}$$

$$\sum_{i\in\mathcal{I}} \rho^D_{ij,min} x_{ij} \leq 1, \quad \forall j \in \mathcal{J} \tag{12b}$$

$$\alpha_j m_j \leq M_j, \quad \forall j \in \mathcal{J} \tag{12c}$$

$$(10), (11), \quad \forall x_{ij} = 1, i \in \mathcal{I}, j \in \mathcal{J} \tag{12d,12e}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \tag{12f}$$

where the objective is the throughput of edge inference, measured by the number of user prompts processed successfully. Constraints (12a)-(12c) relate to uplink communication, downlink communication, and peak footprint memory. (12d) ensures that each scheduled request receives generation output within its deadline. Meanwhile, (12e) mandates each edge node's outputs meet user accuracy requirements. By consolidating all constants unrelated to the variable, we simplify P1. The streamlined constraints are: (12c) $\sum_{i\in\mathcal{I}}(s'+n_i)x_{ij} \leq \frac{M_j - k_1\alpha_j}{k_2\alpha_j}$ and (12d) $\sum_{i\in\mathcal{I}}(k_3 + k_4 n_i + k_5 n_i^2)x_{ij} \leq \frac{\tau'_{ij} C_j}{\beta_j}$, with $\tau'_{ij} = \tau_i - t_{w,ij} - T_U - T_D$ being a constant for all $i \in \mathcal{I}, j \in \mathcal{J}$.

Proposition 1: Problem P1 is NP-hard.

Proof: By omitting (12d) and (12e), Problem P1 is reduced to a standard multiple multi-dimensional knapsack problem (MMKP) [30], [31], where each edge node j can been seen as a knapsack with three-dimensional capacities of $(1, 1, \frac{M_j - k_1\alpha_j}{k_2\alpha_j})$ from (12a)-(12c), and each request i can been envisioned as an item with a profit value of 1 and three-dimensional weights of $(\rho^U_{ij,min}, \rho^D_{ij,min}, s' + n_i)$ in (12a)-(12c). The objective is to pack items into knapsacks with maximal profit within capacity limits, aligning with the objective of P1. Since MMKP is NP-hard in strong sense, the reduction of P1 also is. Now examine the influence of (12d) and (12e) on P1. During the exploration of the solution space, the value of the right-hand side of constraint (12d) fluctuates as i dynamically joins and leaves batches on edge nodes, thereby aggravating the complexity. (12e) simply restricts edge node options for requests without further complicating the solution. In conclusion, P1 is proved to be NP-hard. □

B. Optimal Scheduling: Single Edge Node & Large Bandwidth

In this part, we consider the simplest case where a single edge node serves multiple users with sufficient bandwidth.

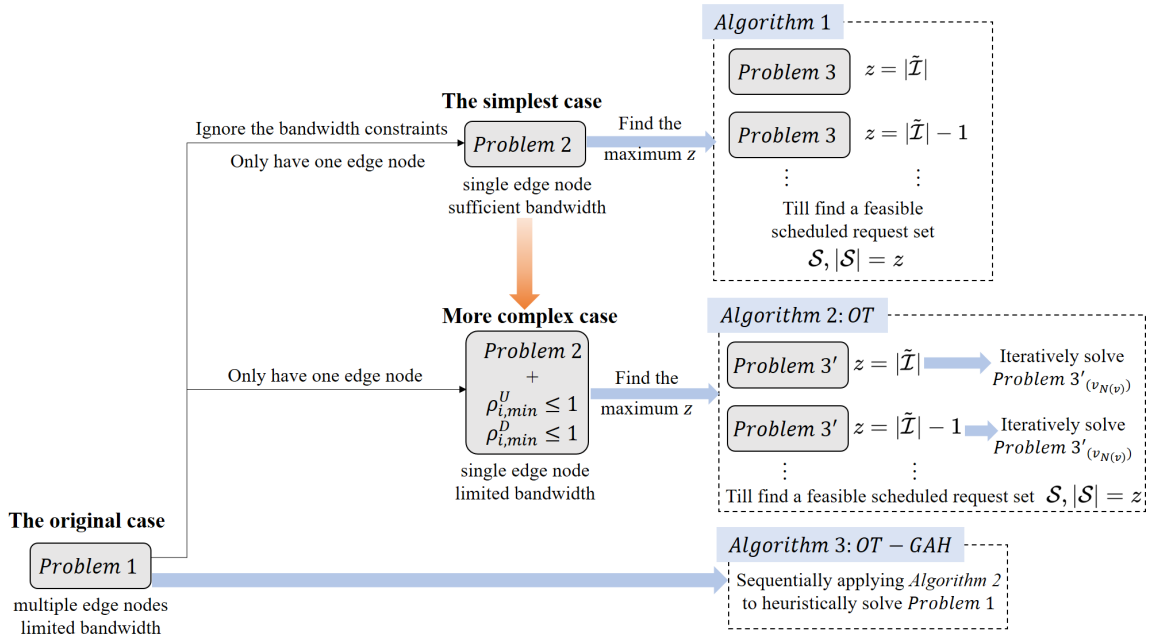


Fig. 5. The connections between problems across various versions and their corresponding algorithms.

We first transform the problem into a series of feasibility problems. Then, an optimal solution algorithm is designed by solving the feasibility problems sequentially and its optimality is proved.

Let $\tilde{\mathcal{I}}$ be the set of users who are under the edge node's coverage and are satisfied with its generation accuracy. The large uplink bandwidth mathematically means that:

$$B^U \ge \sum_{i\in\tilde{\mathcal{I}}} \frac{s_i}{T_U \log_2\left(1+\frac{p_i^U h_i^{U\,2}}{N_0}\right)}.$$

Here, we omit j because only one edge node is considered in this part. The downlink bandwidth can be represented similarly. Given the above inequality, the bandwidth constraints are always inactive. Hence, the edge inference optimization problem between a single edge node and multiple users without bandwidth constraints is as follows:

$$\text{P2:} \max_{\tilde{\boldsymbol{X}}} \sum_{i\in\tilde{\mathcal{I}}} x_i \tag{13}$$

$$\text{s.t.} \quad \sum_{i\in\tilde{\mathcal{I}}} (s'+n_i)x_i \le \frac{M-k_1\alpha}{k_2\alpha} \tag{13a}$$

$$\sum_{i\in\tilde{\mathcal{I}}} (k_3+k_4n_i+k_5n_i^2)x_i \le \frac{\tau_i' C}{\beta} \tag{13b}$$

$$x_i \in \{0,1\}. \tag{13c}$$

We observe that the optimization object is only concerned with the number of scheduled requests, whereas the constraints determine requests to be chosen. Let $\mathcal{S} = \left\{i | i \in \tilde{I}, x_i = 1\right\}$ be the set of scheduled requests, we have $|\mathcal{S}| = \sum_{i\in\tilde{\mathcal{I}}} x_i$. Problem P2 then translates to identifying such an $\mathcal{S}$ that possesses the maximum cardinality while satisfying the constraints. Given $|\mathcal{S}| = z$, Problem P2 is reformulated as:

$$\text{P3:} \quad \text{find } \mathcal{S} \subseteq \tilde{\mathcal{I}} \tag{14}$$

$$\text{s.t.} \quad |\mathcal{S}| = z \tag{14a}$$

$$\sum_{i\in\mathcal{S}} n_i \le \tilde{M} \tag{14b}$$

$$\sum_{i\in\mathcal{S}} k_4n_i + k_5n_i^2 \le \tau_{min}, \tag{14c}$$

where $\tilde{M} = \frac{M-k_1\alpha}{k_2\alpha} - s'z$ and $\tau_{min} = \min_{i\in\mathcal{S}} \tilde{\tau}_i$, $\tilde{\tau}_i = \frac{\tau_i' C}{\beta} - k_3 z$. The feasible solution to Problem P3 with the maximum z is exactly the optimal solution to P2. Given that the left hand sides of (14b) and (14c) increase with n_i, it can be deduced that when selecting a fixed number of user requests, those with smaller n_i values and larger $\tilde{\tau}_i$ are favored, reducing the likelihood of violating the constraints.

We propose an efficient algorithm to search for the optimal solution to P2, presented as Algorithm 1. In order to get the optimal as quickly as possible, we start with the highest z value and seek a feasible solution for P3. Requests are ranked by $\tilde{\tau}_i$. Denoting latency index depth as d, we check the feasibility of P3 among the top d requests, which are then reordered by n_i. τ_{min} is the minimum latency among the top d requests. The process terminates once a feasible solution emerges. The essence of Algorithm 1 is the sequential prioritization of z, $\tilde{\tau}_i$ and n_i, which guides the search direction and accelerates the solution-finding process.

Algorithm 1 can be more vividly understood through a graphical representation, as showcased in Fig. 6. In each iteration of z, imagine having z playing cards. Each card is characterized by two properties $\tilde{\tau}_i$ and n_i. The remaining cards are arranged in descending order based on $\tilde{\tau}_i$. We check whether the cards in hand meet constraints. If they do, we have identified a feasible solution. If not, we draw the card in our hand with the highest n_i value and take a new card from the remaining deck, subsequently assessing feasibility. The process continues until a feasible solution is found.

Algorithm 1 Optimal Scheduling Among a Single Edge Node and Multiple Users With Large Bandwidth

Input: Available request set $\tilde{\mathcal{I}}$.
Output: Optimal solution $\mathcal{S}$ to Problem P2.
Initialize $z = 0$, $d = 0$, $\mathcal{F}_d = \emptyset$, $\mathcal{S}' = \emptyset$;
$\tilde{\mathcal{I}}$ is sorted in $\tilde{\tau}_i$'s descending order;
for $z = |\tilde{\mathcal{I}}|, |\tilde{\mathcal{I}}| - 1, \ldots, 1$ **do**
  for $d = z, z+1, \ldots, |\tilde{\mathcal{I}}|$ **do**
    $\tau_{min} \leftarrow \tilde{\tau}_d$, $\mathcal{F}_d \leftarrow$ the first $d-1$ requests in $\tilde{\mathcal{I}}$;
    Sort $\mathcal{F}_d$ according to n_i's ascending order;
    $\mathcal{S}' \leftarrow$ the first $z-1$ requests in $\mathcal{F}_d$ and the d-th request;
    if $\mathcal{S}'$ is feasible to P3 **then**
      $\mathcal{S} \leftarrow \mathcal{S}'$, return $\mathcal{S}$;
  return no solution

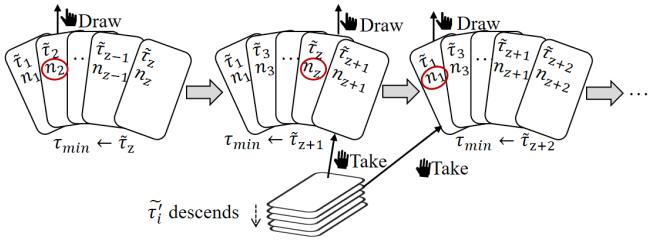


Fig. 6. Card draw-and-take mechanism, a graphic interpretation of Algorithm 1. The τ_{min} is influenced by the $\tilde{\tau}_i$ of the most recently taken card. The red circle points out the card with the highest n_i within our hand, which will be replaced with the newly taken card in the next iteration.

Proposition 2: Algorithm 1 provides an optimal solution to Problem P2.

Proof: By searching for a feasible solution to Problem P3 in z's descending order and stop upon identifing one, the resulting solution $\mathcal{S}$ necessarily possesses the maximal object value of P2. In the following, we prove that the searching process is optimal during each iteration of z. Suppose $\mathcal{S}_d$ is the selected set of requests at any d's iteration. If $\mathcal{S}_d$ is feasible for P3, the optimal solution is found. If not, this implies violation of (14b) or (14c) due to a large sum of n_i or n_i^2. We then draw the card with the highest n_i. In the $(d+1)$-th iteration, if the newly taken card's n_i exceeds the drawn one, the constraints are still violated. Otherwise, feasibility is re-evaluated under the updated τ_{min}. The search proceeds towards changes in n_i and τ_{min}. If no solution is identified throughout the iterations of d, the objective value for P2 cannot be z. □

Complexity: Given that the time complexity to obtain the maximum is $O(I)$, the complexity of d iterations is $O(I^2)$, the overall complexity of Algorithm 1 is $O(I^3)$.

In this part, the optimization problem P1 is converted to a sequence of feasibility problems P2. Within each P2, we rank the requests first by $\tilde{\tau}_i$ and then by n_i to expedite the solution search. At its core, in the case of single edge node and multiple users with large bandwidth, the optimization revolves around metrics of n_i and $\tilde{\tau}_i$ under the optimization object of z. By crafting the search in the order of z, $\tilde{\tau}_i$, and n_i, our greedy approach efficiently achieves the optimal solution.

C. OT Algorithm: Single Edge Node & Limited Bandwidth

In this part, we incorporate uplink and downlink bandwidth constraints into Problem P3. This means that in addition to z, $\tilde{\tau}_i$ and n_i in Algorithm 1, we now need to integrate two additional metrics, $\rho^U_{i,min}$ and $\rho^D_{i,min}$, into the search. However, this elevates the solution space and the complexity, making the methodology of Algorithm 1 less intuitive. To address this, while retaining the priority of z and $\tilde{\tau}_i$ from Algorithm 1, we develop the OP algorithm. This algorithm builds a tree to methodically represent solution space consisted of bandwidth, memory, and latency constraints. Leveraging this tree, the search consistently gravitates towards branches that align best with the problem's constraints. We also introduce the tree-pruning technique to sidestep redundant branches.

Intuitively, given $|\mathcal{S}| = z$, the edge inference optimization problem under limited bandwidth introduces two additional constraints: $\rho^U_{i,min} \leq 1$ and $\rho^D_{i,min} \leq 1$ into Problem P3. However, a closer examination of $\rho^D_{i,min}$ in (2), reveals that the downlink transmit power of the edge node, denoted as p^D, remains consistent for all users. Furthermore, the edge node can obtain users' channel status via interference measurement techniques, such as CSI-RS, which means that h^D_i is already known to the edge node before inference scheduling. It is reasonable to assume that the edge node allocates downlink bandwidth to users based on the worst channel gain, mitigating potential issues where users might receive insufficient downlink bandwidth due to channel status fluctuation. As a result, we can reexpress the expression in (2) as $\rho^D_{i,min} \triangleq n_i/k_6$, where k_6 is a constant, identical for all users within the coverage of a given edge node. Hence, the edge inference optimization problem under limited bandwidth is as follows:

$$\text{P3}': \quad \text{find } \mathcal{S} \subseteq \tilde{\mathcal{I}} \tag{15}$$
$$\text{s.t.} \quad |\mathcal{S}| = z \tag{15a}$$
$$\sum_{i\in\mathcal{S}} \rho^U_{i,min} \leq 1 \tag{15b}$$
$$\sum_{i\in\mathcal{S}} n_i \leq k_6 \tag{15c}$$
$$\sum_{i\in\mathcal{S}} n_i \leq \tilde{M} \tag{15d}$$
$$\sum_{i\in\mathcal{S}} k_4 n_i + k_5 n_i^2 \leq \tilde{\tau}_i. \tag{15e}$$

Due to the added constraints of (15b) and (15c), solving Problem P3′ necessitates a more intricate searching strategy. To enhance search efficiency, we employ a tree-search method. Similar to Algorithm 1, we still start from the largest value of z, sort $\tilde{\mathcal{I}}$ based on $\tilde{\tau}_i$ in descending order, and iteratively search for a feasible solution under $\tilde{\tau}_d$. Denote the top d candidate requests as $\mathcal{F}_d$. We categorize $\mathcal{F}_d$ based on output length: N_1 represents the shortest length and N_{max} is the longest. We express $\mathcal{F}_d$ as $\mathcal{F}_d = \mathcal{F}_{N_1} \cup \mathcal{F}_{N_2} \cup \cdots \cup \mathcal{F}_{N_{max}}$, where $\mathcal{F}_{N_k} = \{i | i \in \mathcal{F}_d, n_i = N_k\}, k \in \{1, 2, \ldots, N\}$. Let $\mathcal{S}'$ define the set of z selected requests that will be further evaluated for adherence to communication and computation constraints. The set can be represented as $\mathcal{S}' = \mathcal{S}'_1 \cup \mathcal{S}'_2 \cup \cdots \cup \mathcal{S}'_N$, where $\mathcal{S}'_k = \{i | i \in \mathcal{F}_{N_k}\}, k \in \{1, 2, \ldots, N\}$. Each $\mathcal{S}'_k$ is a subset

of $\mathcal{F}_{N_k}$. With $\mathcal{F}_d$ established, we aim to identify a potential solution $\mathcal{S}'$ efficiently by constructing a search tree. As shown in Fig. 7, the methodology for this construction is as below.

1) Root node, parent node, and child node: The root node, v_0, has child nodes representing possible choices for $\mathcal{S}_1$. While $\mathcal{S}_1$ can be any subset of $\mathcal{F}_{N_1}$, requests with lower bandwidth requirements are prioritized. This means that for each integer value u between 0 and $|\mathcal{F}_{N_1}|$, only the choice of $\mathcal{S}_1$ that includes the top u requests with the smallest bandwidth is selected. Thus, the child nodes of the root node are sequentially indexed by $0, 1, \ldots, |\mathcal{F}_{N_1}|$. Bandwidth includes both uplink and dowlink bandwidth. Since the allocated downlink bandwidth ratio is proportional to n_i, constraint (15c) is applied using the same tree-search principle as (15d) and (15e), which will be discussed later. Here, we focus on (15b). Every $\mathcal{S}_1$ consists of $|\mathcal{S}_1|$ requests with minimum uplink bandwidth requirements from $\mathcal{F}_{N_1}$. Every $\mathcal{S}_1$ acts as a parent node for $\mathcal{S}_2$. Specifically, each $\mathcal{S}_1$ has $|\mathcal{F}_{N_2}| + 1$ child nodes. This is how the search tree is built layer by layer.

2) Path: An arbitrary node, $v_{N(v)}$, with its depth $N(v)$, indicates the cardinality of $\mathcal{S}_{N(v)}$, which is distinct among all its sibling nodes. We can then refer to a vector $\boldsymbol{p} = [v_0, v_1, \ldots, v_{N(v)}]$ to uniquely trace the path from the root node to $v_{N(v)}$. Hence, the path defines a partial solution $\mathcal{S}$ with $|S_{N(k)}| = v_k$, where $k = 1, 2, \ldots, N(v)$.

3) Leaf node: We determine a node $v_{N(v)}$ to be a leaf node without child nodes if it satisfies either of the two following conditions: (1) $\sum_{k=1}^{N(v)} v_k = z$, in which case the accumulated number of selected requests reaches z, suggesting that $\boldsymbol{p}$ represents a solution and its sum bandwidth should now be checked. (2) $\sum_{k=1}^{N(v)} v_k < z$ and $N(v) = N_{max}$, in which case $\boldsymbol{p}$ has reached the maximum depth but is not a solution. If none of the two conditions is satisfied, i.e., $N(v) < N_{max}$ and $\sum_{k=1}^{N(v)} v_k < z$, the node $v_{N(v)}$ has child nodes to be discovered.

The search tree represents the entire solution space for specified z and d. Each path within this tree represents a potential solution $\mathcal{S}$. We keep visiting nodes along paths until reaching leaf nodes. As we *visit node* $v_{N(v)}$, we intend to discover its potential child nodes, which would represent solutions to the following sub-problem:

$$\text{P3}'_{(v_{N(v)})}: \text{find } \mathcal{S}' = \bigcup_{k=N(v)+1}^{N} \mathcal{S}_k \tag{16}$$

$$\text{s.t.} \quad \sum_{k=N(v)+1}^{N} |\mathcal{S}_k| = z' \tag{16a}$$

$$\sum_{i\in\mathcal{S}'} \rho^U_{min,i} \le 1 - \rho^U_{min}(v_{N(v)}) \tag{16b}$$

$$\sum_{i\in\mathcal{S}'} n_i \le k_6 - k_6 \rho^D_{min}(v_{N(v)}) \tag{16c}$$

$$\sum_{i\in\mathcal{S}'} n_i \le \tilde{M} - M(v_{N(v)}) \tag{16d}$$

$$\sum_{i\in\mathcal{S}'} k_4 n_i + k_5 n_i^2 \le \tau_{min} - \tau_{min}(v_{N(v)}), \tag{16e}$$

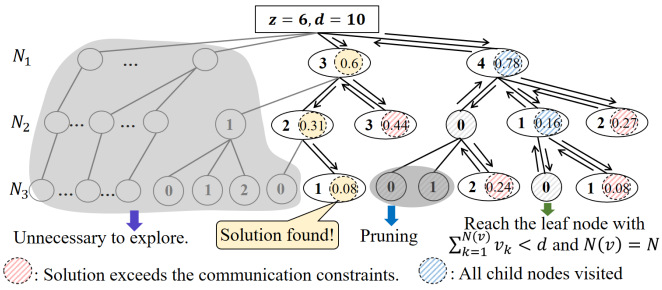


Fig. 7. An example of Algorithm 2. $|\mathcal{F}_{N_1}| = |\mathcal{F}_{N_2}| = 4, |\mathcal{F}_{N_3}| = 2$. All paths meet the memory and latency constraints. The number inside the dotted circle represents the accumulated uplink bandwidth requirements of $\mathcal{S}_k$.

where $z' = z - \sum_{k=1}^{N(v)} v_k$. $\rho^U_{min}(v_{N(v)}), \rho^D_{min}(v_{N(v)}), M(v_{N(v)})$, and $\tau_{min}(v_{N(v)})$ denote the accumulated sum demands for uplink bandwidth, downlink bandwidth, memory, and consumed latency along the path to $v_{N(v)}$, respectively.

Consider a search over the tree constructed for given z and τ_{min}. When dealing with large values of z and d, it is impractical to store the entire tree in memory and check the feasibility of all paths. This challenge is addressed by crafting an efficient *tree-search* algorithm. Central to this design are two imperative considerations: the order in which nodes should be explored and the methods to avoid unnecessary search complexity.

1) Tree-search: To answer the first question, two fundamental search strategies follow. Firstly, when exploring all child nodes of any node, preference is given to the one with the maximal index. Such an approach aligns with always selecting more requests which has fewer n_i, helping to bound the left hand side of constraints (16d) and (16e). Secondly, the searching direction should prioritize depth rather than breadth. This depth-oriented direction guarantees that the first leaf node is accessed rapidly, enabling the quick discovery of a viable solution $\boldsymbol{p}$. Specifically, upon visiting any node among all its unvisited child nodes, the next step is to visit the one with the largest index, increasing the search depth by 1. Upon reaching a leaf node or a node without any unvisited child node, we backtrack to visit its parent node and subsequently repeat the aforementioned steps.

2) Tree-pruning: To answer the second question, we adopt a *tree-pruning* technique for complexity reduction. For any given node $v_{N(v)}$, if $\sum_{k=N(v)}^{N_{max}} |\mathcal{F}_{N_k}| < z'$, it is unnecessary to visit $v_{N(v)}$ and its child nodes, as well as $v_{N(v)}$'s sibling nodes with indices smaller than $v_{N(v)}$ and their respective child nodes. Since $|\mathcal{F}_{N_k}|, k \in \{1, 2, \ldots, N\}$ are already known under fixed d, it is easy to prune, in an online manner, the tree branches that cannot meet $\sum_{k=N(v)}^{N_{max}} |\mathcal{F}_{N_k}| < z'$.

The whole process of the tree-search algorithm with online tree-pruning is shown in Algorithm 2.

Example: To illustrate the above algorithm, Fig. 7 shows the complete process of *tree-searching* and *tree-pruning* for Problem P3′ given $z = 6$, $d = 10$, and $N_{max} = 3$.

Complexity: Ordering nodes by bandwidth has a complexity of $O(I^2)$, hence the complexity of searching each path is $O(I^2 \max\{I, N\})$. There are $O((\frac{I}{N})^N)$ nodes in a tree, and thus the complexity of *DFS* is $O(I^2 \max\{I, N\}(\frac{I}{N})^N)$. The total

Algorithm 2 Optimal Tree-Search (OT)

Input: Available request set $\tilde{\mathcal{I}}$.
Output: Optimal solution $\mathcal{S}$ to Problem P2 with added communiation constraints.

Initialize $z = 0$, $d = 0$, $\mathcal{F}_d = \emptyset$, $\mathcal{S}' = \emptyset$;
for $z = |\mathcal{I}|, |\mathcal{I}| - 1, \ldots, 1$ **do**
  Sort $\tilde{\mathcal{I}}$ according to $\tilde{\tau}_i$'s descending order;
  for $d = z, z + 1, \ldots, |\mathcal{I}|$ **do**
    $\tau_{min} \leftarrow \tilde{\tau}_d$, $\mathcal{F}_d \leftarrow$ the first $d - 1$ requests in $\tilde{\mathcal{I}}$;
    Construct root node v_0 for $\mathcal{F}_d$ and the d-th request;
    Call DFS(v_0, d);
    if DFS(v_0, d) returns solution $\mathcal{S}'$ **then**
      $\mathcal{S} \leftarrow \mathcal{S}'$, return $\mathcal{S}$;
  return no solution
function DFS($v_{N(v)}, d$)
  $N(v) \leftarrow$ depth of $v_{N(v)}$;
  Visit the path to $v_{N(v)}$ along unvisited nodes;
  if $\sum_{k=1}^{N(v)} v_k = d$ **then**
    Recover the subset $\mathcal{S}'$ from $v_{N(v)}$;
    if $\mathcal{S}'$ meets constraints (15b)-(15e) **then**
      return $\mathcal{S}'$
    else
      Mark $v_{N(v)}$ visited;
      $v' \leftarrow v_{N(v)}$'s sibling node with the largest index;
      Call DFS(v', d);
  else if $\sum_{k=1}^{N(v)} v_k < d$ and $N(v) = N$ **then**
    Mark $v_{N(v)}$, its parent node, and its sibling nodes with lower index visited;
    $v' \leftarrow v_{N(v)}$'s parent node;
    Call DFS(v', d);
  else if $\sum_{k=1}^{N(v)} v_k < d$ and $N(v) < N$ **then**
    if all child nodes are visited **then**
      if $v_{N(v)}$ is the root node **then**
        return no solution
      else
        $v' \leftarrow v_{N(v)}$'s parent node;
        Call DFS(v', d);
    $v_{N(v)+1} \leftarrow \min\{z', |\mathcal{F}_{N(v)+1}|\}$;
    Call DFS($v_{N(v)+1}, d$);

time complexity of Algorithm 2 is $O(I^4 \max\{I,N\}(\frac{I}{N})^N)$. Since N is a constant much smaller than I, the complexity is polynomial in the number of I, even without considering complexity ruduction by pruning.

In this part, we propose the OT algorithm for edge inferene scheduling across a single edge node and multiple users. For a given z and τ_{min}, the OT algorithm structures the solution space as a tree, factoring in constraints like uplink bandwidth, downlink bandwidth, memory, and latency. It then performs a sequential depth-first tree-search with online tree-pruning to maximize throughput.

D. OT-GAH Algorithm:Multiple Edge Nodes & Limited Bandwidth

In this part, we shift our focus to edge inference scheduling across multiple edge nodes, an intricate scenario made by the interplay among various edge nodes. We address this complexity by modeling the original problem through the *Generalized Arrangement Problem (GAP)* [32], [33], [34]. The OT algorithm is empolyed to heuristically search for local-optimal solutions for each edge node.

The *GAP* is defined as follows: Given a set of knapsacks and a set of items, each knapsack has a distinct capacity. Each item possesses a size and a profit contingent upon the knapsack that it is placed in. The objective is to identify a subset of items that can feasibly be packed within the knapsacks, ensuring the profit is optimized. The Problem P1 can be represented as a GAP, in which edge nodes are treated as knapsacks and each item is with a multi-dimension size and a profit value of 1.

To address this, we leverage the *Next-Bin* algorithm in [32]. The *Next-Bin* employs the local-ratio technique, offering an innovative algorithmic framework that converts any algorithm designed for a single knapsack problem into an approximation algorithm for the GAP. Specifically, *Next-Bin* solves singular knapsack problems sequentially by utilizing any given algorithm tailored for the single knapsack problem. Once an item is designated for packing in knapsack j, its profit p_{j+1} is updated to $p_{j+1} - p_j$, and the item joins the scheduling in the subsequent knapsack.

Since the item's profit is the same across all knapsacks in Problem P1, which is equal to 1, the value $p_{j+1} - p_j = 0$. This implies that once an item is allocated to a knapsack, it cannot be placed elsewhere. Hence, the *Next-Bin* utilized in P1 adheres to the following procedure: for every individual edge node, Algorithm 2 is used to select requests into the edge node's batch. Requests not chosen are eligible for the next edge node's scheduling, and the process continues up to the final edge node. The complete scheduling process for Problem P1 is detailed in Algorithm 3.

Approximation Ratio: According to [32], if the approximation ratio of a single knapsack algorithm is θ, the *Next-Bin* algorithm guarantees a $\frac{1}{1+\theta}$-approximation ratio. Consequently, as Algorithm 2 is optimal, the approximation ratio for Algorithm 3 is $\frac{1}{2}$. This implies that the total edge inference throughput produced by Algorithm 3 surpasses half of the optimal throughput.

Complexity: Given that Algorithm 3 executes Algorithm 2 for J times, its time complexity is $O(I^4 J \max\{I, N\}(\frac{I}{N})^N)$, which is polynomial in terms of I and J, with J representing the number of edge nodes within a wireless network.

Algorithm 3 Optimal Tree-Search With Generalized Assignment Heuristics (OT-GAH)

Input: Available request set $\mathcal{I}$.
Output: Approximated solution $\mathcal{S}$ to Problem P1.

Initialize the solution for a single edge node $\mathcal{S}^{(j)} = \emptyset$;
for edge node $j \in \mathcal{J}$ **do**
  Apply Algorithm 2 to select $\mathcal{S}^{(j)}$ from $\mathcal{I}$;
  $\mathcal{I} \leftarrow \mathcal{I} \setminus \mathcal{S}^{(j)}$;
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}^{(j)}$;
return $\mathcal{S}$.

V. Simulations

In this section, we first illustrate the advantages of the proposed LLM edge inference optimization using dynamic batching over other batching strategies. Next, we demonstrate the impact of various model quantization methods on edge inference. Finally, we showcase the benefits of our proposed OT-GAH algorithm complemented with tree-pruning.

A. Simulation Settings

The experimental settings are as follows unless specified otherwise. The arrivals of multiple user requests are modeled as a Poisson process with the arrival rate λ varying from 5 to 250 requests per second [35]. The input prompt length is randomly selected from $\{128, 256, 512\}$ tokens, as well as the generation sequence length. We introduce the Byte-Pair Encoding (BPE) tokenization method widely used in most of LLMs [36], [37], where each token is represented by a 2-byte (16 bits) integer index. The required accuracy is uniformly selected from $[0, 1]$ [17], [28].

An edge server, an UAV, and a vehicle are placed in a wireless network. In the simulation, the edge server is assumed to be equipped with 20 NVIDIA JETSON TX2 GPUs, each with 1.33TFLOPs and 32GB memory. The edge server is placed together with a base station. The uplink and downlink bandwidth are 20MHz, the transmit power from users to the base station is 20dBm, and the transmit power from the base station to users is 43dBm [38]. $N_0 = -174$dBm/Hz. The channel gain follows Rayleigh fading with average path loss as 10^{-3} [19]. The UAV is assumed to carry a TX2 GPU and fly at the height of 100m, which is within the free-space path loss. The uplink and downlink bandwidth are both 1MHz. The transmit power between the UAV and a user is 10dBm [39], [40]. The vehicle is assumed to be equipped with three TX2 GPUs and receive the user requests from road-side small base stations. The uplink and downlink bandwidth are both 2MHz. The transmit power from the base station to the vehicle is 30dBm, and the transmit power from the vehicle to the base station in 23dBm. The pathloss and slow fading model follows the Manhatann grid layout [41]. The duration of an epoch is set as 2 seconds, with $T_U = T_D$ =250ms.

Two benchmarking schemes of batched inference are adopted [19], [35], [42]:

- *Static batching:* Edge node operates with a predefined batch size, which is established considering epoch duration, the implemented LLM, and the requisite input and output lengths to ensure there is no GPU overflow. During each epoch, the edge node aggregates prompts, either of the designated size or smaller, for parallel inference.
- *No batching:* Requests randomly arrive at edge nodes in uplink communication slots and wait in a first-in-first-out queue. When any GPU is idle, the edge node accepts the request at the queue head, executes inference on that single GPU without batching. During execution, if the accumulated uploading, waiting, and computing latency exceeds the request's deadline, it is dropped. Once a request is completed, the output is sent back.

TABLE I
LLM Settings in the Simulation

Model	Layer number	Dimension	Head number	Head dimension
BLOOM-3B [43]	30	2560	32	80
BLOOM-7.1B [44]	30	4096	32	128
OPT-13B [45]	40	5120	40	128

TABLE II
PPL Differential of Different Quantization Methods

Precision	Quantization method	BLOOM-3B	BLOOM-7.1B	OPT-13B
W4A16	GPTQ	0.75	0.54	0.2
	ZQ-Local	0.92	0.59	0.42

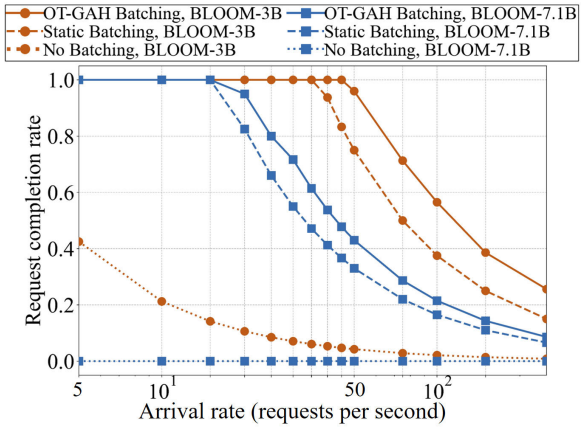


Fig. 8. Completion rate vs. arrival rate of LLM inference with different batching schemes.

A benchmarking algorithm is adopted to demonstrate OT-GAH's superiority. Unlike OT-GAH, it searches the entire solution space without the online tree-pruning technique.

Three open-source LLMs are adopted in this simulation, as detailed in Table I. All of them adopt the transformer decoder-based architectures. The FFN dimension is four times that of the model dimension.

In our simulation, we incorporate various model quantization techniques. As evaluated by [17], the quantization schemes using the precision of 8-bit weight and 16-bit activation (W8A16) have a negligible degradation in accuracy. Therefore, by default, we employ this W8A16 scheme on edge nodes to reduce memory usage. Other quantization methods and their performance on LLMs are evaluated in [17], and the details are shown in Table II.

The request completion rate denotes the proportion of successfully completed inference requests to the total number of arriving requests within an epoch.

B. Edge Inference With Different Batching Schemes

Fig. 8 compares batching scheme performance for BLOOM-3B and BLOOM-7.1B, with latency requirements between 1.5 and 2 seconds. Increased arrival rates decrease completion rates due to edge node resource limits. The OT-GAH scheme outperforms both static and no batching. This is because OT-GAH dynamically adjusts the batch size based on received

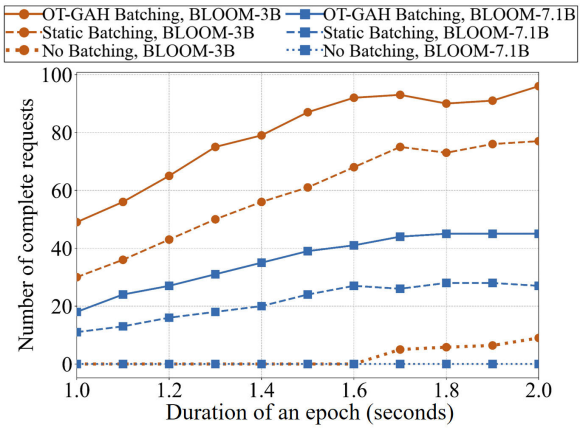


Fig. 9. Number of completed requests within one epoch vs. epoch duration. $T_U = T_D$ =250ms.

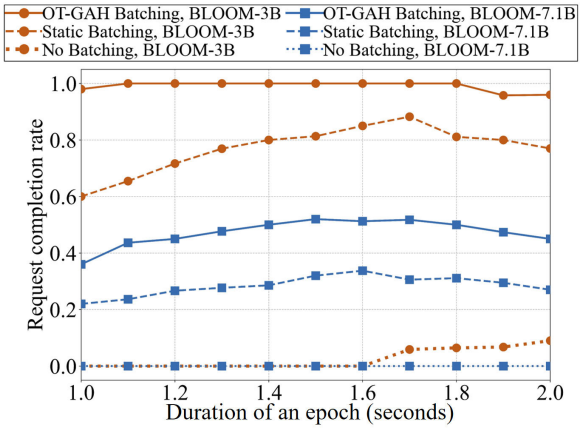


Fig. 10. Completion rate vs. epoch duration. $T_U = T_D$ =250ms.

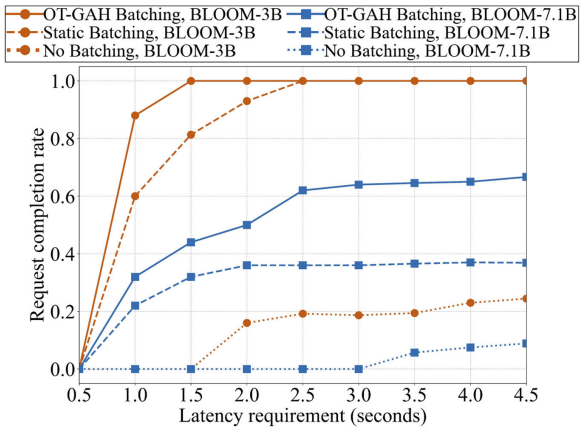


Fig. 11. Completion rate vs. latency requirement of LLM inference with different batching schemes.

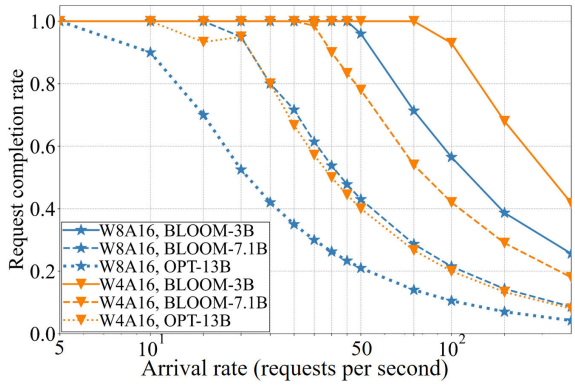


Fig. 12. Completion rate vs. arrival rate under LLM quantization with different precisions.

requests, whereas the static scheme processes a fixed number of requests at each epoch to prevent overflow. The no-batching scheme, in contrast, fails to leverage parallel computing. The single GPU's processing speed is quite limited, resulting in most requests being dropped. Furthermore, BLOOM-7.1B consistently exhibits a lower completion rate than BLOOM-3B across all batching schemes, as it possesses more parameters and thus requires greater resources during inference. No BLOOM-7.1B requests meet their deadlines due to this intense computational load.

Figures 9 and 10 highlight the impact of epoch duration settings on edge inference. With latency requirements between 1.5 and 2.5 seconds and an arrival rate of 50 requests per second, an increase in epoch duration correlates with a rise in completed request number per epoch. However, the completed rate initially grows but then decreases. This behavior can be attributed to the edge inference scheduling protocol we have developed. Specifically, beyond each request's latency requirement, the cumulative inference latency for all batched requests must remain under T_C. A larger T_C accommodates more requests in an epoch. But if excessively prolonged (as in 1.7s to 2.0s in Fig. 10), batched requests finish well before T_C ends, causing idle computational periods and a decreased completion rate. Interestingly, the no-batching method in BLOOM-3B, by processing requests in a first-in-first-out manner and continually employing computational resources, shows a slightly increasing completion rate.

Fig. 11 showcases how the request completion rate varies based on users' latency requirements. Epoch duration is equal to the latency requirement. Arrival rate is 50 requests per second. As latency expectations become more relaxed, more requests can be batched for inference within their set deadlines. BLOOM-3B consistently outperforms BLOOM-7.1B, attributed to its streamlined tensor dimensions and fewer matrix operations. In no-batching, a single GPU starts to satisfy the deadline for large-scale BLOOM-7.1B inferences when epoch duration and deadline exceed 3 seconds.

Overall, the OT-GAH dynamic batching significantly enhances the throughput of LLM edge inference. Concurrently, the inference protocol should adapt dynamically to both users and models.

C. Edge Inference With Different Quantization Schemes

In Fig. 12, we exclude user accuracy requirements to focus on quantization's effect on edge inference. With a 2-second epoch, 50 requests/sec arrival rate, 1.5-2 second deadlines, larger LLM model sizes with the same quantization yield fewer completed requests due to resource constraints. Additionally, reduced quantization precision translates to lesser memory

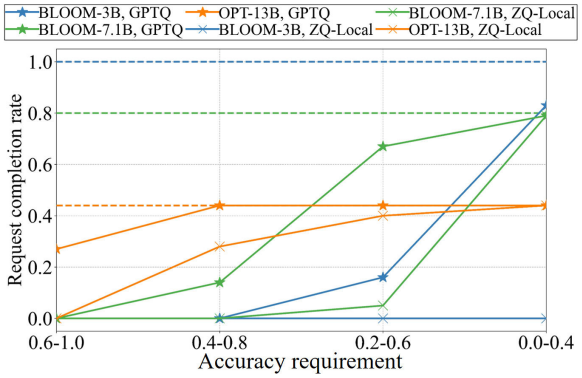


Fig. 13. Completion rate vs. accuracy requirement under different LLM quantization schemes.

TABLE III
ALGORITHM TIME COST AND TIME REDUCTION WITH TREE-PRUNING

Arrival rate	10	50	100	200
Time Cost	1.98ms	2.2ms	35ms	1.74s
Complexity Reduction	45.52%	71.18%	79.07%	97.92%

usage. Consequently, the completion rate for LLM inference sees a substantial rise when a more aggressive quantization scheme is in place.

However, this enhancement in completion rate carries an inherent trade-off, as depicted in Fig. 13. The relationship between provided output accuracy and PPL differential is $f(\Delta PPL) = 1 - \Delta PPL$. The dotted lines indicate the completion rate when the W8A16 quantization method is deployed. As a model undergoes more extensive quantization, there is a notable increase in PPL degradation, leading to diminished accuracy and, by extension, a reduced completion rate. Even with the same precision levels, LLM inference exhibits varied performance between GPTQ and ZQ-Local quantizations due to the distinct tensor operations each quantizes. Moreover, as users' accuracy requirements become more lenient, there is an increase in completion rate.

A joint analysis of Fig. 12 and Fig. 13 offers insights into the balance between accuracy and memory footprint reduction. This informs our strategy in selecting the most fitting quantization schemes tailored to specific LLMs and the needs of the target users.

D. OT-GAH Algorithm Performance

Finally, we examine the computational complexity of the proposed OT-GAH algorithm, designed for efficient tree-search, in contrast to the benchmark design that lacks tree-pruning. As presented in Table III, the time cost escalates with the increase in arrival rate yet remains quite affordable. The computational complexity of brute-force searching without pruning escalates exponentially as the arrival rate grows. The efficiency gained through the OT-GAH algorithm becomes increasingly pronounced with a rising arrival rate, even achieving a remarkable 97.92% reduction when the arrival rate reaches 200.

VI. CONCLUSION

In this paper, we have investigated the ubiquitous access to generative LLM services by leveraging wireless edge networks. We have studied the LLM edge inference optimization across multiple edge nodes and multi-users featuring dynamic batching, LLM quantization, and joint communication-and computation resource allocation for throughput maximization. We have proposed the OT-GAH algorithm, which can efficiently obtain the optimal solution to within a factor of $\frac{1}{2}$ in polynomial time by leveraging tree-search and tree-pruning strategies. Simulations results have demonstrated that OT-GAH not only surpasses brute-force tree-search algorithms by reducing complexity by over 45% but also outperforms benchmarking batching schemes on request completion rate across various user settings and inference settings. Furthermore, these results offer insights into the application of quantization schemes tailored for specific LLMs in resource-limited wireless edge networks. In future work, we will delve deeper into more scalable GAI offloading strategies. We will explore multi-modal GAI architectures, retaining lightweight components for local processing while offloading the rest to the edge. This approach will further enhance privacy protection.

REFERENCES

[1] C.-X. Wang, M. D. Renzo, S. Stanczak, S. Wang, and E. G. Larsson, "Artificial intelligence enabled wireless networking for 5G and beyond: Recent advances and future challenges," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 16–23, Feb. 2020.

[2] P. Hacker, A. Engel, and M. Mauer, "Regulating ChatGPT and other large generative AI models," in *Proc. FAccT*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1112–1123.

[3] Y. Huang et al., "AI-generated network design: A diffusion model-based learning approach," *IEEE Netw.*, vol. 38, no. 3, pp. 202–209, May 2024.

[4] Y. Huang et al., "Large language models for networking: Applications, enabling techniques, and challenges," *IEEE Netw.*, early access, 2024.

[5] X. Zhang, J. Liu, Z. Xiong, Y. Huang, G. Xie, and R. Zhang, "Edge intelligence optimization for large language model inference with batching and quantization," 2024, *arXiv:2405.07140*.

[6] W. Xin Zhao et al., "A survey of large language models," 2023, *arXiv:2303.18223*.

[7] M. Xu et al., "Unleashing the power of edge-cloud generative AI in mobile networks: A survey of AIGC services," 2023, *arXiv:2303.16129*.

[8] H. Du et al., "Enabling AI-generated content (AIGC) services in wireless edge networks," 2023, *arXiv:2301.03220*.

[9] H. Du et al., "Exploring collaborative distributed diffusion-based AI-generated content (AIGC) in wireless networks," *IEEE Netw.*, vol. 38, no. 3, pp. 1–8, May 2024.

[10] M. Schreiner. *GPT-4 Architecture, Datasets, Costs and More Leaked*. Accessed: Oct. 13, 2023. [Online]. Available: https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/

[11] D. Podell et al., "SDXL: Improving latent diffusion models for high-resolution image synthesis," 2023, *arXiv:2307.01952*.

[12] A. Vaswani et al., "Attention is all you need," in *Proc. NeurIPS*, vol. 30. Long Beach, CA, USA: Curran Associates, 2017, pp. 5998–6008.

[13] T. Schuster et al., "Confident adaptive language modeling," in *Proc. NeurIPS*, vol. 35. New Orleans, LA, USA: Curran Associates, 2022, pp. 17456–17472.

[14] Y. Sheng et al., "High-throughput generative inference of large language models with a single GPU," 2023, *arXiv:2303.06865*.

[15] M. Xu et al., "Joint foundation model caching and inference of generative AI services for edge intelligence," 2023, *arXiv:2305.12130*.

[16] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[17] Z. Yao, C. Li, X. Wu, S. Youn, and Y. He, "A comprehensive study on post-training quantization for large language models," 2023, *arXiv:2303.08302*.

[18] Y. Choi, Y. Kim, and M. Rhu, "Lazy batching: An SLA-aware batching system for cloud machine learning inference," in *Proc. HPCA*, Seoul, South Korea, 2021, pp. 493–506.

[19] Z. Liu, Q. Lan, and K. Huang, "Resource allocation for multiuser edge inference with batching and early exiting," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1186–1200, Apr. 2023.

[20] 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. OSDI*. Carlsbad, CA, USA: Advanced Computing Systems Association, 2022, pp. 521–538.

[21] L. Yunjiu, W. Wei, and Y. Zheng, "Artificial intelligence-generated and human expert-designed vocabulary tests: A comparative study," *Sage Open*, vol. 12, no. 1, pp. 1–12, Jan. 2022.

[22] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proc. NeurIPS*. Red Hook, NY, USA: Curran Associates, 2020, vol. 33, pp. 6840–6851.

[23] M. Sea. *Samsung Bans ChatGPT, AI Chatbots After Data Leak Blunder*. Accessed: Oct. 13, 2023. [Online]. Available: https://sea.mashable.com/tech/23460/samsung-bans-chatgpt-ai-chatbots-after-data-leak-blunder

[24] S. Cole. *ChatGPT Users Report Being Able To See Random People's Chat Histories*. Accessed: Oct. 13, 2023. [Online]. Available: https://www.vice.com/en/article/5d9zd5/chatgpt-users-report-being-able-to-see-random-peoples-chat-histories

[25] OpenAI. *ChatGPT4 Now With 50 Messages Every 3 Hours*. Accessed: Oct. 13, 2023. [Online]. Available: https://community.openai.com/t/chatgpt4-now-with-50-messages-every-3-hours/304697

[26] Antropic. *Claude 2*. Accessed: Oct. 13, 2023. [Online]. Available: https://www.anthropic.com/index/claude-2

[27] Bloc. *Future of RPG Games-Bannerlord and ChatGPT*. Accessed: Oct. 13, 2023. [Online]. Available: https://www.youtube.com/watch?v=akceKOLtytw

[28] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale," 2022, *arXiv:2208.07339*.

[29] *OpenAI Playground*. Accessed: Oct. 13, 2023. [Online]. Available: https://platform.openai.com/playground

[30] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, "Knapsack problems—An overview of recent advances. Part II: Multiple, multi-dimensional, and quadratic knapsack problems," *Comput. Operations Res.*, vol. 143, Jul. 2022, Art. no. 105693.

[31] W. Chang, Y. Xiao, W. Lou, and G. Shou, "Offloading decision in edge computing for continuous applications under uncertainty," *IEEE Trans. Wireless Commun.*, vol. 19, no. 9, pp. 6196–6209, Sep. 2020.

[32] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, Nov. 2006.

[33] T. Zhu, J. Li, Z. Cai, Y. Li, and H. Gao, "Computation scheduling for wireless powered mobile edge computing networks," in *Proc. INFOCOM*. Toronto, ON, Canada: IEEE, 2020, pp. 596–605.

[34] J. Li et al., "Budget-aware user satisfaction maximization on service provisioning in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 12, pp. 1–13, Dec. 2023.

[35] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," in *Proc. SC20*. Atlanta, GA, USA: IEEE, 2020, pp. 1–15.

[36] A. Rai and S. Borah, "Study of various methods for tokenization," in *Applications of Internet of Things* (Lecture Notes in Networks and Systems). Agartala, Tripura, India: Springer, 2021, pp. 193–200.

[37] F. Stollenwerk, "Training and evaluation of a multilingual tokenizer for GPT-SW3," 2023, *arXiv:2304.14780*.

[38] C. W. Zaw, N. H. Tran, Z. Han, and C. S. Hong, "Radio and computing resource allocation in co-located edge computing: A generalized Nash equilibrium model," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2340–2352, Apr. 2023.

[39] M. Hua, Y. Wang, Z. Zhang, C. Li, Y. Huang, and L. Yang, "Optimal resource partitioning and bit allocation for UAV-enabled mobile edge computing," in *Proc. IEEE 88th Veh. Technol. Conf.* Chicago, IL, USA: IEEE, Aug. 2018, pp. 1–6.

[40] M. Hua, Y. Wang, C. Li, Y. Huang, and L. Yang, "UAV-aided mobile edge computing systems with one by one access scheme," *IEEE Trans. Green Commun. Netw.*, vol. 3, no. 3, pp. 664–678, Sep. 2019.

[41] Y. Ku, P. Chiang, and S. Dey, "Real-time QoS optimization for vehicular edge computing with off-grid roadside units," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 11975–11991, Oct. 2020.

[42] NVIDIA. *NVIDIA AI Inference Platform Technical Overview*. Accessed: Oct. 13, 2023. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/t4-inference-print-update-inferen ce-tech-overview-final.pdf

[43] HuggingFace. *BLOOM-3B Model Card*. Accessed: Oct. 13, 2023. [Online]. Available: https://huggingface.co/bigscience/bloom-3b

[44] HuggingFace. *BLOOM-7B1 Model Card*. Accessed: Oct. 13, 2023. [Online]. Available: https://huggingface.co/bigscience/bloom-7b1

[45] S. Zhang et al., "OPT: Open pre-trained transformer language models," 2022, *arXiv:2205.01068*.

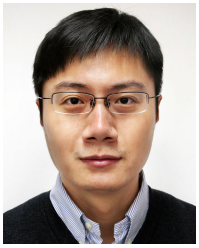
Xinyuan Zhang (Graduate Student Member, IEEE) received the B.S. degree in communication engineering from Beijing University of Posts and Telecommunications (BUPT), China, in 2019, and the Ph.D. degree from the State Key Laboratory of Networking and Switching Technology, BUPT, in 2024. She was a Visiting Ph.D. Student with the Information Systems Technology and Design Pillar, Singapore University of Technology and Design, Singapore, in 2022. Her current research interests include satellite-terrestrial integrated networks, edge intelligence, and generative AI.

Jiangtian Nie received the B.Eng. degree in electronics and information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2016, and the Ph.D. degree from ERI@N, Interdisciplinary Graduate School, Nanyang Technological University (NTU), Singapore, in 2021. She is currently a Research Fellow with NTU. Her research interests include network economics, game theory, crowd sensing, and learning.

Yudong Huang (Graduate Student Member, IEEE) received the B.S. and Ph.D. degrees in communication engineering from Beijing University of Posts and Telecommunications (BUPT), China, in 2019 and 2024, respectively. He was a Visiting Ph.D. Student with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, from 2022 to 2023. His current research interests include time-sensitive networks, deterministic networks, and network intelligence.

Gaochang Xie (Graduate Student Member, IEEE) received the B.E. degree from Shandong Agricultural University, China, in 2020. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China. He has been a Visiting Scholar with the Information Systems Technology and Design Pillar, Singapore University of Technology and Design, Singapore, since May 2023. His current research interests include edge intelligence and the Internet of Vehicles.

Zehui Xiong (Senior Member, IEEE) received the Ph.D. degree from Nanyang Technological University (NTU), Singapore. He is currently an Assistant Professor at Singapore University of Technology and Design, and also an Honorary Adjunct Senior Research Scientist with Alibaba-NTU Singapore Joint Research Institute, Singapore. He was the visiting scholar at Princeton University and the University of Waterloo. His research interests include wireless communications, the Internet of Things, blockchain, edge intelligence, and metaverse. Recognized as a Highly Cited Researcher, he has published more than 300 research papers in leading journals, and he has won over ten Best Paper Awards in international conferences. He is now serving as the editor or guest editor for many leading journals including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING, and IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING. He is the recipient of Forbes Asia 30u30, IEEE Asia Pacific Outstanding Young Researcher Award, IEEE Early Career Researcher Award for Excellence in Scalable Computing, IEEE Technical Committee on Blockchain and Distributed Ledger Technologies Early Career Award, IEEE Internet Technical Committee Early Achievement Award, IEEE TCSVC Rising Star Award, IEEE TCI Rising Star Award, IEEE TCCLD Rising Star Award, IEEE ComSoc Outstanding Paper Award, IEEE Best Land Transportation Paper Award, IEEE Asia Pacific Outstanding Paper Award, IEEE CSIM Technical Committee Best Journal Paper Award, IEEE SPCC Technical Committee Best Paper Award, IEEE Big Data Technical Committee Best Influential Conference Paper Award, and IEEE VTS Singapore Best Paper Award. He has served as the Associate Director of Future Communications Research and Development Programme, and Deputy Lead of AI Mega Centre.



Jiang Liu received the B.S. degree in electronics engineering from Beijing Institute of Technology, Beijing, China, in 2005, the M.S. degree in communication and information systems from Zhengzhou University, Zhengzhou, China, in 2009, and the Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, in 2012. He is currently a Professor with Beijing University of Posts and Telecommunications. His current research interests include network architecture, network virtualization, satellite networking, software-defined networking (SDN), information-centric networking (ICN), and network testbed.

Dusit Niyato (Fellow, IEEE) received the B.Eng. degree from the King Mongkut's Institute of Technology Ladkrabang (KMITL), Thailand, and the Ph.D. degree in electrical and computer engineering from the University of Manitoba, Canada. His research interests inlcude mobile generative AI, edge intelligence, decentralized machine learning, and incentive mechanism design. He is currently a Professor with the College of Computing and Data Science, Nanyang Technological University, Singapore.

Xuemin Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990.

He is currently an University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests include network resource management, wireless network security, the Internet of Things, 5G and beyond, and vehicular networks. He is a registered Professional Engineer in ON, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Member, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society. He received the West Lake Friendship Award from Zhejiang Province in 2023, the President's Excellence in Research from the University of Waterloo in 2022, the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R. A. Fessenden Award in 2019 from IEEE, Canada, the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, the James Evans Avant Garde Award from the IEEE Vehicular Technology Society in 2018, the Joseph LoCicero Award in 2015, the Education Award from the IEEE Communications Society (ComSoc), in 2017, and Technical Recognition Award from the Wireless Communications Technical Committee in 2019 and the AHSN Technical Committee in 2013. He has also received the Excellent Graduate Supervision Award from the University of Waterloo in 2006 and the Premier's Research Excellence Award (PREA) in ON, Canada, in 2003. He serves/served as the General Chair for the 6G Global Conference'23 and ACM Mobihoc'15, the Technical Program Committee Chair/Co-Chair for IEEE Globecom'24, 16, and 07, IEEE Infocom'14, and IEEE VTC'10 Fall, and the Chair for the IEEE ComSoc Technical Committee on Wireless Communications. He was the President of the IEEE ComSoc, the Vice President of Technical & Educational Activities, the Vice President for Publications, the Member-at-Large on the Board of Governors, the Chair of the Distinguished Lecturer Selection Committee, and a Member of the IEEE Fellow Selection Committee of the ComSoc. He served as the Editor-in-Chief for IEEE INTERNET OF THINGS JOURNAL, *IEEE Network*, and PPNA.