Ygg: Tree-Based Collaborative Speculative Decoding with Token-Only Transmission

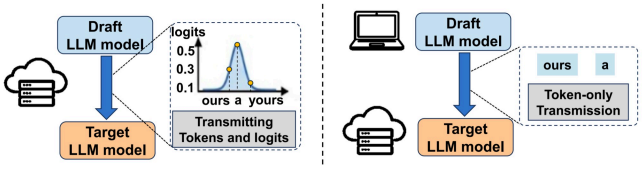
Sijia Li[†‡], Yumeng Liang[‡], Ruiqi Li[§], Jianjiang Li[†*], and Jie Wu[‡¶*]
[†]University of Science and Technology Beijing, Beijing, China
[‡]China Telecom Cloud Computing Research Institute, Beijing, China
[§]Peking University, Beijing, China
[¶]Temple University, USA
lisijia@xs.ustb.edu.cn, liangym9@chinatelecom.cn, rqli25@stu.pku.edu.cn, lijianjiang@ustb.edu.cn, jiewu@temple.edu

***Abstract*—While Large Language Models (LLMs) have exhibited remarkable performance, the QoS for the LLM inference system is still constrained by the substantial latency inherent in autoregressive decoding. To accelerate the decoding process, speculative decoding has recently been proposed, employing both a lightweight draft model and the target LLM residing in the cloud to rapidly generate candidate tokens and verify these candidates in parallel, enabling the processing of multiple tokens per step. However, co-deploying the draft and target models in the cloud wastes valuable GPU memory resources and limits the cloud's ability to perform parallel inferences for higher throughput. To further improve inference QoS by achieving higher throughput, in this paper, we propose Ygg to offload the draft model onto the cost-effective edge device, relieving the computational workload of the cloud in collaborative manners. To address the drawback of excessive data transmission in existing speculative decoding methods, we design a novel collaborative paradigm with token-only transmission, which drastically cuts down communication latency. Furthermore, Ygg incorporates two additional key designs: budget-aware adaptive tree-based draft generation and asynchronous pipelining, which serve to elevate the acceptance rate of generated draft tokens and boost inference throughput, respectively. Experimental results demonstrate that our proposed framework achieves up to 1.94× throughput improvement and reduces serving costs by 47.3% compared to the cloud-only speculative decoding method. Furthermore, in comparison with the state-of-the-art edge-cloud collaborative method, Ygg reduces the overall inference latency by up to 49.3% and consistently achieves the best performance across diverse tasks and varying LLM scales.**

***Index Terms*—LLM, Collaborative Inference, Speculative Decoding, QoS of LLM inference**

I. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in reasoning and text generation [1], [2], but the QoS for the LLM inference system is still constrained by the significant computational and latency burdens of sequential auto-regressive decoding on cloud infrastructure [3]. To accelerate the decoding process so as to improve the QoS of LLMs inference, *speculative decoding* has recently been proposed [4]–[6], in which both the lightweight draft model and the target LLM are deployed on the cloud. The draft model rapidly generates candidate tokens, while the target model verifies these candidates in parallel, enabling multiple tokens to be processed in a single step. Although such a draft-target model collaboration achieves significant acceleration gains, *the additional deployment of the draft model on the cloud consumes valuable cloud GPU memory,* which restricts large-scale parallel processing and thereby limits further improvement of inference throughput.



(a) Existing cloud-only speculative decoding (b) Our collaborative speculative decoding

Fig. 1: Illustration of prior speculative decoding methods and ours. Prior methods deploy both draft and target LLMs on the cloud to generate candidate tokens and verify them. Whereas we offload the draft model to an inexpensive edge device to alleviate the cloud GPU burden and design a token-only transmission mechanism to reduce the communication latency.

Luckily, the computing power of edge GPUs has been growing steadily, which is highly cost-effective compared with cloud servers [7], creating a promising opportunity to alleviate the cloud-side inference burden [8]. Given the draft model's moderate demands of computation resources and GPU memory, in this paper, we envision *whether we can offload the lightweight draft model to inexpensive edge devices, and improve the inference QoS by achieving higher throughput in an edge-cloud collaborative manner*. In this way, the valuable GPU memory on the cloud could be saved for the target LLMs to process more samples in parallel, so as to improve the overall throughput.

However, collaborative speculative decoding faces following unique challenges in achieving satisfactory acceleration:

(i) Foremost, in conventional speculative decoding, the large volume of data transmitted between the draft and target

Sijia Li and Yumeng Liang contributed equally to this work.
*: Corresponding authors.

979-8-3195-1703-6/26/$31.00 ©2026 IEEE

models introduces substantial communication latency. The candidate tokens generated by the draft model, as well as their associated probability distributions (logits), are required to be transmitted to the target LLM model. The high-dimensional logits for every drafted token incurs substantial communication latency overhead, which even outweigh the performance gains brought by the collaboration itself. Although some existing methods [9], [10] attempt to alleviate this issue by splitting the verification phase or transmitting only top-k sparse logits, the high-dimensional logits remains a critical bottleneck for communication latency. *In this paper, we rethink the form of speculation itself, and employ a token-only mechanism to perform speculative decoding without transmitting logits.* As illustrated in Fig. 1, our token-only communication significantly reduces transmission overhead, thereby achieving higher throughput gains.

(ii) Although transmitting fewer tokens can reduce communication latency, the resulting drop in the acceptance rate of candidate tokens degrades the overall acceleration performance. The acceleration performance of speculative decoding depends on the acceptance probability of the candidate tokens generated by the draft model during the verification of the target LLM, *i.e.*, any rejected token must be regenerated, which directly impairs the end-to-end speedup. When transmitting fewer candidate tokens, the hit rate of the draft model will drop, leading to more frequent token regeneration. Thus, striking a balance between communication overhead and acceptance rate to achieve stronger overall acceleration becomes a critical problem. To tackle this challenge, we regard the generation of sequential candidate tokens as the construction of a tree structure and propose the *budget-aware adaptive tree-based draft generation scheme*. Instead of aggressively generating a large number of redundant nodes, we dynamically select the highest-confidence nodes from a global buffer and control the total number of nodes using a budget constraint. In this way, we can achieve a high candidate token acceptance rate with a small transmission volume.

(iii) Collaboration between draft generation on the edge and verification on the cloud tends to introduce bubbles, which limits the improvement of overall inference speed. In traditional speculative decoding, the draft and target models work in a serial manner, avoiding the issue of excessive cloud memory occupation caused by parallel processing. However, such a concern is nonexistent in our collaboration mode, and the pipeline parallel mode can better utilize the resources of both sides to achieve acceleration. Nevertheless, resource disparities and model differences between the edge and cloud during parallel processing cause inconsistent execution speeds, introducing bubbles. To address this, we design an *asynchronous pipelining* method that adjusts the number of draft tokens based on cloud verification latency, fully utilizing resources and achieving higher acceleration.

The corresponding methods to address the above challenges are integrated into a framework named *Ygg*[1], which enables tree-based collaborative speculative decoding with token-only transmission. We implemented Ygg in the real edge-cloud testbed and evaluate its performance via extensive experiments. Experimental results show that Ygg achieves a 1.94× throughput gain compared with the cloud-side speculative decoding method, and reduce the cost by 47.3% by leveraging an inexpensive edge device. In comparison with other state-of-the-art edge-cloud collaborative speculative decoding methods, our approach cuts the inference latency by 49.3%, which validates the effectiveness of the proposed method.

Our contributions are summarized as follows:

- We propose a communication-efficient edge–cloud speculative decoding framework Ygg to improve the LLMs inference QoS. It eliminates the transmission of draft logits by introducing token-only, tree-structured speculation.
- A budget-aware adaptive tree-based draft generation scheme is proposed to improve acceptance rates of candidate tokens under tight token budgets. And an asynchronous pipelining mechanism is designed to improve inference throughput.
- Extensive experiments on the real edge-cloud testbed validate that Ygg can achieve up to 1.94× speedup of throughput and outperforms the other existing collaborative speculative decoding methods.

II. Background and Motivation

A. Speculative Decoding

LLMs generate text autoregressively, which is often bounded by memory rather than arithmetic intensity [11], [12]. To accelerate this, speculative decoding [4]–[6] introduces a lightweight draft model M_q to generate γ candidate tokens, which are then verified in parallel by the target model M_p. In this way, the target model can leverage multiple candidate tokens rapidly generated by the lightweight model to batch-process and generate tokens for multiple positions in one step.

The core mechanism of speculative decoding is how to verify the candidate tokens generated by the draft model. Let $P_i(x)$ and $Q_i(x)$ denote the probability distributions of the target and draft models at the i-th step, respectively. The verification process consists of two key phases:

a) Accept/Reject: For a drafted candidate token x_i, let $p_i(x_i)$ and $q_i(x_i)$ denote probability values within $P_i(x)$ and $Q_i(x)$. The target model evaluates the acceptance probability:

$$\Pr(\text{accept } x_i) = \min\left(1, p_i(x_i)/q_i(x_i)\right). \tag{1}$$

If accepted, the token is retained. Otherwise, it is rejected, and the process moves to the resampling phase.

b) Resample: To ensure the output strictly follows the target distribution, a rejected token is replaced by a resampled new token x_i' from the rectified distribution:

$$x_i' \sim \text{norm}\left(\max\left(0, P_i(x) - Q_i(x)\right)\right). \tag{2}$$

[1]Named after Yggdrasill, the world tree in Norse mythology that connects realms of different worlds.

This mechanism guarantees the identical distribution as that of the target model M_p, as long as it has access to the full conditional distribution $Q_i(x)$ of the approximation. Therefore, in traditional speculative decoding methods, the candidate tokens as well as the corresponding logits are sent to the target LLMs model, as shown in Fig. 1 (a). This mechanism incurs little latency within the cloud, yet becomes a communication bottleneck in edge-cloud collaborative inference.

B. The Communication Bottleneck in Edge-Cloud Collaborative Speculative Decoding

In an edge-cloud collaborative setting, placing the draft model on the edge and the target model on the cloud is a promising architecture [13], [14]. However, applying standard speculative decoding creates a communication bottleneck. Calculating Eq. (2) necessitates transmitting the probability distribution for every draft token. For a vocabulary size V (typically 32k–128k) and draft length N, this results in a transmission payload of $N \times V$, often reaching 1MB per step. While this data transfer overhead is negligible in standard SD deployments within high-bandwidth cloud clusters (*e.g.*, via NVLink), it becomes a prohibitive bottleneck in edge-cloud scenarios operating over wireless networks [15]. Although recent optimization strategies such as DSSD [9] and Top-K Sparse Logits Transmission (TK-SLT) [10] attempt to mitigate this overhead by splitting verification phases or transmitting sparse logits, they do not avoid transmitting probability distributions. In this paper, we rethink the form of speculation and employ a novel speculative decoding mechanism with token-only transmission to reduce the communication overhead.

III. Methodology

A. Kernel Idea: Naive Sampling and Tree Attention

To ensure the generated output remains distributionally consistent with the target model without accessing draft logits, we employ *Naive Sampling* (NS) [16] for verification. Specifically, the target model M_p independently computes the target distribution $P_i(x)$ and samples a verification token x' accordingly. The draft token x_i is accepted if and only if it strictly matches x':

$$\text{Accept}(x_i) = \mathbb{I}(x_i = x'), \quad \text{where } x' \sim P_i(x). \tag{3}$$

Although efficient, NS has a low acceptance rate, especially when few candidates are generated for each position, because of its strict matching criterion. Transmitting many candidates causes high communication overhead, so a strategy is needed to balance the acceptance rate and transmission cost.

To recover the acceptance rate under low communication cost, we introduce a *tree-based drafting strategy*. Instead of producing a single sequence, the draft model M_q constructs a token tree $\mathcal{N}$ covering multiple plausible paths. Using tree attention [16]–[18], the target model M_p verifies the entire tree in one forward pass. Specifically, the topology-aware mask $\mathbf{M}$ allows a query token u to attend to a key token v only if v is an ancestor of u:

$$\mathbf{M}_{uv} = \begin{cases} 0, & \text{if } v \in \text{Ancestors}(u) \cup \{u\}; \\ -\infty, & \text{otherwise.} \end{cases} \tag{4}$$

By applying this topology-aware mask, the verification complexity for a tree of size $|\mathcal{N}|$ remains $O(1)$ through parallel computation, ensuring that the latency cost is comparable to verifying a single chain.

We implement the above idea in a well-designed edge-cloud collaborative speculative decoding frame. In the following, we separately introduce the framework overview and the two kernel designs.

B. Framework Overview

This section presents our communication-efficient framework Ygg, designed for edge-cloud collaborative speculative decoding. To overcome the critical communication bottleneck in standard speculative decoding caused by the transmission of high-dimensional probability distributions, we propose an token-only communication protocol. By combining budget-aware adaptive tree construction on the device side with a pipelined draft verification mechanism, the proposed framework enables fast edge–cloud collaborative inference. As illustrated in Fig. 2, the framework operates in two main stages connected by a lightweight protocol:

Edge-Side Draft. The process initiates on an edge device hosting a lightweight draft model. Instead of generating a single sequence, the drafter constructs a token tree to maximize acceptance rate. As shown in the Draft panel, the model outputs token probability distributions $(p_0, p_1, \dots)$ from which multiple candidate tokens are sampled. Crucially, candidates not immediately selected for the current tree expansion are retained in a global candidate buffer pool. These buffered tokens compete in subsequent selection rounds, allowing high-confidence candidates from previous steps to be reconsidered, thereby constructing a high-quality tree topology.

Token-Only Communication. To break the bandwidth bottleneck, we avoid transmitting probability distributions. Instead, the device packages the drafted tree into a lightweight speculation packet $\mathcal{P}$, containing only discrete integer indices:

$$\mathcal{P} = (\mathbf{t}, \mathbf{p}, \mathbf{pos}), \tag{5}$$

where $\mathbf{t}$ represents the token IDs, $\mathbf{p}$ denotes the parent indices defining the tree topology, and $\mathbf{pos}$ indicates the position IDs for alignment.

Cloud-Side Verify. The packet $\mathcal{P}$ is transmitted to the cloud server hosting the large-parameter target model. Upon receipt, the target model constructs a tree attention mask based on $\mathbf{p}$ to compute logits $(q_0, q_1, \dots)$ for all nodes in a single forward pass. Since draft logits are unavailable, we employ NS verification strategy. The cloud performs path-based dynamic verification starting from the root: it samples a target token t' from its computed distribution and compares it against the drafted child node. If t' matches, verification proceeds to the next level; otherwise, the process terminates, and t' is returned

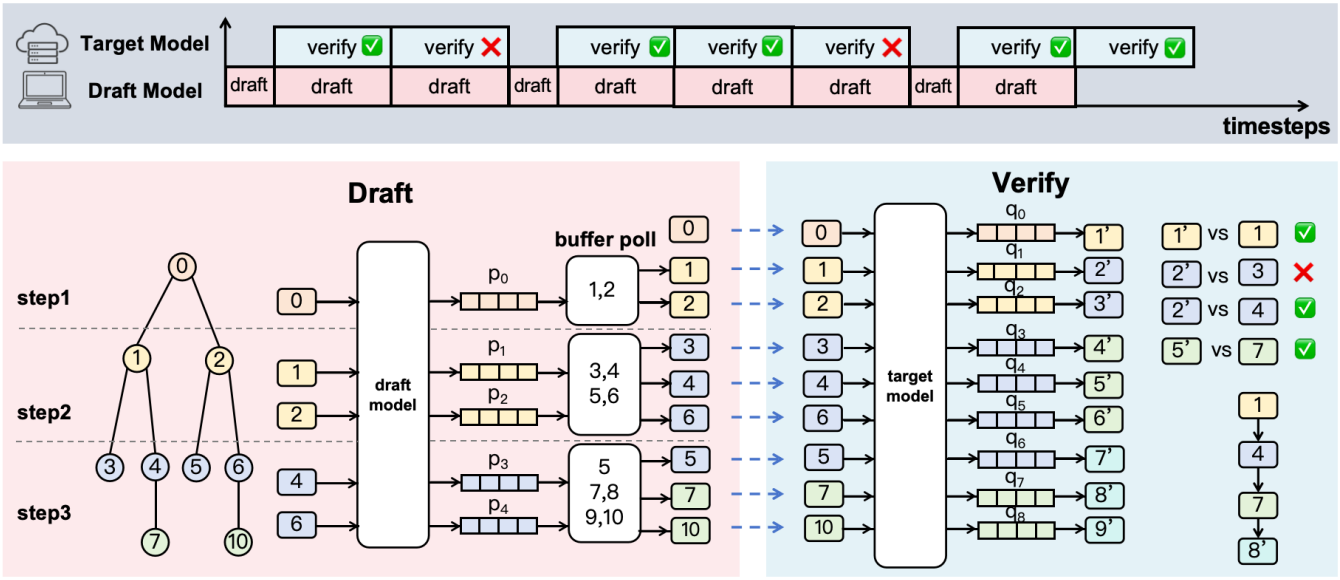


Fig. 2: Overview of Ygg, which adopts edge-cloud collaborative speculative decoding via token-only transmission. The draft model sequentially generate tokens for different positions in multiple steps (*e.g.*, yellow nodes 1–2, blue nodes 3–6). At each step, multiple candidates are produced; the top-k ones with higher confidence scores are selected dynamically, while others remain in a global buffer pool. Only the selected tokens are finally transmitted to the cloud without logits, which are compared with the target LLMs's output(*e.g.*, target $2'$ rejects draft 3, while $5'$ accepts 7) in parallel. The top timeline illustrates Asynchronous Pipelining, where the edge pre-drafts future tokens during cloud verification to mask latency bubbles.

as a corrective bonus token. The cloud then sends back a response $\mathcal{R}$ containing the verified path.

As depicted in the top timeline, the framework employs an asynchronous execution model. During the cloud-side verification phase, the device does not idle; instead, it concurrently executes a draft phase for the next round. If the verified path from the cloud matches the prefix of this concurrent draft, the waiting period draft is immediately activated and transmitted as the new request, effectively masking the drafting latency.

C. Budget-Aware Adaptive Tree Construction

To compensate for the reduced acceptance rate of NS, the device must construct a high-quality token tree $\mathcal{T}$ that covers the most probable outputs of the target model. We propose a *Budget-Aware Adaptive Tree Construction* algorithm. Unlike existing approaches that either lack lateral expansion [19] or introduce excessive computational redundancy [20], our approach utilizes normalized probabilities as confidence scores and introduces a global candidate buffer to enable cross-layer competition, addressing the inefficiencies of redundancy and the lack of lateral expansion in existing methods.

Problem Formulation. Prior research demonstrates a significant positive correlation between the output probability of the draft model and the probability of acceptance by the target model. Consequently [20], we utilize the predictive probability of the draft model as a confidence proxy to estimate the likelihood of a generated token being ultimately accepted. Let $\mathcal{V}$ denote the vocabulary. Given a prefix context sequence $x_{1:t}$, the draft model M_d predicts the next token probability distribution via a softmax operation:

$$P(x_{t+1}|x_{1:t}) = \text{Softmax}(\text{Logits}(M_d(x_{1:t}))). \tag{6}$$

We define the *Token Tree* $\mathcal{T}$ as a set of nodes, where each node u represents a token sequence $x_{1:t+d}$ extending from the root. Considering the sequential dependence inherent in the verification process, where the acceptance of node u necessitates the acceptance of all its ancestor nodes, the probability of the path being accepted by the target model approximates the joint probability of all tokens along the trajectory. Accordingly, we adopt the Cumulative Log-Probability (CLP) as the metric $S(u)$ to quantify the quality of node u:

$$S(u) = \sum_{k=1}^{d} \log P(x_{t+k}|x_{1:t+k-1}). \tag{7}$$

Our objective is to construct a tree $\mathcal{T}$ subject to strict memory and computational overhead constraints defined by $|\mathcal{T}| \leq K$, where K represents the token budget. Specifically, the tree is designed to include the set of nodes with the highest $S(u)$ values to maximize the coverage of high-confidence regions within the draft model's predicted distribution, thereby enhancing the speculation success rate of speculative decoding.

Algorithm. Existing tree construction algorithms typically perform layer-wise expansion, selecting the top-k nodes at depth d to expand to $d+1$ while immediately discarding the remainder. We identify two critical limitations of this paradigm within the context of edge-cloud collaborative inference. First, the strategy of generating a large tree before pruning to locate the optimal path introduces excessive computational

Algorithm 1: Budget-Aware Global Adaptive Tree Construction

Input: Prefix $x_{1:t}$, Budget K, Max Depth D, Batch Size B, Branch Width W, Admit Rate N_{limit}
Output: Token Tree $\mathcal{T}$

Initialize Tree $\mathcal{T} \leftarrow \{x_{1:t}\}$, $Q \leftarrow \{(0, \text{root})\}$, Buffer $\mathcal{B} \leftarrow \emptyset$;
while *depth* $< D$ ***and*** $|\mathcal{T}| < K$ **do**
  // 1. Batch Expansion;
  Pop batch of parents $\mathcal{P}_{batch}$ from Q (size $\leq B$);
  Compute logits for $\mathcal{P}_{batch}$ via Draft Model;
  for *parent* $u \in \mathcal{P}_{batch}$ **do**
    Get top-W children $\{v_1, \ldots, v_W\}$ via LogSoftmax;
    Calculate scores: $S(v_i) \leftarrow S(u) + \log P(v_i|u)$;
    Add children to Buffer: $\mathcal{B} \leftarrow \mathcal{B} \cup \{v_1, \ldots, v_W\}$;
  // 2. Global Selection (Lateral Comparison);
  Sort $\mathcal{B}$ descending by score $S(\cdot)$;
  $N_{admit} \leftarrow \min(|\mathcal{B}|, N_{limit}, K - |\mathcal{T}|)$;
  $\mathcal{C}_{best} \leftarrow$ Top N_{admit} nodes from $\mathcal{B}$;
  Remove $\mathcal{C}_{best}$ from $\mathcal{B}$;
  // 3. Tree Update;
  for *node* $v \in \mathcal{C}_{best}$ **do**
    Add v to $\mathcal{T}$;
    Push v to Q for future expansion;
  if $N_{admit} = 0$ **then**
    break;
return $\mathcal{T}$

redundancy. On resource-constrained edge devices, calculating logits for nodes that are ultimately pruned constitutes a significant waste of computational power. Second, confidence decay manifests in two dimensions: depth (where probability decreases as sequence length increases) and breadth (where the probability of a Rank-2 token is lower than that of a Rank-1 token within the same layer). Existing methods tend to prioritize depth, often exclusively extending the optimal branch. However, a high-probability node situated at a shallow layer may possess a higher global likelihood $S(u)$ than a node at a deeper layer where confidence has degraded. Neglecting these shallow nodes results in a tree structure that fails to achieve global optimality, as seen in Fixed-Shape and Layer-wise expansions (Fig. 3(a) and (b)).

To address these drawbacks, we propose implementing lateral expansion via a global buffer pool, achieving the Global Adaptive Expansion shown in Fig. 3(c). Instead of restricting comparisons to within the same hierarchy, we allow candidate nodes from shallow layers to compete directly with those from deeper layers. Consequently, if a deep branch traverses into a low-confidence region, the algorithm dynamically pivots to expand previously buffered high-probability nodes from shallower layers.

The detailed procedure is outlined in Algorithm 1, which

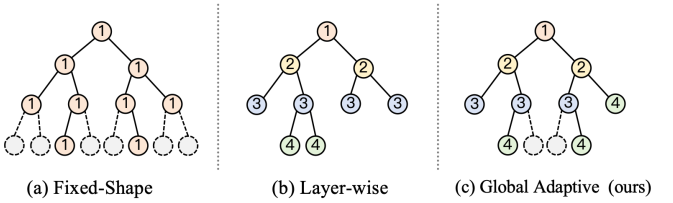


Fig. 3: Comparison of tree expansion strategies. (a) Fixed-Shape expansion first expands aggressively into a large, regular tree and then prunes it, wasting computation. (b) Layer-wise expansion selects only the best node(s) within the current layer at each step. (c) Our global adaptive expansion dynamically chooses the highest-confidence nodes from a global buffer, optimizing the resulting token-tree structure.

orchestrates the generation process through two primary components: a *Global Candidate Buffer* ($\mathcal{B}$) and a *Priority Queue* (Q). The process commences with the initialization of the tree $\mathcal{T}$ containing the input prefix, while Q is seeded with the root node to regulate the expansion sequence. In each iterative cycle, a batch of parent nodes is retrieved from Q to undergo a parallel forward pass via the draft model. For each parent, the probabilities of the top-W child tokens are computed. Crucially, rather than being immediately appended to the tree, these potential nodes are deposited into the Global Candidate Buffer $\mathcal{B}$ alongside their cumulative scores $S(u)$. Subsequently, a global selection mechanism is employed, wherein the entire buffer—comprising both newly generated candidates and unselected options from previous layers—is sorted by score.The top-N_{limit} nodes are admitted into $\mathcal{T}$ and moved from $\mathcal{B}$ to Q for subsequent expansion, while unselected nodes remain in $\mathcal{B}$. This enables lateral exploration by preserving high-probability shallow candidates when deeper paths become less promising.

D. Asynchronous Pipelining

While the pure index protocol minimizes transmission latency, the computational latency inherent in target model verification is unavoidable, resulting in an idle window (or "bubble") where the edge device awaits the response (Fig. 4). To bridge this gap and fully exploit edge computational resources, we propose a *Batched Pre-Drafting* mechanism. This approach enables the edge to speculatively generate potential drafts for the subsequent round while concurrently awaiting verification. Upon receiving the cloud's response, if a finalized path aligns with a branch that was speculatively extended during the waiting period, the system can bypass the drafting phase for the next cycle and immediately transmit the pre-computed tree. The overall procedure is summarized in Algorithm 2.

Candidate Tree Initialization. Rather than simply deepening the current token tree $\mathcal{T}_{curr}$, we opt to pre-construct independent *Candidate Trees* rooted at the high-confidence leaf nodes of $\mathcal{T}_{curr}$. This strategy ensures that the concurrently generated candidate trees structurally align with the drafts that would otherwise be generated *post-verification*, effectively

Algorithm 2: Batched Pre-Drafting with Asynchronous Verification

Input: Current tree $\mathcal{T}_{curr}$, draft model $\mathcal{M}_d$, cloud model $\mathcal{M}_c$, top-k k

send $\mathcal{T}_{curr}$ to Cloud;
async (Cloud): $(\mathcal{R}, l_{acc}) \leftarrow \text{Verify}(\mathcal{T}_{curr}, \mathcal{M}_c)$;
async (Edge): $\mathcal{T}' \leftarrow \text{Expand}(\mathcal{T}_{curr}, \mathcal{M}_d, 1)$;
$\mathcal{L} \leftarrow \text{TopKLeaves}(\mathcal{T}', k)$;
$\mathbb{S} \leftarrow \{\text{InitTree}(l) : l \in \mathcal{L}\}$;
repeat: $\mathcal{B} \leftarrow \text{Frontier}(\mathbb{S})$;
$\mathbb{S} \leftarrow \text{BatchExpand}(\mathcal{B}, \mathcal{M}_d)$;
until $\mathcal{R}$ received;
$l_{acc} \leftarrow \text{AcceptedLeaf}(\mathcal{R})$;
$\mathcal{T}_{next} \leftarrow \begin{cases} \text{Retrieve}(\mathbb{S}, l_{acc}), & l_{acc} \in \mathcal{L} \\ \text{StandardDraft}(l_{acc}, \mathcal{M}_d), & \text{otherwise} \end{cases}$;
send $\mathcal{T}_{next}$ to Cloud if $l_{acc} \in \mathcal{L}$;

anticipating the cloud's acceptance decision. This parallelization aims to synchronize the drafting of the next round with the verification of the current round, thereby eliminating the bubble caused by the cloud waiting for the edge's drafting phase. Let $\mathcal{L} = \{l_1, l_2, \ldots, l_N\}$ denote the set of leaf nodes in $\mathcal{T}_{curr}$. After transmitting $\mathcal{T}_{curr}$ to the cloud, the edge device performs a single-step expansion on $\mathcal{T}_{curr}$—accounting for the potential bonus token returned by the cloud—to obtain $\mathcal{T}'_{curr}$. Subsequently, a subset of high-confidence leaf nodes $\mathcal{L}_{top} \subset \mathcal{T}'_{curr}$ is selected as potential starting points for the next inference step. For each selected leaf $l_i \in \mathcal{L}_{top}$, a corresponding candidate tree $\mathcal{T}'_i$ is initialized, where the root of $\mathcal{T}'_i$ logically connects to l_i, representing a hypothetical path continuation contingent upon the acceptance of l_i.

Batched Expansion and Activation. To efficiently construct multiple candidate trees simultaneously on resource-constrained edge devices [21], we leverage the *Batch Processing* capability of the draft model. During the expansion at each depth d, candidate nodes from all candidate trees $\{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\}$ are aggregated into a single inference batch. This batched approach amortizes the overhead associated with kernel launches and memory I/O [22], ensuring that the total latency for concurrently building k candidate trees remains significantly lower than that of sequential execution. Upon receipt of the verification response $\mathcal{R}$ from the cloud, the edge device immediately performs a matching operation. If the accepted path terminates at a leaf node $l^* \in \mathcal{L}_{top}$, the corresponding candidate tree $\mathcal{T}'_{l^*}$ is instantly "activated." This pre-computed tree is immediately serialized and transmitted to the cloud, effectively eliminating the drafting latency for the subsequent round (i.e., $T_{draft} \to 0$). If the accepted path matches no pre-speculated leaf, the candidate trees are discarded and the edge resumes standard drafting from the corrected tokens. By overlapping candidate drafting with cloud verification, the framework improves throughput and keeps edge computation active during communication and verification intervals.

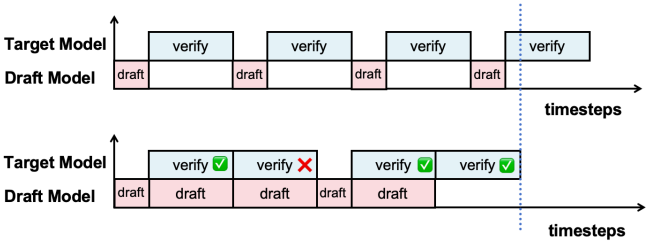


Fig. 4: Timeline comparison between standard sequential inference (top) and our asynchronous pipelining (bottom). In the standard approach, the draft model idles while the target model verifies. Our method leverages this interval to perform pre-drafting, effectively masking the verification latency.

IV. Evaluation

Experimental Setup. We evaluate the proposed framework in a heterogeneous edge-cloud collaborative computing environment. The cloud server is equipped with eight NVIDIA A100 GPUs (40GB) hosting the large-scale target model (M_p), representing a high-performance centralized infrastructure, while the edge node is a consumer-grade workstation with two NVIDIA RTX 4090 GPUs (24GB) running the lightweight draft model (M_q), simulating a typical edge device. The system is implemented in PyTorch [23], and all interactions between the edge and cloud nodes are conducted via *gRPC* to minimize communication overhead and ensure reliable transmission. All experiments are performed under a wide-area network (WAN) setting with a maximum bandwidth of 200 Mbps to emulate real-world deployment conditions. To assess the generalization and effectiveness of the proposed method, we conduct extensive evaluations across multiple model pairs and standard benchmarks.

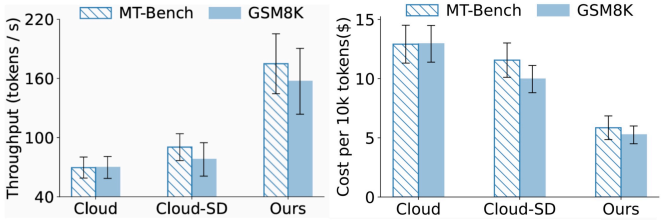
Evaluation Metrics. The key evaluation metric is the *token throughput* of the system, formally defined as:

$$\text{Throughput} = \sum_{i=1}^{\text{Batch Size}} \text{Token-Length}_i / \text{Inference-Latency} \tag{8}$$

It is calculated by dividing the total number of tokens generated (corresponding to Batch Size parallel requests each producing Token Length tokens) by the total latency consumed for completing this batch of inference tasks. Specifically, the batch size is adaptively set to occupy a fixed GPU memory footprint on the cloud side, enabling fair comparison across different settings in edge-cloud collaborative and cloud-only scenarios.

Beyond throughput, we also quantify the serving *costs* based on prevailing market rental prices: $4.05 per hour for an A100 40G GPU and $0.35 per hour for an RTX 4090 GPU [24], [25]. In the ablation study, we verify the effectiveness of our tree-based drafting algorithm using *accept rate*, which is the proportion of the final generated tokens that are produced by the draft model and accepted by the target LLM.

Baselines. To comprehensively evaluate the effectiveness of our proposed framework, we compare it against *two cloud-only* inference methods as well as *four state-of-the-art collaborative*



(a) Throughput Comparison (b) Cost Comparison

Fig. 5: Comparison of cloud-only LLM inference and our method on the dataset of MT-Bench and GSM8K.

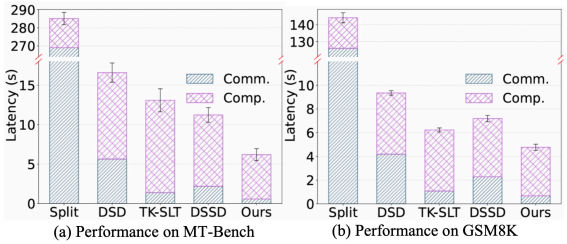
TABLE I: Comparison in terms of cloud memory and latency.

Method	Cloud Model Memory	Latency	Throughput
Cloud	121 GB	16.4 s	69.5 tokens/s
Cloud-SD	146 GB	**5.36 s**	90.3 tokens/s
Ours	**121 GB**	6.20 s	**175 tokens/s**

methods. The two representitive cloud-only methods are as follows. (1) *Cloud-Only Autoregressive Generation*, where the target model executes entirely on the server, serving as the standard baseline for inference latency. (2) *Cloud-Only Specultive Decoding (Cloud-SD)* [5], where both the draft model and target LLMs are deployed on the cloud to perform speculative decoding. And we configure the batch size for Cloud-SD to 5 to ensure stable execution.

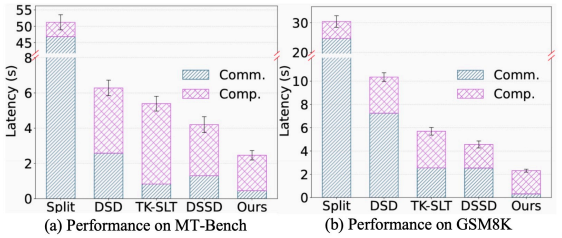
Then we choose four state-of-the-art collaborative methods. (1) *Model Split [26]*, a traditional offloading strategy that partitions the model layers between the edge device and the cloud. (2) *Distributed Speculative Decoding (DSD)* [13], which applies standard speculative decoding in a distributed setting, necessitating the transmission of full probability distributions. (3) *DSSD* [9], which optimizes communication latency of the collaborative speculative decoding via splitting the verification phases. (4) *TKSLT* [10], which mitigates communication bottlenecks in collaborative speculative decoding by transmitting only the top-k sparse logits.

Models and Datasets. To demonstrate the scalability of our framework across different parameter scales, we conduct evaluations using the Vicuna family of models [27], which are fine-tuned from LLaMA models [2]. We establish two distinct draft-target pairs: Vicuna-33B paired with Vicuna-7B, and Vicuna-7B paired with Vicuna-68M. In terms of hardware allocation, the 33B model is distributed across 8 GPUs to accommodate its massive memory footprint, whereas the 7B model and 68M model are deployed on a single GPU. We assess these models on two widely used benchmarks to cover diverse capabilities, specifically utilizing MT-Bench [28] for multi-turn conversation and instruction following, and GSM8K [29] for mathematical reasoning tasks. For GSM8K, we use only the first 100 test examples for evaluation. Unless specifically noted, a uniform temperature setting of 1.0 is applied across all experimental configurations, and the maximum generation length is set to 128 tokens.



(a) Performance on MT-Bench (b) Performance on GSM8K

Fig. 6: Communication and computation latency of different collaborative methods on two datasets using 33B-7B models.



(a) Performance on MT-Bench (b) Performance on GSM8K

Fig. 7: Latency performance on 7B-68M models.

A. End-to-End Performance and Cost-Efficiency

We first evaluate the end-to-end performance of our proposed framework against cloud-only baselines to demonstrate the core advantages freeing up cloud memory to maximize throughput. Results are shown in Fig. 5 and Table I.

As detailed in Table I, co-deploying both the draft 7B model and target 33B model in the cloud (Cloud-SD) consumes 146 GB of cloud memory. By offloading the draft model to the edge device, our method reduces the cloud memory footprint back to 121 GB, matching the memory usage of standard autoregressive generation (Cloud). Crucially, this saved memory allows the cloud server to accommodate a larger batch size, which translates into a massive throughput advantage: *our approach achieves a higher throughput of 175 tokens/s, which is 2.52× of cloud-only autoregression and 1.94× of Cloud-SD*. It verifies that, despite the inherent cloud-edge communication latency that makes our method slightly slower compared to Cloud-SD (6.20 s vs. 5.36 s), the significant memory savings translate to a remarkable throughput benefit.

Fig. 5 further illustrates this advantage across different datasets. Compared to Cloud-SD, our edge-cloud collaborative method *consistently achieves about 2× throughput gains* on both MT-Bench and GSM8K datasets. Meanwhile, since the draft model computations are offloaded to cost-effective edge devices, *our method separately reduces the token generation cost by about 47.3% and 31.0% on each dataset*. This indicates that our method is promising as an economical solution for fast and large-scale LLM serving.

B. Latency Breakdown and Collaborative Efficiency

To thoroughly compare with existing edge-cloud collaborative methods, we break down the total inference time into communication and computation latency. Comparison results

TABLE II: **Ablation Study.**

Method	Throughput (token/s)	Cost_{100k} ($)
w/o Tree	143.70	6.51
w/o Parallel	148.56	6.30
Full	**174.94**	**5.35**

against Split, DSD, TK-SLT, and DSSD are shown in Fig. 6 and Fig. 7, using 33B-7B models and 7B-68M models.

Communication Latency. Due to the large intermediate features of LLMs, the Split approach incurs a transmission volume that is dozens of times higher than that of other methods, which severely impacts the corresponding latency. The other collaborative speculative decoding methods all suffer from the substantial communication overhead of logits transmission. On the contrary, *our token-only communication protocol consistently reduces the communication latency by over 38.4% across diverse datasets using LLM models of varying sizes.* As a result, our method visibly outperforms all state-of-the-art methods in overall latency.

Computation Latency. Beyond communication savings, our method also demonstrates superior computation latency. On the MT-Bench dataset, where the task is more challenging with higher computational complexity, this superiority becomes more pronounced. Specifically, *compared to the state-of-the-art methods, our computation latency of 33B-7B model and 7B-68M model are reduced by 37.9% and 30.5%, respectively.* This is attributed to two key designs. First, our budget-aware tree drafting strategy significantly improves the acceptance rate of candidate tokens, thereby reducing the total number of required interaction rounds between the edge and the cloud. Second, our asynchronous pipelining mechanism effectively overlaps the edge draft model's computation time with the cloud's verification time, which masking this computational overhead by minimizing the idle bubbles.

Therefore, our method consistently achieves the lowest overall latency across diverse task complexities (MT-Bench and GSM8K) and varying model scales (33B-7B and 7B-68M), and *reduces the overall inference latency by up to 49.3%*. This verifies its strong generalization capabilities in real-world edge-cloud environments.

C. Ablation Study

To comprehensively understand the contributions of each components and the performance under various hyperparameters, we conduct detailed ablation studies and evaluations.

Effectiveness of Core Components. We first ablate the two fundamental components of our framework: the tree-based draft structure and the asynchronous pipelining mechanism. As shown in Table II, the full Ygg framework achieves the highest throughput of 174.94 tokens/s and the lowest inference cost. The configuration without tree structures (*w/o Tree*) exhibits a severe performance drop, achieving a throughput of only 143.70 tokens/s. This substantial degradation indicates that, without a tree-based structure, the system degenerates into a fragile single-sequence drafting paradigm. Such a linear approach fails to exploit the cloud's parallel verification capability, resulting in lower token acceptance rates and increased communication rounds. Furthermore, the configuration without parallel execution (*w/o Parallel*) drops to 148.56 tokens/s. This performance gap reveals the presence of an idle computational "bubble" during the cloud verification phase under standard sequential execution. By leveraging our algorithm to pre-draft candidate trees, the full framework effectively overlaps drafting with verification, masking the latency and maximizing throughput.

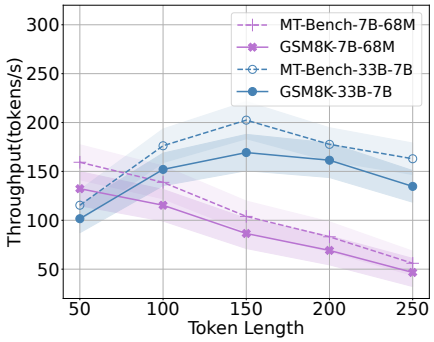
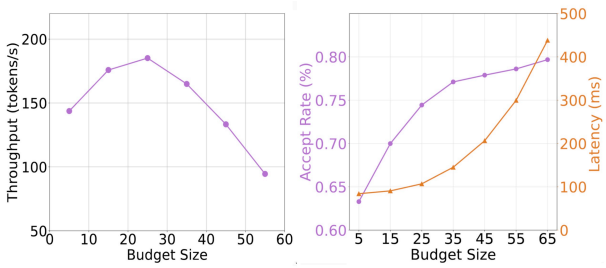


Fig. 8: Performance with different sequence length.



(a) Throughput Performance (b) Accept Rate and Latency

Fig. 9: Impact of budget size choice. As the budget increases, the acceptance rate rises, but the transmission latency also increases, so the throughput first increases and then decreases.

Impact of Sequence Length. As shown in Eq. 8, the throughput is affected by the length of generated tokens. We separately set the maximum token length to different values and plot the throughput in Fig. 8. It's observed that for large LLM models, *i.e.*, 33B-7B, the throughput performance remains stable above 150 tokens/s across different sequence lengths. While for 7B-68M model pair, we observe a continuous downward trend in throughput as the sequence length increases. The kernel reason is that the release of the 68M draft model saves relatively little GPU memory, which hardly achieves sufficient batch size gains for long sequences. *This implies that our approach yields more stable acceleration for larger LLM models, which releases substantial GPU memory.* As models continue to grow in complexity, it is expected to play a more important role in the future.

Impact of Budget Size. Fig. 9 illustrates the trade-off introduced by the tree budget size. As the budget increases, the edge device is permitted to transmit more candidate tokens. This improves the acceptance rate as in Fig. 9 (b). However, an

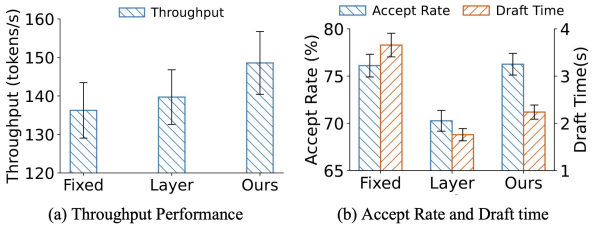


Fig. 10: Impact of tree construction algorithm.

expanded budget simultaneously increases the computational and transmission latency of a single interactive round (drafting, verification, and communication). Consequently, as shown in Fig. 9 (a), *the throughput exhibits a concave trend: it initially increases as the benefits of a higher acceptance rate dominate, but eventually decreases when the per-round latency overhead outweighs the reduction in interaction rounds*.

Impact of Tree Construction Algorithm. Fig. 10 compares our Budget-Aware Global Adaptive Tree Construction against standard Fixed and Layer-wise expansion algorithms. The *Fixed* strategy aggressively expands a large, regular tree before pruning; while this exhaustive approach yields the highest token acceptance rate, its throughput is bottlenecked by the massive redundant draft time overhead on the edge device. Conversely, the *Layer-wise* approach yields a significantly lower acceptance rate because it restricts node selection strictly within current layers, frequently discarding globally optimal nodes located at shallower depths. Our proposed global adaptive algorithm effectively resolves this dilemma. By dynamically selecting high-confidence nodes from a global buffer, it maintains a highly competitive acceptance rate with acceptable level of computational redundancy. As a result, it achieves a better trade-off and delivers the highest overall throughput among the three strategies.

V. Related Work

Efficient On-Device Inference. To mitigate the heavy deployment overhead of LLMs on centralized clouds, recent research has pivoted towards enabling local inference. However, existing workarounds fail to address this issue without significant compromises. Specifically, weight quantization methods [30], [31] successfully compress the memory footprint but typically incur an irreversible degradation in generation quality. Alternatively, substituting them with edge-native lightweight models facilitates local execution [32], [33], yet their reasoning capabilities fall drastically short of their large-scale counterparts. Other approaches, such as storage offloading [34], attempt to circumvent GPU memory limits via system RAM or SSD. However, they suffer from severe I/O bandwidth bottlenecks, often resulting in higher latencies than pure cloud solutions. Moreover, distributed edge inference [11], [35], which partitions models across multiple devices, is largely undermined by the prohibitive synchronization overheads over bandwidth-limited networks. Ultimately, relying strictly on resource-constrained local environments is inadequate for high-performance LLM inference.

Speculative Decoding. Speculative Decoding has emerged as a promising paradigm to accelerate autoregressive generation while maintaining lossless consistency with target LLMs [4]–[6]. The fundamental speculative decoding process employs a lightweight draft model to rapidly generate candidate sequences for parallel verification by the target model. To improve drafting efficiency and acceptance rates, recent advances have diversified drafting mechanisms, exploring self-speculative decoding [36], retrieval-based methods [37], and draft-free generation architectures [18]. Furthermore, state-of-the-art approaches widely adopt tree-based decoding [16], [20], [38], [39] to explore multiple generation paths simultaneously, maximizing the yield of each verification pass. However, these advanced speculative decoding techniques are inherently designed for cloud-only infrastructures equipped with shared memory and high-bandwidth interconnects, rendering them inapplicable to edge-cloud collaborative scenarios.

Edge-Cloud Collaborative Inference. To alleviate the burden of cloud servers in LLM inference, edge-cloud collaboration has been actively studied. Early collaborative paradigms predominantly relied on split computing [40]. While effective for Convolutional Neural Networks (CNNs), it incurs unacceptable communication overheads for LLMs due to the continuous transmission of massive intermediate activation tensors. To circumvent tensor transmission entirely, collaborative speculative decoding extends the aforementioned speculative decoding paradigm by offloading the draft model to the edge while retaining the target model on the cloud [13], [14]. Nevertheless, it introduces a new communication bottleneck that necessitates the continuous transmission of drafted tokens alongside their high-dimensional probability distributions (logits) over constrained networks [9]. Even with optimized protocols like Top-K Sparse Logits [10], transmitting these heavy payloads remains prohibitive. In stark contrast, our framework pioneers a token-only communication protocol to fully eradicating this limitation. In stark contrast, our framework pioneers a token-only communication protocol to overcome this bottleneck and achieve full acceleration.

VI. Conclusion

In this paper, we proposed Ygg, an efficient edge-cloud collaborative speculative decoding framework with minor transmission overhead to improve LLM inference QoS. By offloading the lightweight draft model to cost-effective edge devices and employing a token-only transmission mechanism, Ygg successfully resolves both the severe GPU memory constraints of cloud-only setups and the heavy communication bottlenecks found in existing collaborative methods. In the framework, we introduced two key mechanisms: a budget-aware global adaptive tree-based draft generation scheme that ensures high token acceptance rates with minimal overhead, and an asynchronous pipelining strategy that overlaps edge drafting with cloud verification to eliminate idle computational bubbles. Extensive experiments verify that it can effectively improve throughput and reduce cost of LLM inference.

REFERENCES

[1] DeepSeek-AI, "DeepSeek-V3 technical report," DeepSeek-AI, Tech. Rep., 2024, arXiv preprint arXiv:2412.19437.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] R. Pope, S. Douglas, A. Chowdhery *et al.*, "Efficiently scaling transformer inference," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 5, 2023.

[4] H. Xia, T. Ge, P. Wang, S.-Q. Chen, F. Wei, and Z. Sui, "Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation," in *Findings of the Association for Computational Linguistics: EMNLP*, 2023, pp. 3909–3925.

[5] C. Chen, S. Borgeaud, G. Irving *et al.*, "Accelerating large language model decoding with speculative sampling," *arXiv preprint arXiv:2302.01318*, 2023.

[6] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the International Conference on Machine Learning (ICML 2023)*. PMLR, 2023, pp. 19 274–19 286.

[7] P. SK, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the performance of accelerated jetson edge devices for training deep learning models," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 3, pp. 1–26, 2022.

[8] P. Nikolaou, Y. Sazeides, A. Lampropulos, D. Guilhot, A. Bartoli, G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, K. Tovletoglou, L. Mukhanov *et al.*, "On the evaluation of the total-cost-of-ownership trade-offs in edge vs cloud deployments: A wireless-denial-of-service case study," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 2, pp. 334–345, 2019.

[9] J. Ning, C. Zheng, and T. Yang, "Dssd: Efficient edge-device llm deployment and collaborative inference via distributed split speculative decoding," *arXiv preprint arXiv:2507.12000*, 2025.

[10] C. Zheng and T. Yang, "Communication-efficient collaborative llm inference via distributed speculative decoding," *arXiv preprint arXiv:2509.04576*, 2025.

[11] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[12] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International conference on machine learning*. PMLR, 2022, pp. 18 332–18 346.

[13] W. Zhao, W. Jing, Z. Lu *et al.*, "Edge and terminal cooperation enabled llm deployment optimization in wireless network," in *Proceedings of the IEEE/CIC International Conference on Communications in China Workshops (ICCC 2024)*. IEEE, 2024, pp. 220–225.

[14] Z. Hao, H. Jiang, S. Jiang *et al.*, "Hybrid slm and llm for edge-cloud collaborative inference," in *Proceedings of the Workshop on Edge and Mobile Foundation Models*, 2024, pp. 36–41.

[15] X. Zhang, N. Yan, Y. Su, Y. Deng, and T. Mahmoodi, "Communication-aware knowledge distillation for federated llm fine-tuning over wireless networks," *arXiv preprint arXiv:2509.01750*, 2025.

[16] X. Miao, G. Oliaro, Z. Zhang *et al.*, "Specinfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ACM, 2024, pp. 932–949.

[17] Y. Li, F. Wei, C. Zhang, and H. Zhang, "Eagle: Speculative sampling requires rethinking feature uncertainty," *arXiv preprint arXiv:2401.15077*, 2024.

[18] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple llm inference acceleration framework with multiple decoding heads," *arXiv preprint arXiv:2401.10774*, 2024.

[19] R. Svirschevski, A. May, Z. Chen *et al.*, "Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices," in *Advances in Neural Information Processing Systems*, vol. 37, 2024, pp. 16 342–16 368.

[20] J. Wang, Y. Su, J. Li *et al.*, "Opt-tree: Speculative decoding with adaptive draft tree structure," *Transactions of the Association for Computational Linguistics*, vol. 13, pp. 188–199, 2025.

[21] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, and J. Chen, "A review on edge large language models: Design, execution, and applications," *ACM Computing Surveys*, vol. 57, no. 8, pp. 1–35, 2025.

[22] 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 *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[24] Google Cloud, "Google cloud pricing," https://cloud.google.com/, 2026, accessed: 2026.

[25] Vast.ai, "Vast.ai global gpu market pricing," https://vast.ai/, 2026, accessed: 2026.

[26] L. Jiang, S. D. Fu, Y. Zhu *et al.*, "Janus: Collaborative vision transformer under dynamic network environment," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 2025)*. IEEE, 2025, pp. 1–10.

[27] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez *et al.*, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," *See https://vicuna. lmsys. org (accessed 14 April 2023)*, vol. 2, no. 3, p. 6, 2023.

[28] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in neural information processing systems*, vol. 36, pp. 46 595–46 623, 2023.

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

[30] E. Frantar, S. Ashkboos, T. Hoefler *et al.*, "OPTQ: Accurate post-training quantization for generative pre-trained transformers," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[31] J. Lin, J. Tang, H. Tang *et al.*, "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 6, 2024.

[32] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi *et al.*, "Mobilellm: Optimizing sub-billion parameter language models for on-device use cases," in *Forty-first International Conference on Machine Learning*, 2024.

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

[34] Y. Sheng, L. Zheng, B. Yuan *et al.*, "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023, pp. 31 094–31 116.

[35] C. Hu and B. Li, "When the edge meets transformers: Distributed inference with transformer models," in *Proceedings of the 44th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 82–92.

[36] J. Zhang, J. Wang, H. Li, L. Shou, K. Chen, G. Chen, and S. Mehrotra, "Draft& verify: Lossless large language model acceleration via self-speculative decoding," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 11 263–11 282.

[37] Z. He, Z. Zhong, T. Cai, J. Lee, and D. He, "Rest: Retrieval-based speculative decoding," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1)*, 2024, pp. 1582–1595.

[38] Z. Chen, A. May, R. Svirschevski, Y. Huang, M. Ryabinin, Z. Jia, and B. Chen, "Sequoia: Scalable, robust, and hardware-aware speculative decoding," *arXiv preprint arXiv:2402.12374*, 2024.

[39] F. Lin, H. Yi, Y. Yang, H. Li, X. Yu, G. Lu, and R. Xiao, "Bita: Bi-directional tuning for lossless acceleration in large language models," *Expert Systems with Applications*, vol. 279, p. 127305, 2025.

[40] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 615–629.