DIAA: A Decoding-Efficient Inference Acceleration Approach for On-Device Large Language Models

Hao Tian[1], Sheng Lu[1], Fuwen Tian[1], Guangming Cui[2], Zheng Li[1], Xuyun Zhang[3], Quan Z. Sheng[3], Wanchun Dou[1]*

[1]State Key Laboratory for Novel Software Technology, School of Computer Science, Nanjing University, China
[2]School of Software, Nanjing University of Information Science and Technology, China
[3]School of Computing, Macquarie University, Australia
{htian,sheng-lu,fuwentian}@smail.nju.edu.cn, gcui@nuist.edu.cn, zheng_li@smail.nju.edu.cn,
{xuyun.zhang,michael.sheng}@mq.edu.au, douwc@nju.edu.cn

Abstract

Large Language Models (LLMs) have revolutionized intelligent interactions, enabling mobile applications such as personal assistants on edge devices for local execution. Speculative decoding (SD) has emerged as a promising paradigm to accelerate LLM inference without compromising generation quality, employing a draft-then-verify manner. However, due to the constrained computing and memory resources on edge devices, existing SD works heavily rely on an auxiliary draft model that incurs additional memory burden and hinders the adaptability, as well as static token trees that yield suboptimal inference performance. To this end, we propose DIAA, a Decoding-efficient Inference Acceleration Approach for on-device LLMs. DIAA achieves plug-and-play and model-agnostic inference speedup with memory and computation efficiency for edge devices. Specifically, a pair of lightweight look-up tables (LUTs) is constructed by Top-K token sampling to cache historical tokens and probabilities for rapid candidate drafting. DIAA integrates a dynamic token tree with prior LUTs enabling paralleled verification, updated during decoding process, to adapt the online context. A computation overlap is then employed to pipeline the update operations of token tree, LUTs, and KV cache to improve the computational efficiency. Finally, through extensive experiments implemented on edge platform NVIDIA Jetson, DIAA outperforms existing baselines in generation speed and inference wall-clock time, while incurring minimal memory overhead.

Introduction

The advent of Large Language Models (LLMs) represents a paradigm shift in Natural Language Processing (NLP) (Vaswani et al. 2017). LLMs like Llama (Touvron et al. 2023) and DeepSeek (Liu et al. 2024a) have demonstrated remarkable capabilities across a spectrum of NLP tasks, ranging from sophisticated summarization to complex reasoning. It has fueled a paradigm shift to move LLMs from the cloud to the edge (Zheng et al. 2025). The prospect of running LLMs directly on edge devices, such as IoT gadgets, is particularly compelling. On-device execution offers significant advantages, including reduced latency, enhanced privacy and security by keeping data local, and lower operational costs by eliminating the need for constant communication with remote servers. The success of LLMs has propelled the widespread deployment of on-device applications, fostering innovation in AI assistants (Lee et al. 2024), smart home agents (King et al. 2024), and autonomous systems (Huang et al. 2025). However, unprecedented success comes with substantial computational and memory demands, which severely constrain the deployment on resource-limited edge devices under stringent service-level objectives (SLOs) (Xu et al. 2025; Ye et al. 2025).

As illustrated in Fig. 1(a), the inference phase of LLMs, especially in the autoregressive decoding (AR) paradigm, presents a major computational bottleneck. During generation, each token is produced sequentially, with every step involving the full forward pass (Kwon et al. 2023). The AR mode leads to significant latency, which are particularly prohibitive for mobile applications to improve the user experience where real-time responsiveness are critical. While existing efforts have explored model pruning (Ma, Fang, and Wang 2023; Gao et al. 2024; Zhang et al. 2025b), quantization (Lin et al. 2024; Zeng et al. 2025; Hu, He, and Wu 2025), and distillation (Wang et al. 2025; Hu et al. 2025a), these methods often incur a trade-off between computational efficiency and model performance, limiting practical applicability in maintaining high-quality generation. Other works (Zhang et al. 2025a; Ye et al. 2025) focus on offloading the computations of LLMs from edge to cloud or edge peers, comprising the unpredictable communications overhead.

A promising alternative paradigm that directly tackles the sequential dependency of AR to speedup the inference is speculative decoding (SD) (Leviathan, Kalman, and Matias 2023; Chen et al. 2023). Shown in Fig. 1(b), SD employs a smaller and faster draft model to generate a sequence of candidate tokens speculatively in a draft-then-verify manner. The candidates are then efficiently verified in parallel by the original target LLM. If the candidates are accepted, multiple tokens are generated per single forward pass of the target model, leading to significant latency reduction. Crucially, SD guarantees that the output distribution remains identical to the target model, offering lossless acceleration (Xia et al. 2024). Unlike conventional compression methods, SD accelerates the decoding phase without requiring

*Corresponding author.

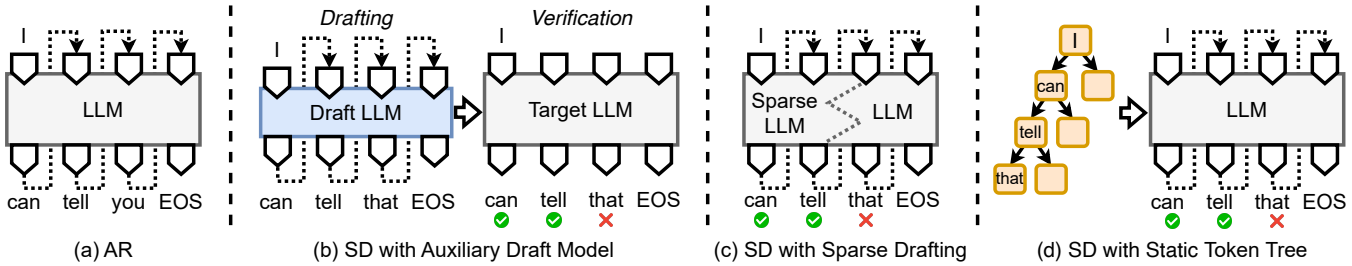

Figure 1: An example of existing decoding approaches.

model modification, making it particularly appealing for deployment in resource-constrained environments.

Despite the promise of SD for speeding up inference, existing approaches still face significant limitations, particularly in the context of on-device deployment: (1) The model-based approaches (Li et al. 2024b,a; Cai et al. 2024) need a fine-tuned lightweight draft model that aligns the output of original. However, fine-tuning or re-training a draft model for widely-used LLMs obstacles the fast adaptability for mobile scenarios, especially for domain-specific on-device applications. (2) Similarly, model-based approaches, such as EdgeLLM (Xu et al. 2025), rely on an auxiliary draft model, which introduces additional memory overhead and maintenance complexity for edge devices with limited memory and compute capacity. (3) Existing tree-based works (Luo et al. 2025; Miao et al. 2024; Chen et al. 2024) employ static verification trees that lack adaptability by static structures, which cannot dynamically adjust to the changing context of the generated text, leading to suboptimal draft acceptance rates and limited speedup. Consequently, we lie out the key research question that: *How to maximize inference speedup for on-device LLMs without requiring an auxiliary model while dynamically adapting to the generation context?*

To address the above issue, we propose DIAA, a Decoding-efficient Inference Acceleration Approach for on-device LLMs by SD. DIAA is designed to deliver plug-and-play, model-agnostic decoding acceleration with minimal memory overhead, tailored for resource-constrained edge devices. Specifically, DIAA introduces a pair of lightweight look-up tables (LUTs), a token LUT (t-LUT) and a probability LUT (p-LUT), which are initialized via Top-K sampling to cache historical token feedback from the LLM. The LUTs enable efficient sampling of candidate draft tokens with negligible memory overhead. Subsequently, a dynamic token tree is constructed based on prior LUTs, facilitating parallel token verification, which dynamically adapts to both prediction confidence and the evolving input context, thereby balancing computational cost and decoding speed. Moreover, CUDA stream-level parallelism is employed to concurrently update the dynamic token tree, LUTs, and KV cache, significantly improving pipeline efficiency. We evaluate DIAA on realistic edge platform *NVIDIA Jetson Orin Nano Super*. Through extensive experiments on various models and tasks, DIAA gains superior performance on goodput and latency compared to state-of-the-art baselines. Our main contributions are summarized as

- We conduct pilot experiments of existing SD works on edge devices to explore the potential for model-agnostic inference acceleration via dynamic token tree drafting. Motivated by this, we propose DIAA, enabling plug-and-play SD on edge devices to achieve lossless speedup.
- We introduce an LUT-enhanced candidate token drafting with lightweight memory overhead, to cache historical token feedback generated from LLMs. Moreover, design a dynamic token tree to adapt the speculative search tree via real-time context. The CUDA steam-level overlap is then developed to improve computing efficiency.
- We implement DIAA on real-world edge platform NVIDIA Jetson and extensive experiments through various LLMs and tasks demonstrate the superiority and effectiveness of our approach.

Motivating Example

Fig. 1 presents an example comparing typical decoding approaches. As shown in Fig. 1(a), AR generates tokens one by one, requiring a full model pass at each step. On resource-constrained edge devices, this leads to high latency, making real-time applications impractical. To accelerate inference, SD employs a lightweight draft model to predict multiple tokens before verification by the target model, as illustrated in Fig. 1(b). However, using an auxiliary draft model increases memory usage and increasing deployment complexity for edge devices. Fig. 1(c) shows sparse self-drafting, where selected layers of the target model are reused for drafting. Although this reduces overhead, it relies on larger models with redundant layers (e.g., LLaMA-2-70B) (Zhang et al. 2024), limiting its use on edge platforms. Finally, static token tree sampling, shown in Fig. 1(d), builds a speculative tree of token sequences, enabling parallel verification via tree attention. While effective at reducing redundant drafting, static trees lack adaptability to dynamic input, limiting the performance in real-time scenarios.

In this work, we aim to accelerate the inference, especially the decoding latency, for resource-limited edge devices. However, the existing decoding methods are still facing challenges in computing and memory efficiency under the resource constraints. The motivations are revealed below by pilot experiments, conducted on a representative edge platform *NVIDIA Jetson Orin Nano Super* (8GB).

Motivation #1. The SD using an auxiliary model incurs additional GPU memory overhead, which significantly

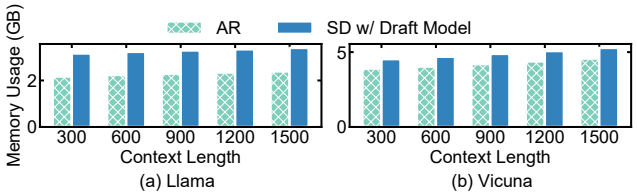

Figure 2: The GPU memory usage under different context lengths on NVIDIA Jetson. Draft model: Lama-3.2-1B-Instruct-INT4 (left), Vicuna-160m (right).

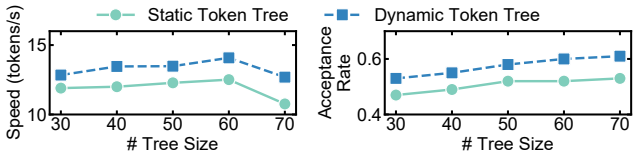

Figure 3: The generation speed and acceptance rate between static token tree and dynamic token tree on NVIDIA Jetson.

burdens resource-limited edge devices and deployment complexity. As illustrated in Fig. 2, we evaluate the memory footprint across varying context lengths to compare vanilla AR and SD with an auxiliary draft model. For example, Fig. 2(a) shows that SD with a draft model requires an additional 1 GB memory over AR for the Llama-3.2-1B-Instruct-INT4, due to the necessity of loading and running both the draft and target models simultaneously. While a 1 GB increase may appear modest on devices with 8 GB or more memory, it is important to consider that many ubiquitous edge devices under unified memory operate with substantially less memory and must allocate significant portions for OS processes and other applications. In the constrained environments, even such additional overhead can cause resource contention and limit the feasibility of running LLM inference locally. Furthermore, many widely-used LLMs lack smaller variants that align with the original model with sufficient accuracy to yield meaningful latency improvements. However, such tight coupling also limits the draft model's adaptability, challenging to quickly fine-tune or reconfigure for domain-specific applications on edge devices. Therefore, the model-agnostic method is desired to achieve inference acceleration while maintaining the resource efficiency.

Motivation #2. The static token tree-based SD achieves sub-optimal inference performance, limiting its applicability for low-latency, on-device generation. Unlike the SD with sparse self-drafting that relies on the layer redundancy of larger models, token tree sampling enables a lightweight and model-agnostic acceleration by drafting multiple branches in parallel. However, static token trees, most existing work adopted (Miao et al. 2024; Luo et al. 2025; Chen et al. 2024), are constructed once using fixed Top-K sampling and remain unchanged. To validate these drawbacks, we compare the generation speed and token acceptance rate between static and dynamic token tree (details of dynamic token tree are provided as follows) under varying tree sizes, as shown in Fig. 3. Dynamic trees achieve faster

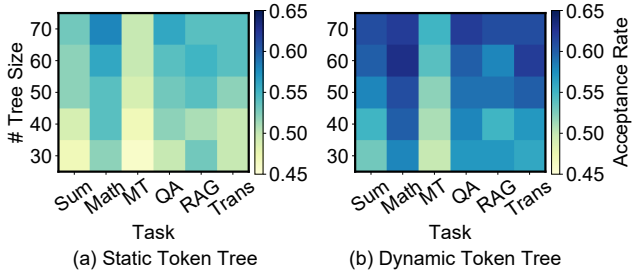

Figure 4: The acceptance rate heatmap across six tasks under different scale of tree sizes on NVIDIA Jetson.

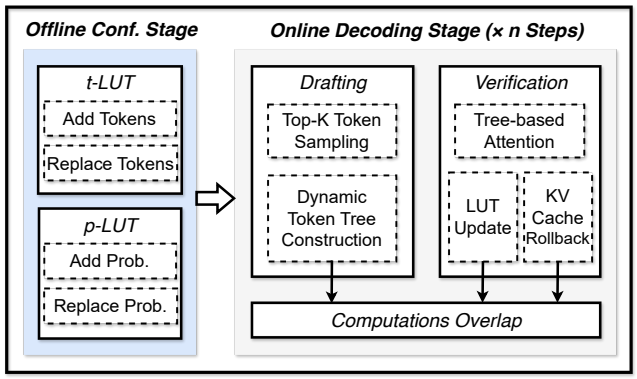

Figure 5: The framework of DIAA.

generation and higher acceptance, which adaptively restructures the speculative paths based on intermediate feedback and adjust depth and branching accordingly. Furthermore, we assess the acceptance rate across six representative tasks from Spec-Bench (Xia et al. 2024). In Fig. 4, the dynamic token tree maintains a higher acceptance rate across all tasks. Static trees speculatively expand unpromising paths without runtime update, exacerbating latency and reducing throughput. Moreover, static structures are poorly suited to adapt across dynamic input distributions or task shifts, which are common in interactive mobile scenarios.

A Decoding-Efficient Inference Acceleration Approach for On-Device LLMs

Approach Overview

Fig. 5 presents the framework of DIAA, which consists of two stages: **1) Offline Configuration**, which initializes two precomputed LUTs, t-LUT caches Top-K token candidates and p-LUT caches the probabilities. These LUTs guide token selection and tree construction during decoding. **2) Online Decoding**, iteratively updates the draft tree over N steps until the EOS token is generated. In each step, we construct the dynamic token tree, balancing search depth and efficiency. The draft is then verified against the target model using tree-based attention. To maximize the resource utilization, the computations overlap is introduced to parallel the update operations of dynamic tree, LUTs, and KV cache.

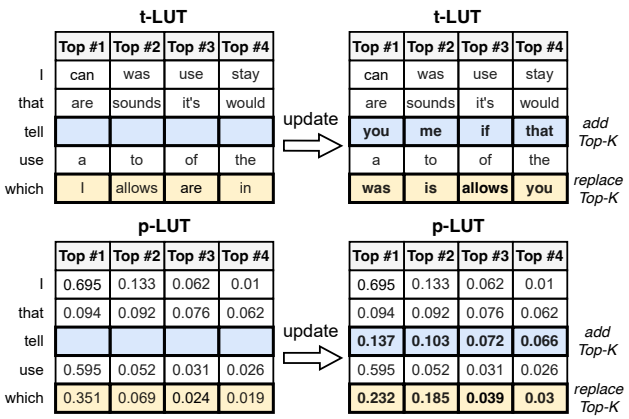

Figure 6: An illustrative example of updating t-LUT and p-LUT with add and replacement operations.

Look-Up Table-Enhanced Token Initialization

To enable fast and resource-efficient token proposal during inference, an LUT-enhanced token initialization is proposed, including the t-LUT and the p-LUT. These two LUTs provide Top-K token candidates and their estimated probabilities based on past model behavior, significantly reducing the computational burden associated with online candidate drafting. With LUTs, the Top-K tokens and probabilities are cached and reused for future decoding phase, with fast sampling of drafted tokens that some common tokens follow the similar patterns. **1) t-LUT** stores a mapping from partial input contexts to the corresponding Top-K most likely next tokens, as observed during offline profiling. Each entry in the t-LUT is indexed by a context representation (which can be a fixed-length token prefix or hashed embedding), and maps to a list of token IDs ordered by descending likelihood. This structure serves as a lightweight prior that facilitates efficient Top-K candidate proposal during the online decoding phase. **2) p-LUT** complements the t-LUT by storing the associated probabilities for each token in the Top-K list. These probabilities are obtained either through offline empirical statistics or through approximations of the target model's softmax output. During runtime, the p-LUT provides a fast estimation of the expected confidence for each speculative token, enabling budget-aware tree expansion and early pruning of low-confidence paths. To mitigate the cold start of LUTs, the t-LUT and p-LUT are initialized with a warm-up dataset to collect and cache the candidates in an offline manner.

As shown in Fig. 6, both LUTs are continuously updated during inference through two operations: 1) *add operation*. When a verified context-token pair is not present in the LUTs, a new entry is created, adding the token to the Top-K list and recording its probability. 2) *replacement operation*. When the Top-K list is full, a newly verified token replaces the least informative entry, with the p-LUT updated accordingly. Consequently, the t-LUT and p-LUT evolve adaptively to reflect the most recent and confident Top-K candidates, thereby enhancing the likelihood of proposing accurate tokens in future decoding steps and reducing unnecessary verification overhead.

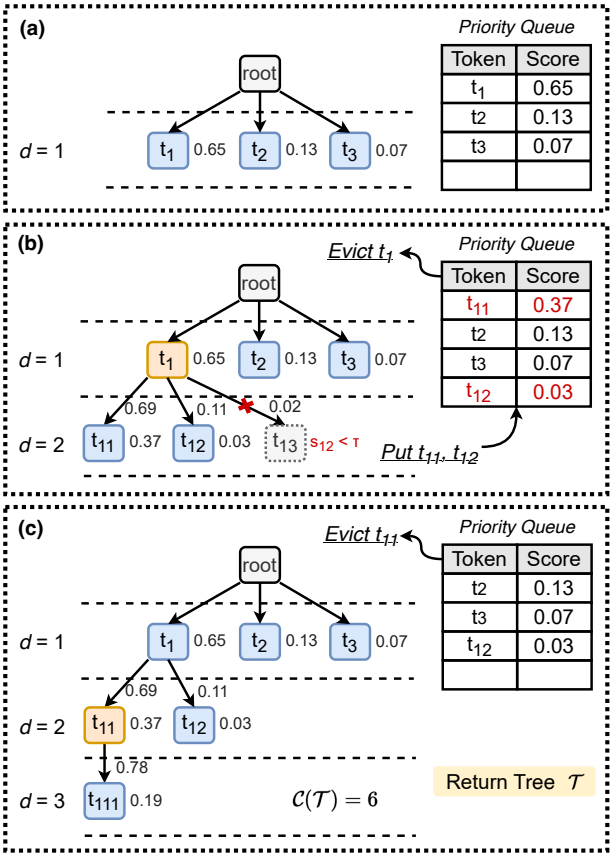

Figure 7: An illustrative example of dynamic tree construction, where B=6, δ=0.8, α=0.7, and τ=5e-3.

Dynamic Tree Construction and Drafting

According to prior work (Li et al. 2024a), the confidence score of drafted candidates shows a strong positive correlation with the token's acceptance rate. Let $x_{1:t} = [x_1, x_2, ..., x_t]$ denote the current input context, and let $p_{draft}(x_{t+1}|x_{1:t})$ be the probability distribution over next tokens given by the drafting. A draft tree $\mathcal{T}$ is a rooted, directed tree structure where 1) each node $v \in V(\mathcal{T})$ corresponds to a speculative token x_v; 2) each path from the root to a leaf represents a speculative sequence extension $x_{t+1:t+l}$ with length l; 3) and the root node corresponds to the current context $x_{1:t}$, and each edge encodes a transition.

The goal of dynamic tree construction is to maximize the cumulative probability of speculative paths under a predefined token budget. We define the utility of tree $\mathcal{T}$ as

$$\mathcal{U}(\mathcal{T}) = \sum_{\pi \in \mathcal{P}(\mathcal{T})} \prod_{i=1}^{|\pi|} p_{draft}(x_{\pi_i}|x_{\pi<i}), \qquad (1)$$

where $\mathcal{P}(\mathcal{T})$ is the set of all root-to-leaf paths, and $x_{\pi<i}$ denotes the prefix of token x_{π_i}. The cost of the tree is the total number of tokens drafted as $\mathcal{C}(\mathcal{T}) = \sum_{\pi \in \mathcal{P}(\mathcal{T})} |\pi|$.

A budget constraint B is imposed to ensure $\mathcal{C}(\mathcal{T}) \leq B$. To maximize the utility of tree $\mathcal{T}$ defined in Eq. (1), DIAA employs a dynamic tree construction algorithm. For example, as shown in Fig.7, DIAA iteratively expands a tree $\mathcal{T}$ to

maximize cumulative token probabilities within the budget constraint. A priority queue is maintained to greedily select high-confidence paths based on LUT-enhanced Top-K token sampling. The first-depth nodes are expanded by default. To balance the breadth (token diversity) and depth (sequence length), while keeping the tree probability-guided, the depth decay and width decay are introduced. Therefore, paths are scored based on cumulative probability, adjusted by depth and width decay factors, and pruned if scores fall below a pre-defined threshold τ. Path π is scored as

$$s(\pi) = \prod_{i=1}^{|\pi|} p_{draft}(x_{\pi_i}|x_{\pi<i}) \cdot \delta^{|\pi|-1} \cdot \alpha^{topk(\pi)-1}, \quad (2)$$

where δ and α denote the depth and width decay factor. These decay factors ensure the tree grows toward deeper but high-confidence branches. For example, according to the score of paths, the greedy expansion and threshold pruning are shown in Fig. 7(b). First, the highest-priority node t_1 is popped from the queue for expansion and its successors t_{11}, t_{12}, and t_{13}. Since the score of $t_{13} < \tau$, this path is pruned and discarded. The remaining paths are inserted into the tree and priority queue. Once the predefined tree budget is fully consumed with $\mathcal{C}(\mathcal{T}) = 6$, tree construction halts. Consider the computational cost of the tree construction, the intervals of updating dynamic tree is pre-defined within fixed steps during the decoding phase.

Verification by Dynamic Token Tree

Building upon the LUT-augmented token guidance and the constructed dynamic token tree, verification efficiency is further enhanced. As a result, the path with the maximum scores are selected as verified path during the verification phase, to intelligently focus on critical paths while avoiding exhaustive examination of all possible branches.

In practice, although multiple token paths are generated in parallel, only one is ultimately verified as correct. Without reusing the KV cache for verified tokens, the model would need to recompute hidden states, leading to extra computation and latency. To address this, once the correct path is verified, its corresponding KV cache entries are copied into the primary cache buffer. In addition, all of the generated tokens and probabilities within the tree nodes are updated into the t-LUT and p-LUT accordingly.

Resource Efficiency-centric Computation Overlap

To maximize the computational efficiency, it is essential to fully utilize both CPU and GPU resources of edge devices. Although the dynamic token tree is updated at fixed intervals, accelerating this process is key to minimizing idle time and improving throughput.

To achieve this, we leverage CUDA stream-level parallelism to overlap GPU-based tasks, including LUT updates and KV cache rollback, with CPU-side tree construction. Each decoding iteration includes a forward pass on the GPU to predict tokens, verification of the maximum token path, and KV cache. Meanwhile, the CPU builds and updates the dynamic token tree using t-LUT and p-LUT. Without parallelism, these tasks would be executed sequentially, leading to inefficiencies. To address this, a CUDA asynchronous execution model is employed. Accordingly, verified KV cache rollback and LUT updates are offloaded to a background stream, allowing the main stream to continue with the next GPU operations while the CPU updates the tree in parallel. This design improves resource utilization, particularly on edge devices with limited computing capacity.

Experimental Evaluation

Experimental Settings

Models and Environments. We evaluate DIAA on three representative LLMs: Llama-3.2-3B-Instruct, Llama-3.2-1B-Instruct (Grattafiori et al. 2024), and Vicuna-7b-v1.5 (Zheng et al. 2023). Following prior work (Xia et al. 2025; Hu et al. 2025b; Luo et al. 2025), we focus on edge scenarios with greedy decoding (batch size 1) and Top-K set to 8. The t-LUT and p-LUT are built using the ShareGPT dataset (RyokoAI 2025), capturing common dialogue patterns.

All experiments run on the *NVIDIA Jetson Orin Nano Super* (NVIDIA 2025), a typical edge platform with 8 GB unified RAM, 256 GB SSD, and a 1024-core GPU, enabling 67 TOPS (INT8). Models are downloaded from Hugging Face (Wolf et al. 2020) and quantized to INT4 using GPTQ-Model (ModelCloud.ai and qubitium@modelcloud.ai 2024) to fit within Jetson's memory. The software configurations are Jetpack v6.2, CUDA v12.6, and PyTorch v2.7.0.

Datasets. We utilize Spec-Bench (Xia et al. 2024), a benchmark for SD evaluation across six tasks: Summarization (Sum), Math Reasoning (Math), Multi-turn Dialogue (MT), QA, Retrieval-Augmented Generation (RAG), and Machine Translation (Trans). The generation length is capped at 512 tokens for all tasks.

Baselines. To thoroughly evaluate DIAA, we compare it with five state-of-the-art inference baselines: **1) Vanilla**. Standard autoregressive generation without acceleration, serving as a performance baseline. **2) Speculative Sampling (SpS)** (Chen et al. 2023). A SD method that uses an auxiliary draft model to propose tokens before verification by the target model. The draft models of SpS used through the experiments are Llama-3.2-1B-Instruct-INT4 and Vicuna-160m. **3) SWIFT** (Xia et al. 2025). A plug-and-play, model-free self-SD method that speeds up inference by skipping redundant sub-layers during drafting. **4) SAM Decoding (SAM-D)** (Hu et al. 2025b). A retrieval-based approach that uses suffix automaton to select draft tokens by matching the current prompt with a pre-indexed corpus. **5) Token Recycling (TR)** (Luo et al. 2025). A model-free method that builds a static draft tree using breadth-first traversal of validated token paths, reusing candidates for speculation.

Metrics. To measure the performance of different decoding approaches, we adopt three evaluation metrics (Li et al. 2024b; Chen et al. 2025): **1) Goodput** (tokens/s) measures the number of accepted tokens generated per second, where accepted refers to tokens successfully verified by the target model. **2) Latency** (ms) is defined as the average wall-clock time per token required to produce each final token. It reflects the end-to-end delay experienced in practical applications. **3) Speedup ratio** compares the end-to-end inference latency of a given approach against the Vanilla baseline. We report the average gootput and latency through three trials.

Model	Baseline	Sum		Math		MT		QA		RAG		Trans		Average	
		G ↑	L ↓	G ↑	L ↓	G ↑	L ↓	G ↑	L ↓	G ↑	L ↓	G ↑	L ↓	G ↑	L ↓
7B	Vanilla	7.68	130.22	8.26	121.27	7.97	125.76	8.13	124.41	6.56	156.84	7.80	129.13	7.73	131.27
	SpS	10.00	104.65	8.30	123.54	9.15	118.16	9.84	113.34	8.25	134.48	6.98	164.80	8.75	126.50
	SWIFT	6.02	220.90	9.35	110.40	10.22	100.22	6.88	196.10	4.56	255.10	8.28	127.05	7.55	168.30
	SAM-D	**16.69**	**65.67**	15.97	64.48	14.42	72.54	11.60	90.33	**13.02**	**91.34**	9.80	103.29	13.58	81.28
	TR	9.08	111.82	13.56	74.84	10.82	93.32	10.30	99.85	7.63	139.56	9.69	107.03	10.18	104.40
	DIAA	13.28	76.18	**18.63**	**54.76**	**14.74**	**69.10**	**13.74**	**77.58**	10.60	103.07	**12.16**	**86.26**	**13.86**	**77.83**
3B	Vanilla	7.39	135.40	7.26	137.80	7.38	135.48	7.43	134.65	7.11	140.99	7.44	134.35	7.34	136.44
	SpS	8.93	114.55	10.01	100.91	9.63	105.45	9.65	105.09	9.31	112.65	9.71	104.32	9.54	107.16
	SWIFT	7.78	138.79	8.71	124.83	8.74	119.81	8.24	148.65	7.72	148.15	7.66	144.66	8.14	137.48
	SAM-D	14.94	67.95	13.63	74.90	13.63	75.56	13.83	73.99	**17.50**	**63.64**	13.80	74.76	14.56	71.80
	TR	13.23	76.50	17.25	58.39	15.56	64.84	16.77	59.89	12.95	78.75	16.52	60.74	15.38	66.52
	DIAA	**15.20**	**66.32**	**17.80**	**56.49**	**16.40**	**61.31**	**16.92**	**59.28**	15.09	67.75	**16.80**	**59.69**	**16.37**	**61.81**
1B	Vanilla	13.15	76.09	13.02	76.82	13.13	76.20	12.74	78.60	13.06	77.04	12.93	77.44	13.01	77.03
	SpS	11.84	86.34	13.45	75.62	12.53	81.60	12.46	82.04	12.72	82.33	12.56	82.88	12.59	81.80
	SWIFT	12.42	95.37	13.89	75.45	14.06	73.74	12.20	96.58	13.35	83.33	13.78	77.26	13.28	83.62
	SAM-D	24.67	42.73	25.14	41.76	26.34	39.13	**28.01**	**45.64**	**31.60**	**34.97**	24.17	45.15	26.65	41.56
	TR	21.79	47.10	28.23	35.99	24.81	40.53	25.36	41.73	22.92	46.99	24.29	41.76	24.57	42.35
	DIAA	**25.69**	**41.17**	**29.05**	**34.74**	**27.66**	**36.33**	27.94	39.15	28.66	37.04	**26.99**	**37.95**	**27.67**	**37.73**

Table 1: Performance evaluation of baselines and DIAA. (7B = Vicuna-7b-v1.5-INT4, 3B = Llama-3.2-3B-Instruct-INT4, and 1B = Llama-3.2-1B-Instruct. G = Goodput (tokens/s) and L = Latency (ms)).

Baseline	Llama-3.2-3B-Instruct-INT4	Vicuna-7b-v1.5-INT4	Llama-3.2-1B-Instruct
Vanilla	1.00×	1.00×	1.00×
SpS	1.28×	1.14×	0.94×
SWIFT	1.00×	0.97×	0.93×
SAM-D	1.91×	1.76×	1.87×
TR	2.08×	1.31×	1.83×
DIAA	**2.22**×	**1.78**×	**2.05**×

Table 2: Overall speedup ratio of different models.

Task	w/o LUT+DT	w/o LUT	w/o DT	w/o CO	DIAA
Sum	11.53	12.62	11.91	13.60	**14.92**
Math	15.09	15.40	15.33	16.25	**17.41**
MT	13.78	14.61	13.86	14.62	**16.45**
QA	14.64	15.03	14.67	15.63	**16.33**
RAG	11.34	13.16	11.61	13.42	**14.67**
Trans	14.47	15.19	14.86	15.44	**17.10**

Table 3: Ablation study of DIAA on goodput (tokens/s).

	Llama-3.2-3B-Instruct-INT4	Vicuna-7b-v1.5-INT4
vocab_size	128,256	32,000
Memory usage of t-LUT	7.83 MB	1.95 MB
Memory usage of p-LUT	3.91 MB	0.98 MB

Table 4: Memory usage of t-LUT and p-LUT.

Main Results and Analysis

Performance Evaluation of Goodput and Latency. Table 1 summarizes the goodput and latency results of DIAA compared to baselines across different models and tasks. DIAA consistently achieves top or near-top goodput in most tasks, outperforming baselines. Its advantage is especially clear in tasks with high token redundancy or reasoning demands, showing strong generalizability without relying on auxiliary models. Even when baselines like SAM-D perform well in specific tasks due to retrieval-based drafting, DIAA remains competitive, delivering stable and high goodput with lower variance across diverse workloads. In terms of latency, DIAA achieves consistently lower per-token latency compared to other approaches. It maintains stable performance across tasks and models, even on larger model Vicuna-7b, highlighting its efficiency for real-time and interactive applications. Unlike SpS and SWIFT, which depend on model alignment or redundancy of layers for larger models, DIAA provides reliable acceleration on both large and compact models, confirming its suitability for on-device deployment.

Performance Evaluation of Speedup ratio. We measure the speedup ratio of each method over Vanilla, averaged across six tasks, as shown in Table 2. DIAA consistently outperforms all baselines, achieving the superior speedup across different models. TR and SAM-D also perform well, benefiting from token reuse and retrieval strategies. In contrast, SpS and SWIFT show limited gains, as they rely on larger or more redundant models for effective drafting.

Ablation Study

Ablation Study of Key Components. To assess the contribution of each core component in DIAA, we perform an ablation study by disabling components individually: LUTs with LUT, the dynamic tree (DT), and the computation over-

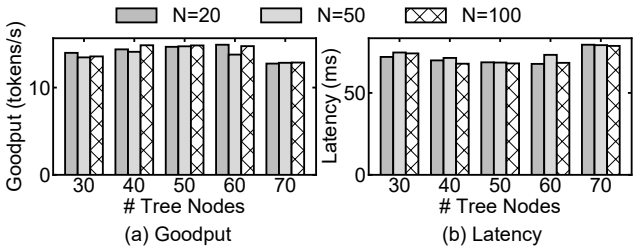

Figure 8: Impact on different number of tree nodes under different update intervals N.

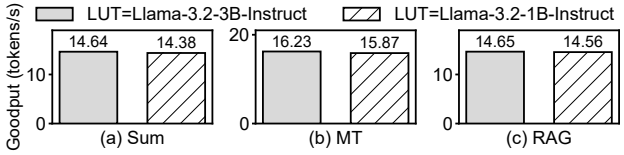

Figure 9: Impact on initialization of different LUTs.

lap (CO). Results for goodput is shown in Table 3. The results show that both the LUT and DT are crucial for improving inference efficiency. Removing either component leads to noticeable declines in goodput, with the DT proving particularly important for tasks that benefit from dynamic multi-path speculation. The CO also contributes by enhancing parallel execution. Overall, DIAA consistently outperforms its ablated versions, confirming the combination of LUTs, DT, and CO is essential for fast on-device inference.

Impact on Different Number of Tree Nodes. To study the sensitivity to dynamic draft tree configuration of DIAA, we vary the tree size (candidate node budget) and update interval (frequency of tree refreshing) in Fig. 8. Performance improves as the tree size increases, reaching optimal goodput and latency with a moderate number of nodes. This indicates that a balanced tree size offers sufficient speculative coverage without adding excessive verification overhead.

Impact on Initialization of LUTs. Table 4 compares memory usage of t-LUT (INT64) and p-LUT (FLOAT32). Llama-3.2-3B (vocabulary size 128,256) requires about 11.7 MB total, while Vicuna-7b (32,000 vocabulary) needs only 2.9 MB. This modest memory footprint makes LUTs highly suitable for resource-constrained edge devices. Intuitively, models with same vocabulary can also share LUTs, facilitating fast adaptation. Fig. 9 confirms similar performance across different LUT initializations, supporting the sharing of LUTs among models with same vocabulary.

Time Breakdown of DIAA during the Decoding Phase. Fig. 10 shows the time breakdown of DIAA during the decoding phase. The decoding forward operation dominates the most of computations, especially over 95% for Vicuna-7b-v1.5-INT4. The portion of dynamic tree update accounts for 0.15% on average, and the LUT update is 2.7%, which identifies the time efficiency of key components in DIAA.

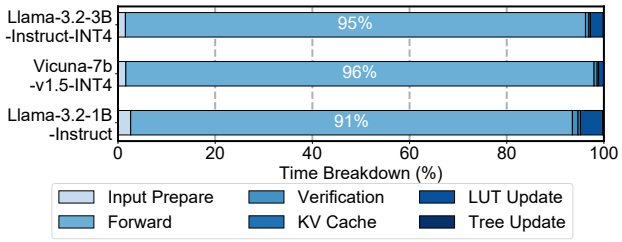

Figure 10: Time breakdown during the decoding phase.

Related Works

Speculative Decoding. Recent SD studies have explored efficient drafting to accelerate inference (Cai et al. 2024; Li et al. 2024b,a; Fu et al. 2024; Chen et al. 2025; Liu et al. 2024b; Zhang et al. 2024; Xia et al. 2025; Zarch et al. 2025; Luo et al. 2025; Hu et al. 2025b). Medusa (Cai et al. 2024) augments the target model with multiple lightweight prediction heads. The EAGLE series (Li et al. 2024b,a, 2025) modifies the target model to improve draft quality by addressing feature uncertainty. SPIN (Chen et al. 2025) uses multiple auxiliary draft models to enhance inference speed. To avoid relying on external draft models, some works propose self-SD approaches, reusing the target model for drafting (Zhang et al. 2024; Liu et al. 2024b; Xia et al. 2025). Luo et al. (Luo et al. 2025) introduce a model-free method with token reuse, while SAM Decoding (Hu et al. 2025b) accelerates inference through retrieval-based suffix automaton. However, most existing works do not focus on edge environments, where draft models add memory overhead or retraining costs. Moreover, static verification often leads to suboptimal performance when adapting to dynamic contexts.

Efficient Inference on Resource-Limited Devices. EdgeLLM (Xu et al. 2025) brings SD-enabled inference to mobile devices by combining draft and target models. EdgeShard (Zhang et al. 2025a) supports collaborative inference through cloud-edge model partitioning. Jupiter (Ye et al. 2025) accelerates inference using pipeline-level optimizations across heterogeneous edge devices. Additionally, model compression (Yu et al. 2024; Xing et al. 2025) and quantization (Shen et al. 2024; Lin et al. 2024; Yi et al. 2025) techniques are adopted in edge environments to decrease computational overhead that sacrifice model accuracy. In contrast, DIAA aims to enable lossless inference acceleration on edge devices without auxiliary models or offloading, making it well-suited for resource-limited environments.

Conclusion

In this paper, we introduce DIAA, a decoding-efficient inference acceleration approach for on-device LLMs. DIAA enables plug-and-play, model-agnostic decoding speedup on edge devices by employing paired LUTs for efficient token drafting with minimal memory. Furthermore, DIAA integrates a dynamic token tree to adapt real-time context and a stream-level pipeline to improve computation efficiency. Extensive experiments conducted on NVIDIA Jetson demonstrate superior performance of DIAA against baselines.

Acknowledgments

This research is supported part by the National Key Research and Development Program of China No. 2024YFE0204500, and also supported in part by the National Natural Science Foundation of China under Grant (No.92267104).

References

Cai, T.; Li, Y.; Geng, Z.; Peng, H.; Lee, J. D.; Chen, D.; and Dao, T. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *International Conference on Machine Learning*, 5209–5235. PMLR.

Chen, C.; Borgeaud, S.; Irving, G.; Lespiau, J.-B.; Sifre, L.; and Jumper, J. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Chen, F.; Li, P.; Luan, T. H.; Su, Z.; and Deng, J. 2025. SPIN: Accelerating Large Language Model Inference with Heterogeneous Speculative Models. In *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications, London, United Kingdom, May 19-22, 2025*, 1–10. IEEE.

Chen, Z.; May, A.; Svirschevski, R.; Huang, Y.-H.; Ryabinin, M.; Jia, Z.; and Chen, B. 2024. Sequoia: Scalable and robust speculative decoding. *Advances in Neural Information Processing Systems*, 37: 129531–129563.

Fu, Y.; Bailis, P.; Stoica, I.; and Zhang, H. 2024. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. In *International Conference on Machine Learning*, 14060–14079. PMLR.

Gao, S.; Lin, C.-H.; Hua, T.; Tang, Z.; Shen, Y.; Jin, H.; and Hsu, Y.-C. 2024. Disp-llm: Dimension-independent structural pruning for large language models. *Advances in Neural Information Processing Systems*, 37: 72219–72244.

Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Hu, M.; He, Q.; and Wu, D. 2025. QLLMS: Quantization-Adaptive LLM Scheduling for Partially Informed Edge Serving Systems. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, 1–10. IEEE.

Hu, S.; Zou, G.; Yang, S.; Lin, S.; Gan, Y.; Zhang, B.; and Chen, Y. 2025a. Large Language Model Meets Graph Neural Network in Knowledge Distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 17295–17304. AAAI Press.

Hu, Y.; Wang, K.; Zhang, X.; Zhang, F.; Li, C.; Chen, H.; and Zhang, J. 2025b. SAM Decoding: Speculative Decoding via Suffix Automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 12187–12204. Association for Computational Linguistics.

Huang, K.; Yin, X.; Huang, H.; and Gao, W. 2025. Modality plug-and-play: Runtime modality adaptation in LLM-driven autonomous mobile systems. In *The 31st Annual International Conference on Mobile Computing and Networking (ACM MOBICOM '25)*.

King, E.; Yu, H.; Lee, S.; and Julien, C. 2024. Sasha: creative goal-oriented reasoning in smart homes with large language models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(1): 1–38.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J.; Zhang, H.; and Stoica, I. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 611–626.

Lee, S.; Choi, J.; Lee, J.; Wasi, M. H.; Choi, H.; Ko, S.; Oh, S.; and Shin, I. 2024. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 1119–1133.

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 19274–19286. PMLR.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024a. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 7421–7432.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024b. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, 28935–28948.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2025. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100.

Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liu, F.; Tang, Y.; Liu, Z.; Ni, Y.; Tang, D.; Han, K.; and Wang, Y. 2024b. Kangaroo: Lossless self-speculative decoding for accelerating llms via double early exiting. *Advances in Neural Information Processing Systems*, 37: 11946–11965.

Luo, X.; Wang, Y.; Zhu, Q.; Zhang, Z.; Zhang, X.; Yang, Q.; and Xu, D. 2025. Turning Trash into Treasure: Accelerating Inference of Large Language Models with Token Recycling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 6816–6831. Association for Computational Linguistics.

Ma, X.; Fang, G.; and Wang, X. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36: 21702–21720.

Miao, X.; Oliaro, G.; Zhang, Z.; Cheng, X.; Wang, Z.; Zhang, Z.; Wong, R. Y. Y.; Zhu, A.; Yang, L.; Shi, X.; et al. 2024. 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, 932–949.

ModelCloud.ai; and qubitium@modelcloud.ai. 2024. GPTQModel. https://github.com/modelcloud/gptqmodel.

NVIDIA. 2025. Jetson Orin Nano Super Developer Kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/. Accessed: 2025-06.

RyokoAI. 2025. ShareGPT52K Hugging Face Dataset. https://huggingface.co/datasets/RyokoAI/ShareGPT52K. Accessed: 2025-05.

Shen, X.; Dong, P.; Lu, L.; Kong, Z.; Li, Z.; Lin, M.; Wu, C.; and Wang, Y. 2024. Agile-Quant: Activation-Guided Quantization for Faster Inference of LLMs on the Edge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 18944–18951. AAAI Press.

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

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, Y.; Zhang, D.; Wenren, H.; Wang, Y.; and Li, Y. 2025. EKD4Rec: Ensemble Knowledge Distillation from LLM-based Models to Traditional Sequential Recommenders. In *Companion Proceedings of the ACM on Web Conference 2025*, 1370–1374.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.

Xia, H.; Li, Y.; Zhang, J.; Du, C.; and Li, W. 2025. SWIFT: On-the-Fly Self-Speculative Decoding for LLM Inference Acceleration. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Xia, H.; Yang, Z.; Dong, Q.; Wang, P.; Li, Y.; Ge, T.; Liu, T.; Li, W.; and Sui, Z. 2024. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics ACL 2024*, 7655–7671. Bangkok, Thailand and virtual meeting: Association for Computational Linguistics.

Xing, X.; Liu, Z.; Xiao, S.; Gao, B.; Liang, Y.; Zhang, W.; Lin, H.; Li, G.; and Zhang, J. 2025. EfficientLLM: Scalable Pruning-Aware Pretraining for Architecture-Agnostic Edge Language Models. *arXiv preprint arXiv:2502.06663*.

Xu, D.; Yin, W.; Zhang, H.; Jin, X.; Zhang, Y.; Wei, S.; Xu, M.; and Liu, X. 2025. EdgeLLM: Fast On-Device LLM Inference With Speculative Decoding. *IEEE Transactions on Mobile Computing*, 24(4): 3256–3273.

Ye, S.; Ouyang, B.; Zeng, L.; Qian, T.; Chu, X.; Tang, J.; and Chen, X. 2025. Jupiter: Fast and resource-efficient collaborative inference of generative llms on edge devices. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, 1–10. IEEE.

Yi, R.; Guo, L.; Wei, S.; Zhou, A.; Wang, S.; and Xu, M. 2025. Edgemoe: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 24(8): 7059–7073.

Yu, Z.; Wang, Z.; Li, Y.; Gao, R.; Zhou, X.; Bommu, S. R.; Zhao, Y. K.; and Lin, Y. C. 2024. EDGE-LLM: Enabling Efficient Large Language Model Adaptation on Edge Devices via Unified Compression and Adaptive Layer Voting. In De, V., ed., *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC 2024, San Francisco, CA, USA, June 23-27, 2024*, 327:1–327:6. ACM.

Zarch, H. E.; Gao, L.; Jiang, C.; and Annavaram, M. 2025. DEL: Context-Aware Dynamic Exit Layer for Efficient Self-Speculative Decoding. *arXiv preprint arXiv:2504.05598*.

Zeng, C.; Liu, S.; Xie, Y.; Liu, H.; Wang, X.; Wei, M.; Yang, S.; Chen, F.; and Mei, X. 2025. Abq-llm: Arbitrary-bit quantized inference acceleration for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 22299–22307. AAAI Press.

Zhang, J.; Wang, J.; Li, H.; Shou, L.; Chen, K.; Chen, G.; and Mehrotra, S. 2024. Draft& Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. In Ku, L.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, 11263–11282. Association for Computational Linguistics.

Zhang, M.; Shen, X.; Cao, J.; Cui, Z.; and Jiang, S. 2025a. EdgeShard: Efficient LLM Inference via Collaborative Edge Computing. *IEEE Internet of Things Journal*, 12(10): 13119–13131.

Zhang, S.; Zhang, L.; Zhou, J.; Zheng, Z.; and Xiong, H. 2025b. LLM-Eraser: Optimizing Large Language Model Unlearning through Selective Pruning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, 1960–1971.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623.

Zheng, Y.; Chen, Y.; Qian, B.; Shi, X.; Shu, Y.; and Chen, J. 2025. A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57(8): 1–35.