RCMOE: EFFICIENT RETRIEVAL-CENTRIC MOE INFERENCE ON COMMODITY EDGE DEVICES

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE)-based large language models (LLMs) leverage sparse activation to increase model capacity without proportionally increasing per-token computation. However, their large parameter footprint and storage demands hinder deployment on resource-constrained edge devices. Existing on-device MoE approaches remain largely computation-centric, limiting the efficiency benefits of sparse activation under tight edge resource constraints. Although lookup-based MoE methods can replace expert computation with precomputed expert outputs, existing token-indexed schemes limit retrieval expressiveness and shift the bottleneck to storage overhead, fragmented execution, and data movement. To address these challenges, we propose RcMoE, a *retrieval-centric* on-device MoE inference framework that constructs a semantic-conditioned lookup table (SC-LUT) of precomputed expert outputs, thereby improving retrieval expressiveness. RcMoE further develops a lightweight retrieval execution pipeline with SC-LUT compression, a fused retrieval kernel, and asynchronous prefetching to reduce storage footprint, execution fragmentation, and data movement overhead. Experiments on commodity edge devices demonstrate that RcMoE achieves up to 31.1$\times$ the throughput of D^{2}MoE while maintaining competitive accuracy under edge resource constraints.

1 INTRODUCTION

Large language models (LLMs) (Brown et al., 2020; Touvron et al., 2023; Google DeepMind, 2025) have substantially advanced natural language understanding, generation, and reasoning. While LLM inference is predominantly deployed in cloud infrastructures with abundant computation and memory resources, there is increasing interest in moving inference toward edge devices (Yin et al., 2024; Xue et al., 2024). However, the computation and memory requirements of modern LLMs remain challenging for commodity edge platforms.

Recently, mixture-of-experts (MoE) (Shazeer et al., 2017; Fedus et al., 2022; Dai et al., 2024) have emerged as a promising direction for scaling LLM inference (Yi et al., 2025; Wang et al., 2025; Cao et al., 2025; Zhou et al., 2025). By activating only a subset of experts for each input, MoE increases model capacity without proportionally increasing per-token computation. However, sparse activation creates new challenges for commodity edge deployment. As Figure 1 illustrates, activated experts still execute online as $Y = X \cdot W$. Because experts account for most model parameters, memory-limited edge devices cannot keep them all resident. Efficient edge deployment must therefore minimize the memory footprint of experts and the data movement required to access them.

To reduce the memory footprint of large-scale models, researchers have explored quantization and CPU offloading. Aggressive 4-bit or 2-bit quantization (Frantar et al., 2023; Lin et al., 2024) reduces memory use but adds execution overhead. CPU offloading preserves model precision while easing memory limits, but on edge devices, unified memory architecture (UMA) and limited bus bandwidth constrain data movement. This bottleneck is amplified in MoE inference, where dynamically selected experts must be loaded on demand, making storage input/output (I/O) latency critical (Guo et al., 2023; Cao et al., 2025). Figure 2 shows that quantization and offloading reduce memory use but add dequantization or data-loading overhead, whereas lower latency generally requires keeping more model parameters resident. Both approaches still rely on online expert computation or parameter movement, motivating alternatives for edge MoE inference.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

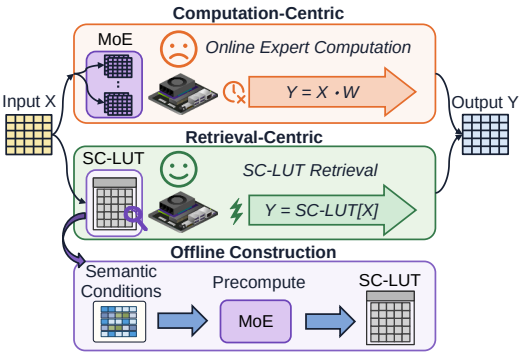

Figure 1: Paradigm shift in MoE inference on edge devices.

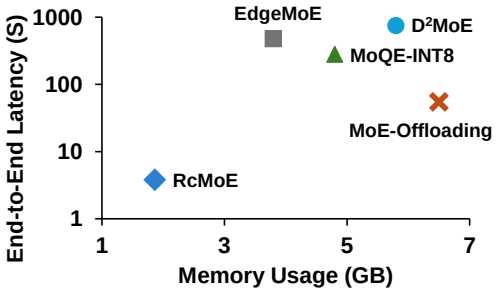

Figure 2: Memory-latency trade-off of existing on-device MoE inference methods.

Lookup-based MoE exploits the deterministic mapping between discrete input representations and expert outputs. It precomputes expert outputs for representative retrieval conditions, stores them in lookup tables (LUTs), and replaces online routed computation with retrieval. MoLE (Jie et al., 2025) demonstrates this approach by conditioning retrieval primarily on token identity. Identical tokens therefore access the same precomputed retrieval-expert vectors regardless of local semantic usage, limiting retrieval expressiveness. Full-precision LUTs also impose substantial storage costs on edge devices, which have limited storage capacity, memory, and bandwidth. In addition, fragmented lookup operators increase memory traffic and kernel launch overhead (Gale et al., 2023; Dao et al., 2022). Supporting richer retrieval conditions therefore requires addressing LUT storage, retrieval execution, and data movement together.

We formulate on-device lookup-based MoE as an end-to-end retrieval problem and propose RcMoE, a *retrieval-centric* framework that constructs a semantic-conditioned lookup table (SC-LUT) for context-dependent expert retrieval. RcMoE further develops a lightweight retrieval execution pipeline that addresses SC-LUT storage, retrieval execution, and data movement. Our contributions are as follows.

- **Semantic-Conditioned Lookup Table.** We construct a semantic-conditioned lookup table (SC-LUT) that stores precomputed expert outputs under semantic-conditioned retrieval conditions, improving retrieval expressiveness.
- **Lightweight Retrieval Execution Pipeline.** We develop a lightweight retrieval execution pipeline that integrates SC-LUT compression, a fused retrieval kernel, and asynchronous prefetching to reduce storage footprint, execution fragmentation, and data movement overhead.
- **Practical Edge Deployment.** We implement RcMoE for commodity edge deployment and evaluate it on NVIDIA Jetson Orin Nano 8 GB, where it achieves up to 31.1× the throughput of D^{2}MoE with competitive accuracy.

2 Background and Motivation

On-Device MoE Inference. Existing on-device LLM systems address resource constraints through quantization, offloading, and storage-aware execution (Yin et al., 2024; Sheng et al., 2023; Xue et al., 2024). For MoE models, recent studies reduce expert execution and loading overhead. MoE-Lightning (Cao et al., 2025) reduces expert swapping through prefetching, while D^{2}MoE (Wang et al., 2025) and EdgeMoE (Yi et al., 2025) use adaptive quantization to reduce expert storage and loading costs. MobileMoE (Chen et al., 2026) explores scaling through architecture, training, and deployment choices. However, these approaches still optimize online expert execution, parameter storage, and data movement rather than alter the computation-centric paradigm.

Lookup-based Inference. Lookup-based inference has been explored to replace arithmetic computation with table lookup for efficient LLM deployment. Existing approaches mainly focus on

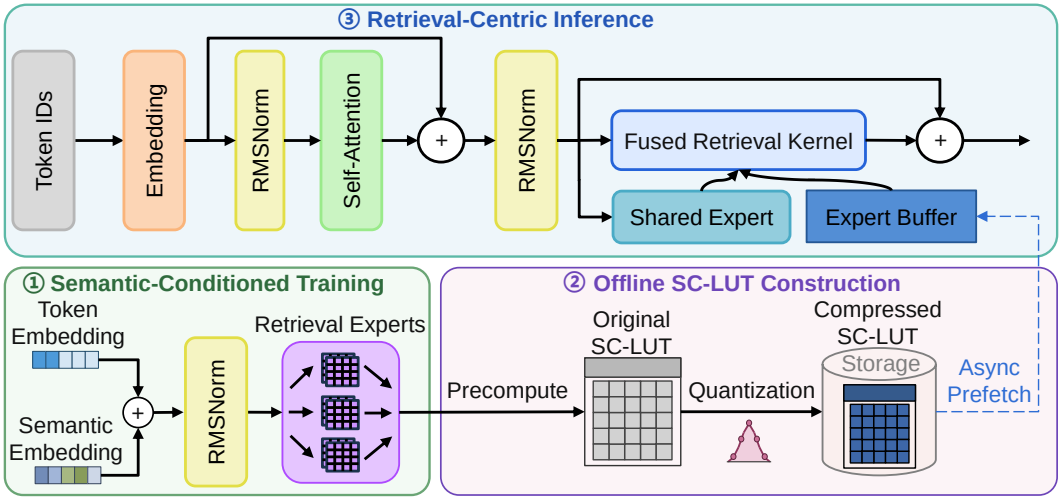

Figure 3: Overview of RcMoE. The framework consists of three stages: Semantic-conditioned training of retrieval experts using token and semantic-pattern embeddings, offline SC-LUT construction from precomputed expert outputs, and retrieval-centric inference with the fused retrieval kernel and asynchronous prefetching.

low-bit matrix computation through optimized LUT construction, memory access, and hardware execution. T-MAC (Wei et al., 2025), Vec-LUT (Li et al., 2026), and LUT-GEMM (Park et al., 2024) improve lookup-based matrix computation on CPUs, edge devices, and graphics processing units (GPUs), while LUT Tensor Core (Mo et al., 2025), CodeQuant (Yin et al., 2026), and OASIS (Wu et al., 2026) investigate hardware-oriented LUT execution. Different from lookup-based matrix acceleration, MoLE (Jie et al., 2025) applies lookup-based reparameterization to MoE inference by precomputing token-indexed expert outputs. MoLE conditions retrieval primarily on token identity, so identical tokens access the same precomputed retrieval-expert vectors regardless of local semantic usage. RcMoE enriches this retrieval condition with local semantic patterns while addressing the resulting storage, execution, and data movement costs.

3 Method

3.1 Retrieval-Centric MoE Overview

As shown in Figure 3, RcMoE consists of three stages. (1) **Semantic-Conditioned Training**: retrieval experts are trained using semantic-conditioned representations. (2) **Offline SC-LUT Construction**: retrieval expert outputs are precomputed and compressed into SC-LUT. (3) **Retrieval-Centric Inference**: fused retrieval kernel reconstructs SC-LUT entries, while asynchronous prefetching overlaps storage access with computation.

3.2 Semantic-Conditioned Training

Token-Indexed Retrieval. Lookup-based MoE methods replace online expert computation with precomputed expert outputs by assigning each retrieval condition a unique identifier. For a vocabulary $\mathcal{V}$ and token $v \in \mathcal{V}$, the embedding layer provides a deterministic mapping $v \rightarrow \mathbf{e}_v$. Based on this property, token-indexed retrieval defines the retrieval identifier as $r = f(v)$, where r selects the precomputed expert outputs associated with token v. Because this retrieval condition depends only on token identity, identical tokens access the same precomputed retrieval-expert vectors regardless of local semantic usage, limiting retrieval expressiveness.

Semantic-Conditioned Representation. To improve retrieval expressiveness beyond token identity, RcMoE introduces semantic conditioning by incorporating token co-occurrence patterns into the retrieval condition. Specifically, RcMoE extracts frequent n-grams from the training corpus and assigns each selected pattern a unique identifier $m \in \mathcal{M}$. Candidate n-grams are ranked using

normalized pointwise mutual information (NPMI), which measures the association strength among tokens. For an n-gram $(w_1, \ldots, w_n)$, where $n \in \{2, 3\}$, NPMI is defined as:

$$\mathrm{NPMI}_n(w_1, \ldots, w_n) = \frac{\log \frac{P(w_1,\ldots,w_n)}{\prod_{i=1}^{n} P(w_i)}}{-\log P(w_1, \ldots, w_n)}. \tag{1}$$

Corpus counts estimate probabilities, and scores are clipped to $[-1, 1]$. High-NPMI bigrams and trigrams form vocabulary $\mathcal{M}$, with each pattern assigned an identifier m and embedding $\mathbf{c}_m$.

For token v with semantic pattern identifier m, RcMoE forms the semantic-conditioned retrieval representation:

$$\mathbf{z}_{v,m} = \mathbf{e}_v + \mathbf{c}_m, \tag{2}$$

where $\mathbf{e}_v$ denotes the token embedding and $\mathbf{c}_m$ represents the semantic pattern embedding. The token identifier v and pattern identifier m jointly determine two SC-LUT rows, whose indices are:

$$r_v = v, \qquad r_m = |\mathcal{V}| + m. \tag{3}$$

If no semantic pattern is identified, RcMoE uses $m = 0$ and sets $\mathbf{c}_0 = \mathbf{0}$, so the zero residual row is added to the token row. Semantic patterns use identifiers $m \in \{1, \ldots, |\mathcal{M}|\}$, so $m = 0$ selects the reserved zero fallback row at index $|\mathcal{V}|$ through $r_m = |\mathcal{V}| + m$.

Retrieval Expert Modeling. We consider an L-layer MoE model where each layer l consists of a shared expert $E_{l,s}$ and a set of N_r retrieval experts $\{E_{l,k}\}_{k=1}^{N_r}$ with hidden dimension d. For an input token v at layer l, RcMoE uses the contextual hidden state $\mathbf{x}_l$ from the previous layer and the semantic-conditioned retrieval representation $\mathbf{z}_{v,m}$. For notational simplicity, $\mathbf{x}_l$ denotes both the residual-stream state and the normalized hidden state supplied to the shared expert and router. The shared expert $E_{l,s}$ operates on $\mathbf{x}_l$, preserving the original contextual representation pathway, while retrieval experts operate on $\mathrm{Norm}(\mathbf{z}_{v,m})$, where $\mathrm{Norm}(\cdot)$ denotes the normalization operation. The routing network dynamically computes aggregation weights based on $\mathbf{x}_l$, allowing retrieved expert outputs to retain contextual information through weighted aggregation. During training, the MoE output is computed as:

$$\mathbf{Y}_{l+1} = \mathbf{x}_l + E_{l,s}(\mathbf{x}_l) + \sum_{k=1}^{N_r} \left(g_{l,k}(\mathbf{x}_l) \cdot E_{l,k}(\mathrm{Norm}(\mathbf{z}_{v,m})) \right). \tag{4}$$

3.3 Offline SC-LUT Construction

SC-LUT Generation. After training, RcMoE performs offline forward passes for each retrieval representation at every MoE layer. Each joint output is stored as the sum of a token row and a pattern residual row:

$$\mathbf{SC\text{-}LUT}[r_v, l, k] + \mathbf{SC\text{-}LUT}[r_m, l, k] = E_{l,k}(\mathrm{Norm}(\mathbf{z}_{v,m})), \tag{5}$$

For each pattern m ending in v, the token row stores $E_{l,k}(\mathrm{Norm}(\mathbf{e}_v))$ and the pattern row stores the difference between joint and token-only outputs; no match uses a zero residual. The table has $|\mathcal{V}| + |\mathcal{M}|$ retrieval entries, plus one zero fallback row, and shape $(|\mathcal{V}|+|\mathcal{M}|+1)\times L\times N_r\times d$. We generate and compress the SC-LUT offline. Although it grows with the number of retrieval conditions, each token loads only a small amount of SC-LUT data, keeping inference-time movement lightweight. Large SC-LUTs can be further compressed using low-rank factorization or hashed indexing.

SC-LUT Compression. Although SC-LUT retrieval eliminates online routed expert computation, storing retrieval expert outputs in external storage introduces substantial storage overhead and increases data movement, limiting edge deployment scalability. To enable efficient retrieval-centric inference, RcMoE compresses SC-LUT to reduce storage footprint and retrieval data movement. Specifically, for each SC-LUT row index j, RcMoE applies the NormalFloat-4 (NF4) quantization operator $Q_{\mathrm{NF4}}(\cdot)$ (Dettmers et al., 2023):

$$\mathbf{SC\text{-}LUT}^q[j, l, k] = Q_{\mathrm{NF4}}(\mathbf{SC\text{-}LUT}[j, l, k]). \tag{6}$$

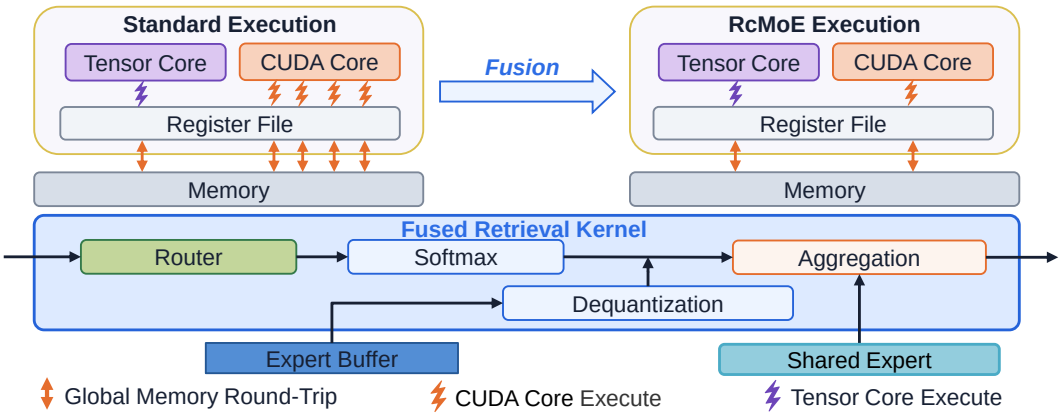

Figure 4: Fused retrieval kernel combining routing, softmax, dequantization, and expert aggregation to reduce kernel launches and global memory round-trips.

3.4 Retrieval-Centric Inference

RcMoE develops a lightweight retrieval execution pipeline to efficiently deploy retrieval-centric inference on edge devices. The pipeline integrates compressed SC-LUT, a fused retrieval kernel, and asynchronous prefetching to optimize storage efficiency, retrieval execution, and data movement.

SC-LUT Retrieval. During inference, RcMoE retains the token ID sequence for n-gram matching. It matches bigrams and trigrams ending at the current token against the semantic pattern vocabulary to determine m. The token ID v and pattern ID m jointly select two SC-LUT rows, indexed by r_v and r_m. When multiple patterns match, RcMoE selects the longest pattern as the semantic condition. If no pattern is found, RcMoE uses the reserved fallback identifier $m = 0$. The two selected rows are dequantized and summed before aggregation:

$$\mathbf{SC\text{-}\hat{L}UT}[v, m, l, k] = Q_{\mathrm{NF4}}^{-1}(\mathbf{SC\text{-}LUT}^{q}[r_v, l, k]) + Q_{\mathrm{NF4}}^{-1}(\mathbf{SC\text{-}LUT}^{q}[r_m, l, k]). \tag{7}$$

RcMoE replaces online routed expert computation with SC-LUT retrieval, while the shared expert continues to operate on the hidden state $\mathbf{x}_l$. The MoE output is computed as:

$$\hat{\mathbf{Y}}_{l+1} = \mathbf{x}_l + E_{l,s}(\mathbf{x}_l) + \sum_{k=1}^{N_r} g_{l,k}(\mathbf{x}_l) \cdot \mathbf{SC\text{-}\hat{L}UT}[v, m, l, k]. \tag{8}$$

Fused Retrieval Kernel. Although SC-LUT retrieval eliminates routed expert computation, its execution remains fragmented: routing, softmax, dequantization, and aggregation run as separate kernels, adding launches and memory traffic (Figure 4).

To reduce this overhead, RcMoE fuses routing, softmax, SC-LUT dequantization, and expert aggregation in one Compute Unified Device Architecture (CUDA) kernel (Figure 4). It retrieves compressed entries from the expert buffer, unpacks codes through codebook lookup, and accumulates weighted expert outputs in registers. The shared expert remains a separate optimized linear-algebra operation and joins the retrieval output during aggregation. This avoids intermediate memory writes and reduces kernel launches and memory traffic for edge inference.

Asynchronous Prefetching. Edge devices typically rely on storage media such as Non-Volatile Memory Express (NVMe) solid-state drives (SSDs) or embedded MultiMediaCard (eMMC) flash, which provide lower bandwidth and higher access latency than server-class memory systems. Under synchronous execution, each MoE layer must wait for the required SC-LUT entries to be fetched from storage, causing computation stalls even when the transferred data size is small. To reduce this overhead, RcMoE employs asynchronous prefetching to overlap storage access with MoE-layer computation. Specifically, while the current layer is executed, a background thread prefetches the next layer's compressed SC-LUT entries $\mathbf{SC\text{-}\tilde{L}UT}^{q}[r_v, l+1, k]$ and $\mathbf{SC\text{-}\tilde{L}UT}^{q}[r_m, l+1, k]$ into the double-buffered expert buffer. The execution latency of the l-th MoE layer can be expressed as:

$$T_{\mathrm{MoE}}^{(l)} = \max\left(T_s^{(l)} + T_{\mathrm{kernel}}^{(l)},\ T_{\mathrm{I/O}}^{(l+1)}\right), \tag{9}$$

Table 1: Model configurations.

Backbone	Arch.	Total Params	Activated Params	N	K	L	d	d_r
SmolLM2-360M	Dense	360M	360M	1	1	30	960	2560
	MoE	2.01B	598M	8	2	30	960	2560
	RcMoE	2.01B	360M	8	1+7	30	960	2560
SmolLM2-1.7B	Dense	1.71B	1.71B	1	1	24	2048	8192
	MoE	5.30B	2.92B	4	2	24	2048	8192
	RcMoE	5.30B	1.71B	4	1+3	24	2048	8192

where $T_s^{(l)}$ represents the execution time of the shared expert, $T_{\text{kernel}}^{(l)}$ is the execution time of the fused retrieval kernel, and $T_{\text{I/O}}^{(l+1)}$ denotes the time required to fetch the next layer's SC-LUT entries from storage.

Retrieval identifiers select the compressed SC-LUT entries. Because retrieval experts are fully active, RcMoE needs no expert prediction or selection and overlaps next-layer I/O with current-layer computation, reducing visible storage-I/O latency.

4 Experiments

4.1 Experimental Setup

Training. We implement RcMoE in PyTorch and train on 8 NVIDIA RTX 4090 GPUs with 24 GB VRAM using DeepSpeed's Zero Redundancy Optimizer (ZeRO) stage 2 (ZeRO-2) (Rajbhandari et al., 2020). AdamW uses a 3×10^{-4} peak learning rate, cosine decay, and 10% warmup. Training runs for 2,000 continued pretraining (CPT) steps and 1,000 supervised fine-tuning (SFT) steps. Variants at each scale share this setup.

Inference. We evaluate performance on an NVIDIA Jetson Orin Nano 8 GB with 68 GB/s memory bandwidth, paired with a ZHITAI TiPlus 7100 SSD. The device is configured under a 15 W power mode. All methods use the same hardware and configuration.

Models. As summarized in Table 1, we adopt SmolLM2-360M and SmolLM2-1.7B (Allal et al., 2025) as dense backbones, where N, K, L, d, and d_r denote the number of experts, active expert configuration, layers, hidden dimension, and expert intermediate dimension, respectively. RcMoE matches the MoE models in total parameter capacity while activating fewer parameters through SC-LUT retrieval. The evaluated model variants are defined as follows.

- **Dense**: We evaluate the original dense backbone as a non-MoE reference for model quality and inference efficiency.
- **MoE**: We upcycle the dense backbones (He et al., 2024) into MoE models with $N = 8$ or $N = 4$ experts per layer and Top-2 routing ($K = 2$), following Mixtral (Jiang et al., 2024).
- **RcMoE**: RcMoE freezes the original multilayer perceptron (MLP) layers as shared experts and initializes 7 or 3 additional retrieval experts, respectively, yielding $K = 1 + 7$ or $1+3$. The retrieval expert outputs are precomputed and stored in the SC-LUT for inference. RcMoE uses a semantic pattern vocabulary of 50K patterns selected by NPMI.

Datasets. CPT uses FineWeb-Edu (Penedo et al., 2024), FineMath-4plus (Initiative, 2024), and The Stack Python (Kocetkov et al., 2022). SFT uses UltraChat-200k (Ding et al., 2023).

Evaluation Metrics. We report average zero-shot accuracy on ARC-Challenge, HellaSwag, PIQA, WinoGrande, BoolQ, and OpenBookQA (Clark et al., 2018; Zellers et al., 2019; Bisk et al., 2020; Sakaguchi et al., 2020; Clark et al., 2019; Banerjee et al., 2019) using lm-evaluation-harness (Gao et al., 2024). We report the average LongBench score at 8,192 tokens (Bai et al., 2024). Efficiency metrics include offloaded and per-token loaded sizes, peak memory, generation latency, and throughput (tokens/s). Appendices A–E detail implementation, Ascend, task-level, and system re-

Table 2: End-to-end performance of different methods on NVIDIA Jetson Orin Nano.

Model	Method	Avg. Acc. ↑	Avg. Long. ↑	Activated Params.	Offloaded Params.	Offloaded Size ↓	Loaded Size per Token ↓	Peak Memory Usage ↓	Latency (s) ↓	Throughput (tokens/s) ↑
Dense-360M	In-Memory	54.10	9.56	360M	–	–	–	0.70 GB	6.73	9.52
MoE-598M	In-Memory	54.32	9.74	598M	–	–	–	3.86 GB	14.53	4.40
	MoE-Offloading	54.32	9.74	598M	1.89B	3.52 GB	900 MB	1.22 GB	45.31	1.41
	EdgeMoE	43.87	3.69	598M	1.89B	1.76 GB	450 MB	0.65 GB	56.49	1.13
	MoQE-INT8	54.06	9.67	598M	1.89B	1.76 GB	450 MB	0.80 GB	35.36	1.81
	D^2MoE	53.57	9.70	598M	1.89B	1.76 GB	450 MB	0.96 GB	62.64	1.02
	MoLE	54.33	9.86	360M	10.57B	19.69 GB	0.41 MB	0.75 GB	9.96	6.43
	RcMoE (Ours)	54.66	9.89	360M	21.32B	9.93 GB	0.21 MB	0.74 GB	9.10	7.04
Dense-1.7B	In-Memory	62.96	10.51	1.71B	–	–	–	3.45 GB	5.77	11.10
MoE-2.9B	In-Memory	63.25	10.55	2.92B	OOM	OOM	OOM	OOM	OOM	OOM
	MoE-Offloading	63.25	10.55	2.92B	4.83B	9.00 GB	4.50 GB	5.65 GB	157.98	0.41
	EdgeMoE	51.91	6.59	2.92B	4.83B	4.50 GB	2.25 GB	2.79 GB	155.01	0.41
	MoQE-INT8	62.29	10.91	2.92B	4.83B	4.50 GB	2.25 GB	3.52 GB	175.25	0.37
	D^2MoE	61.93	10.89	2.92B	4.83B	4.50 GB	2.25 GB	3.55 GB	207.76	0.31
	MoLE	63.47	10.85	1.71B	7.25B	13.50 GB	0.28 MB	3.45 GB	7.14	8.96
	RcMoE (Ours)	63.55	11.44	1.71B	14.62B	6.81 GB	0.14 MB	3.45 GB	6.64	9.64

Note: Prompt length = 16, generation length = 64, batch size = 1.

sults. Appendix E also reports semantic-vocabulary scaling, one-time SC-LUT construction cost, and energy measurements.

Baselines. We compare RcMoE with MoE inference baselines for edge deployment, including in-memory execution, expert offloading, expert quantization, and lookup-based expert retrieval.

- **In-Memory**: All expert weights remain resident in bfloat16 (BF16) precision, providing a full-precision reference without storage I/O.
- **MoE-Offloading**: Expert weights reside in external storage and are loaded on demand with a least recently used (LRU) cache, incurring I/O for full-precision transfers.
- **EdgeMoE (Yi et al., 2025)**: Offline sensitivity analysis assigns static INT2, INT4, or INT8 precision to experts, reducing I/O with potential accuracy loss.
- **MoQE-INT8 (Kim et al., 2023)**: MoQE-INT8 applies uniform INT8 quantization to all experts, reducing expert storage and I/O volume compared with full-precision offloading.
- **D^2MoE (Wang et al., 2025)**: D^2MoE adopts matryoshka weight quantization (MWQ) and dynamically assigns bit-widths (INT5/6/7/8) according to expert importance, followed by runtime dequantization to half-precision floating-point (FP16).
- **MoLE (Jie et al., 2025)**: MoLE is a lookup-based MoE baseline that replaces online retrieval expert computation with token-indexed lookup.

4.2 End-to-end Results

End-to-end Performance. Table 2 reports accuracy, LongBench score, storage, peak memory, latency, and throughput. The reported end-to-end latency and throughput are based on repeated measurements. Avg. Acc. denotes average zero-shot accuracy and Avg. Long. denotes the average LongBench score. For lookup-based methods, Offloaded Params. counts stored LUT elements rather than trainable model parameters, including the 1,024-wide padded layout of RcMoE-360M. Per-token loaded size is the worst-case amount loaded for each generated token. RcMoE maintains competitive accuracy while reducing storage and data movement. In the MoE-598M group, it achieves the highest average accuracy (54.66) and LongBench score (9.89), while reducing per-token loaded size to 0.21 MB and offloaded size to 9.93 GB, compared with 0.41 MB and 19.69 GB for MoLE. It also lowers latency and increases throughput relative to MoLE, with 0.74 GB peak memory. In the MoE-2.9B group, in-memory execution exceeds device capacity and triggers out-of-memory (OOM). RcMoE reaches 63.55 average accuracy and 11.44 LongBench, while halving per-token loaded and offloaded sizes relative to MoLE. RcMoE achieves 31.1× the throughput of D^2MoE with 3.45 GB peak memory, and has the lowest latency among the MoE methods. The dense 1.7B backbone has higher throughput in this setting. RcMoE instead targets MoE capacity under limited active computation and memory while avoiding repeated expert-weight transfers.

Batch Size Scalability. Figure 5 compares end-to-end latency and throughput across batch sizes. Offloading-based baselines scale poorly because expert-weight transfers remain on the critical path. In the MoE-598M group, D^{2}MoE reaches 345.24 s latency at batch size 64, whereas RcMoE stays below 10 s and reaches 423.33 tokens/s. For MoE-2.9B, in-memory execution fails due to OOM, while RcMoE latency remains nearly constant, increasing from 6.64 s to 6.82 s as batch size grows from 1 to 8. Throughput rises from 9.64 to 75.10 tokens/s. This 7.8× throughput gain comes with a 2.7% latency increase, indicating that processing more tokens concurrently does not proportionally increase end-to-end time. Because throughput counts tokens across the batch, this gain reflects serving more sequences concurrently rather than a comparable reduction in each sequence's latency. RcMoE improves batch-size scalability by reducing storage I/O during retrieval-centric inference.

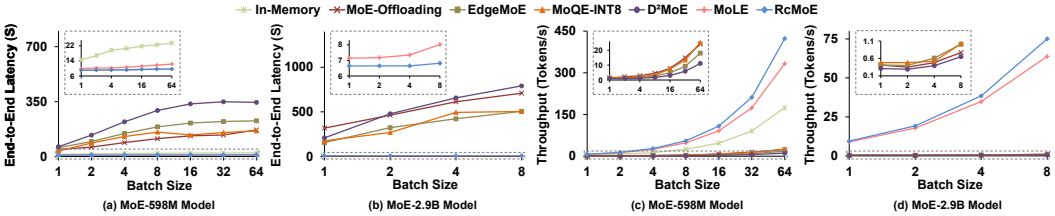

Figure 5: Latency and throughput across batch sizes (dashed boxes indicate zoomed-in views).

Output Length Scalability. Figures 6 and 7 evaluate the scalability of RcMoE. Figure 6(a) shows that RcMoE achieves the lowest latency across all output lengths in the MoE-598M group and outperforms the In-Memory baseline. As the output length increases from 32 to 256 tokens, RcMoE latency increases from 4.80 s to 37.80 s, while D^{2}MoE reaches 227.39 s due to repeated expert data loading. Figure 6(b) demonstrates that RcMoE maintains near-linear latency growth and reaches 44.79 s at an output length of 256 tokens, whereas MoE-Offloading increases to 590.68 s. The throughput results are shown in Figure 7. In the MoE-598M group, RcMoE maintains throughput above 6.7 tokens/s across all output lengths, outperforming offloading-based baselines and the In-Memory baseline. In the MoE-2.9B group, RcMoE throughput increases from 4.50 to 5.72 tokens/s as the output length grows, while D^{2}MoE and MoQE-INT8 remain at lower throughput levels due to storage I/O overhead. RcMoE maintains inference efficiency as generation length increases.

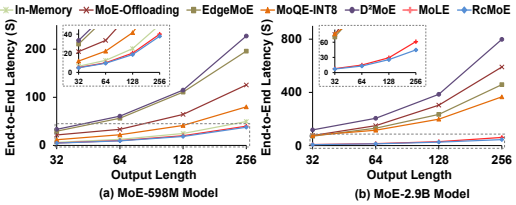

Figure 6: Latency across output lengths.

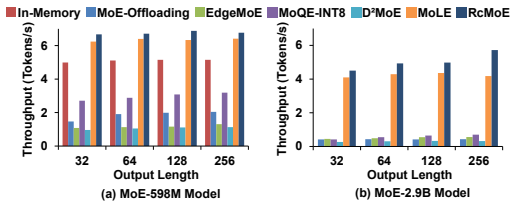

Figure 7: Throughput across output lengths.

4.3 Ablation Study

Architecture Ablation. We assess how semantic retrieval and execution optimizations affect model quality, storage efficiency, and inference performance. Table 3 compares the baselines, Semantic Retrieval, components, and their combination. MoE (Baseline) performs expert computation, requiring offloading at the 1.7B scale. MoLE replaces this computation with token-indexed retrieval, whose retrieval-expert vectors cannot distinguish local semantic uses of the same token. Semantic Retrieval shows modest gains in accuracy and LongBench scores over MoLE at both scales, at the cost of a larger BF16 SC-LUT as the retrieval condition space expands. Applied to Semantic Retrieval, NF4 quantization reduces SC-LUT storage from 39.71 to 9.93 GB at 360M and from 27.23 to 6.81 GB at 1.7B, while lowering throughput because of dequantization. The fused kernel and asynchronous prefetching, evaluated without NF4, improve throughput without changing SC-LUT storage. Combining all three components raises throughput relative to NF4 quantization alone, reaching 7.04 and

9.64 tokens/s, respectively. RcMoE exceeds BF16 Semantic Retrieval in throughput at 360M but remains slightly slower at 1.7B, where the execution optimizations do not offset NF4's cost.

Table 3: Architecture ablation. Effects of individual and combined components.

Backbone Scale	Configurations	Avg. Acc. ↑	Avg. Long. ↑	Offloaded Size ↓	Latency (s) ↓	Throughput (Tokens/s) ↑
360M	MoE (Baseline)	54.32	9.74	–	14.53	4.40
	MoLE	54.33	9.86	19.69 GB	9.96	6.43
	Semantic Retrieval	54.77	9.97	39.71 GB	9.96	6.43
	+ NF4 Quantization	54.66	9.89	9.93 GB	10.90	5.87
	+ Fused Kernel	54.77	9.97	39.71 GB	9.58	6.68
	+ Async Prefetch	54.77	9.97	39.71 GB	9.32	6.87
	Total (RcMoE)	**54.66**	**9.89**	**9.93** GB	**9.10**	**7.04**
1.7B	MoE (Baseline)	63.25	10.55	9.00 GB	157.98	0.41
	MoLE	63.47	10.85	13.50 GB	7.14	8.96
	Semantic Retrieval	63.54	11.44	27.23 GB	6.49	9.86
	+ NF4 Quantization	63.55	11.44	6.81 GB	12.40	5.16
	+ Fused Kernel	63.54	11.44	27.23 GB	6.15	10.41
	+ Async Prefetch	63.54	11.44	27.23 GB	5.92	10.81
	Total (RcMoE)	**63.55**	**11.44**	**6.81** GB	**6.64**	**9.64**

Layer-wise Execution Breakdown. Figure 8 compares mean per-layer latency over 100 forward passes for BF16, unfused NF4, and the fused RcMoE execution path. At batch size 8, NF4 raises RcMoE-360M latency from 1.97 to 2.64 ms because dequantization offsets the savings in SC-LUT I/O. Fusing dequantization and aggregation reduces latency to 1.62 ms, a 1.63× speedup over unfused NF4. At 1.7B, the total latency rises from about 5.0 ms with BF16 to 6.4 ms with unfused NF4, then falls to 3.8 ms with RcMoE. With asynchronous prefetching, the remaining 0.08 ms reflects PyTorch kernel-launch overhead rather than exposed LUT I/O. Thus, compression alone does not improve execution speed. Fusion and prefetching are needed to offset its runtime cost.

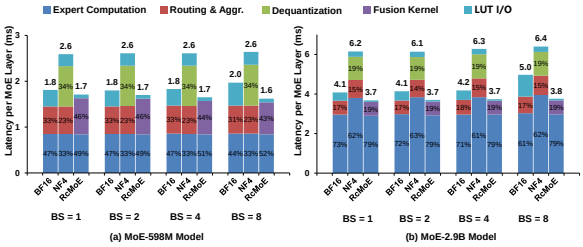

Figure 8: Latency breakdown across batch sizes.

Table 4: SC-LUT precision ablation.

Model	Precision	Avg. Acc. ↑	Avg. Long. ↑	Size (GB) ↓
RcMoE-360M	BF16	54.77	9.97	39.71
	NF4	54.66	9.89	9.93
	INT4	54.45	9.79	9.93
	NF3	54.51	9.83	7.45
RcMoE-1.7B	BF16	63.54	11.44	27.23
	NF4	63.55	11.44	6.81
	INT4	63.45	11.06	6.81
	NF3	63.51	11.29	5.11

SC-LUT Precision Ablation. Table 4 studies the impact of SC-LUT precision on model quality and storage efficiency. BF16 provides the uncompressed reference but incurs SC-LUT storage overhead. NF4 reduces the SC-LUT size by 4.0× while maintaining comparable accuracy across both model scales, with a 0.11 percentage point drop for RcMoE-360M and a slight improvement for RcMoE-1.7B. Although NF3 provides higher compression, it introduces additional accuracy degradation. Therefore, NF4 achieves the best trade-off between SC-LUT compression and retrieval quality among the evaluated precisions and is adopted as the default precision for RcMoE.

5 Conclusion

In this work, we present RcMoE, a retrieval-centric inference framework for deploying MoE models on commodity edge devices. RcMoE replaces routed expert computation with a semantic-conditioned lookup table and enables efficient retrieval execution through fused kernels and asynchronous prefetching. Experiments demonstrate that RcMoE achieves efficient MoE deployment on commodity edge devices with reduced storage and data movement overhead while maintaining competitive accuracy.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing of this paper, including improving grammar, clarity, conciseness, and style. We did not use generative AI for research ideation, theoretical or mathematical contributions, methodology or experiment design, method implementation, data generation, or generation of novel results. Other required research tasks were not applicable. The authors reviewed all AI-assisted edits and take responsibility for the final content.

REPRODUCIBILITY STATEMENT

Section 3 describes the SC-LUT construction, compression, fused retrieval kernel, and asynchronous prefetching used by RcMoE. Section 4 reports the model configurations, training setup, datasets, evaluation tasks and metrics, hardware platforms, and baseline settings used in the experiments. Appendices A and B provide the detailed training, evaluation, quantization, vocabulary-construction, and runtime protocols. The remaining appendices report task-level and cross-platform measurements.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlícek, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big - data-centric training of a small language model. arXiv preprint arXiv:2502.02737, 2025. URL https://arxiv.org/abs/2502.02737.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Pratyay Banerjee, Kuntal Kumar Pal, Arindam Mitra, and Chitta Baral. Careful selection of knowledge to solve open book question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

Yanbei Chen, Hanxian Huang, Ernie Chang, Jacob Szwejbka, Digant Desai, Zechun Liu, Vikas Chandra, and Raghuraman Krishnamoorthi. Mobilemoe: Scaling on-device mixture of experts. arXiv preprint arXiv:2605.27358, 2026. URL https://arxiv.org/abs/2605.27358.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018. URL https://arxiv.org/abs/1803.05457.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23:1–39, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Zenodo software release, version v0.4.3, 2024. URL https://doi.org/10.5281/zenodo.12608602.

Google DeepMind. Gemini 2.5 Pro. Google DeepMind website, 2025. URL https://deepmind.google/gemini/.

Liwei Guo, Wonkyo Choe, and Felix Xiaozhu Lin. Sti: Turbocharge nlp inference at the edge via elastic pipelining. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

Ethan He, Abhinav Khattar, Ryan Prenger, Vijay Korthikanti, Zijie Yan, Tong Liu, Shiqing Fan, Ashwath Aithal, Mohammad Shoeybi, and Bryan Catanzaro. Upcycling large language models into mixture of experts. arXiv preprint arXiv:2410.07524, 2024. URL https://arxiv.org/abs/2410.07524.

Open LLM Data Initiative. Finemath: Mathematical pretraining data for large language models. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/HuggingFaceTB/finemath.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024. URL https://arxiv.org/abs/2401.04088.

Shibo Jie, Yehui Tang, Kai Han, Yitong Li, Duyu Tang, Zhi-Hong Deng, and Yunhe Wang. Mixture of lookup experts. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

Young Jin Kim, Raffy Fahim, and Hany Hassan Awadalla. Mixture of quantized experts (moqe): Complementary effect of low-bit quantization and robustness. arXiv preprint arXiv:2310.02410, 2023. URL https://arxiv.org/abs/2310.02410.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code. arXiv preprint arXiv:2211.15533, 2022. URL https://arxiv.org/abs/2211.15533.

Xiangyu Li, Chengyu Yin, Weijun Wang, Jianyu Wei, Ting Cao, and Yunxin Liu. Vec-lut: Vector table lookup for parallel ultra-low-bit llm inference on edge devices. In *Proceedings of the Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2026.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Zhiwen Mo, Lei Wang, Jianyu Wei, Zhichen Zeng, Shijie Cao, Lingxiao Ma, Naifeng Jing, Ting Cao, Jilong Xue, Fan Yang, and Mao Yang. LUT tensor core: A software-hardware co-design for LUT-based low-bit LLM inference. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2025.

Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. LUT-GEMM: quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh

Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023. URL https://arxiv.org/abs/2307.09288.

Haodong Wang, Qihua Zhou, Zicong Hong, and Song Guo. D2moe: Dual routing and dynamic scheduling for efficient on-device moe-based LLM serving. In *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2025.

Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. T-MAC: CPU renaissance via table lookup for low-bit LLM deployment on edge. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2025.

Xueying Wu, Baijun Zhou, Zhihui Gao, Yuzhe Fu, Qilin Zheng, Yintao He, and Hai Li. Oasis: Outlier-aware lut-based gemm with dual-side quantization for llm inference acceleration. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2026.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. Powerinfer-2: Fast large language model inference on a smartphone. arXiv preprint arXiv:2406.06282, 2024. URL https://arxiv.org/abs/2406.06282.

Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. Edgemoe: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 24:7059–7073, 2025.

Wangsong Yin, Mengwei Xu, Yuanchun Li, and Xuanzhe Liu. Llm as a system service on mobile devices. arXiv preprint arXiv:2403.11805, 2024. URL https://arxiv.org/abs/2403.11805.

Xiangyang Yin, Xingyu Liu, Tianhua Xia, BO BAO, Vithursan Thangarasa, Valavan Manohararajah, Eric Sather, and Sai Qian Zhang. Codequant: Unified clustering and quantization for enhanced outlier smoothing in low-precision mixture-of-experts. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Jixian Zhou, Fang Dong, Ruijun Huang, Hengjie Cao, and Mengyi Chen. Oracle-MoE: Locality-preserving routing in the oracle space for memory-constrained large language model inference. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

A Experimental Details

Implementation and Evaluation Protocol. RcMoE retrieves the longest matching semantic pattern and falls back to the token entry when no pattern is found. We report zero-shot accuracy on six commonsense benchmarks using lm-evaluation-harness and evaluate eight LongBench tasks with an 8,192-token input limit. For each method, we use fixed checkpoints, templates, dataset splits, and aggregation rules, without task-specific tuning. Unless noted otherwise, system measurements use batch size 1, a 16-token prompt, and 64 generated tokens. Storage and transfer sizes use 1024-based conversion, with 1 GB = 1024^3 bytes and 1 MB = 1024^2 bytes. Per-token transfer sizes below 1 GB are reported in MB. RcMoE-360M stores each 960-dimensional SC-LUT row with 1,024-value padding for the tiled kernel, while computation remains 960-dimensional. Tables 5 and 6 summarize the configurations.

Table 5: Additional model configuration details for RcMoE.

Configuration	RcMoE-360M	RcMoE-1.7B
Dense backbone	SmolLM2-360M	SmolLM2-1.7B
SC-LUT storage width	1,024	2,048
Expert dimension	2,560	8,192
Shared experts	1	1
Retrieval experts	7	3
Semantic vocabulary	50K	50K

Table 6: Training and deployment configurations.

Category	Configuration
Training hardware	8× RTX 4090 (24 GB)
Framework	PyTorch 2.5.1, CUDA 12.4
Distributed setup	DeepSpeed ZeRO-2
Optimization	AdamW, peak LR 3×10^{-4}
LR schedule	Cosine decay, 10% warmup
Training schedule	2,000-iteration CPT + 1,000-iteration SFT
Training data	FineWeb-Edu, FineMath-4plus, The Stack Python, UltraChat-200K
Edge device	Jetson Orin Nano 8 GB, 15 W
Storage device	ZHITAI TiPlus 7100 NVMe SSD

Continued Pretraining (CPT). The training entry point reads the CPT mixture as text records from Parquet files and streams them with a sequence length of 512. The dense backbone and shared expert remain frozen, while the retrieval experts, router, expert-input root mean square layer normalization (RMSNorm), and context offsets are trained. CPT uses BF16, AdamW with a learning rate of 3×10^{-4}, a per-device batch size of 4, gradient accumulation of 4, and seed 42. The eight-GPU run uses DeepSpeed ZeRO-2 with CPU optimizer offloading and gradient clipping at 1.0. The script runs for 2,000 dataloader iterations and saves the trainable state every 200 iterations.

Low-Rank Adaptation (LoRA) Fine-Tuning. SFT loads the CPT state through the `--initial-checkpoint` option and runs for 1,000 dataloader iterations in BF16. LoRA uses rank 16, scaling 32, and dropout 0.05, targeting `gate_proj`, `up_proj`, and `down_proj`. The router, expert-input RMSNorm, and context offsets remain trainable, while backbone and expert weights are frozen. AdamW uses a learning rate of 3×10^{-4}, sequence length 512, per-device batch size 4, gradient accumulation 4, and seed 42. UltraChat conversations use role-delimited turns, with the loss applied only to assistant tokens. Both scripts count dataloader iterations, so four accumulation iterations correspond to one optimizer update.

SC-LUT Quantization. Quantization is applied independently to each SC-LUT row and retrieval expert along the hidden dimension. NF4 and INT4 use groups of 512 contiguous values (block/group size 512); NF3 uses the same grouping. Each group stores its own FP16 reconstruction statistics, while the quantized codes are bit-packed. NF4 uses a fixed 16-level normal-float codebook and reconstructs values in the fused retrieval path before router-weighted aggregation.

NPMI Vocabulary Construction. We tokenize UltraChat-200K with the backbone tokenizer and estimate unigram, bigram, and trigram probabilities from corpus counts. For an n-gram $g = (w_1, \ldots, w_n)$, we compute $\mathrm{NPMI}(g) = \log(p(g)/\prod_i p(w_i))\,/[-\log p(g)]$ and clip the score to $[-1, 1]$. Frequent bigrams and trigrams are ranked by NPMI, from which we retain 40K bigrams and 10K trigrams to form the 50K semantic pattern vocabulary. Each retained pattern is assigned a unique context identifier; inference selects the longest matching pattern and uses the default identifier when no pattern is found.

B Retrieval Runtime Implementation

Fused Retrieval Kernel. The CUDA path uses one Triton kernel for router GEMV, softmax, token- and context-row NF4 unpacking, per-group reconstruction, and router-weighted expert aggregation. Token and context entries are decoded directly from packed storage and summed before aggregation, so intermediate expert outputs, router logits, and routing probabilities are not materialized in global memory. The kernel processes the hidden dimension in tiled blocks and accumulates in single-precision floating-point (FP32) before writing the output in the model dtype.

Asynchronous SC-LUT Prefetch. SC-LUT code, mean, and scale files are memory mapped and prefetched by a C++17 thread pool using `madvise` and `posix_fadvise`. The double-buffered expert buffer in the execution pipeline is implemented as two alternating prefetch slots over these mappings. While layer l consumes the token and context rows in slot $l \bmod 2$ and executes its shared expert and fused retrieval kernel, the background workers prefetch the rows required by layer $l + 1$ into the other slot. No additional layer-wide synchronization barrier is introduced. If the next-layer pages are not ready, their first memory-mapped access provides the required handoff; otherwise, storage access is hidden by current-layer computation. Thus, the implementation follows the layer-wise overlap described in the main paper while bounding the active prefetched working set to two layer slots.

C Cross-Platform Deployment

Ascend 310B1 Adaptation and Batch Scaling. To assess portability beyond CUDA platforms, we adapt RcMoE to the Huawei Ascend 310B1. The implementation uses Python 3.9.2, PyTorch 2.1.0, torch-npu 2.1.0, Transformers 4.36.2, and Ascend Runtime 23.0.0. The Ascend path preserves the retrieval semantics of the GPU implementation and supports SC-LUT access through memory mapping, asynchronous loading, and a fused operator for softmax weighting, NF4 reconstruction, and expert aggregation. Both model scales use greedy decoding with a 16-token prompt, 64 generated tokens, and a key-value (KV) cache bucket size of 128. The model and SC-LUT remain resident across requests. Each configuration is warmed up for two iterations and measured for three iterations in an independent process; Table 7 reports the median values. Increasing the batch size from 1 to 8 raises throughput from 2.490 to 9.447 tokens/s for RcMoE-360M and from 0.866 to 5.396 tokens/s for RcMoE-1.7B, while peak memory remains below 814 and 3,491 MB, respectively. The latency coefficient of variation remains below 1.5% in all settings. These results verify that the retrieval pipeline transfers to the Ascend software stack and scales predictably with batch size. They characterize scaling on the same device and are not used to compare speed between Ascend and Jetson. Figure 9 shows the latency and throughput trends.

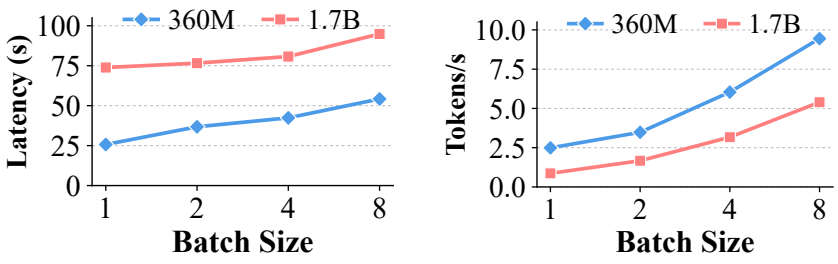

Figure 9: Batch-size scaling of RcMoE on Ascend 310B1: latency (left) and throughput (right).

Table 7: End-to-end batch scaling of RcMoE on Ascend 310B1. Latency and throughput are medians over three measured iterations after two warm-up iterations; CV denotes the coefficient of variation of latency.

Model	Batch	Latency (s)	Throughput (tokens/s)	Memory (MB)	CV (%)
RcMoE-360M	1	25.699	2.490	728.74	0.82
	2	36.786	3.480	740.59	0.95
	4	42.381	6.040	764.28	0.56
	8	54.199	9.447	813.97	0.81
RcMoE-1.7B	1	73.879	0.866	3294.47	1.48
	2	76.624	1.670	3371.74	0.52
	4	80.757	3.170	3376.51	0.53
	8	94.891	5.396	3490.62	0.14

D Extended Model Evaluation

Model Accuracy. Table 8 reports task-level zero-shot accuracy across methods. The results complement the averages in the main paper by showing where each method gains or loses accuracy. In the MoE-598M group, RcMoE exceeds MoLE on five of six tasks and improves ARC-Challenge by 1.45 points. In the MoE-2.9B group, RcMoE improves OpenBookQA and PIQA by 0.80 and 0.60 points, while the remaining differences from MoLE stay within 0.58 points. RcMoE also matches or exceeds the corresponding in-memory MoE average at both scales. The consistent task-level profile distinguishes RcMoE from methods such as EdgeMoE, whose average degradation is distributed across several benchmarks rather than isolated to one task.

Table 8: Per-task zero-shot accuracy across methods. ARC-C and OBQA denote ARC-Challenge and OpenBookQA.

Model	Method	ARC-C	BoolQ	HellaSwag	OBQA	PIQA	WinoGrande	Average
Dense-360M	In-Memory	38.05	61.44	56.34	37.80	71.82	59.12	54.10
MoE-598M	In-Memory	36.68	63.03	56.89	38.90	71.49	58.93	54.32
	EdgeMoE	26.88	55.84	39.32	30.20	60.01	50.99	43.87
	MoQE-INT8	35.92	64.07	56.09	38.60	71.44	58.25	54.06
	D^2MoE	35.32	63.27	56.03	37.60	71.44	57.77	53.57
	MoLE	37.46	63.12	56.57	39.00	71.60	58.25	54.33
	RcMoE	38.91	63.73	56.62	38.40	71.82	58.48	54.66
Dense-1.7B	In-Memory	47.01	72.51	71.36	43.80	76.88	66.22	62.96
MoE-2.9B	In-Memory	46.85	72.35	70.29	45.85	76.39	67.77	63.25
	EdgeMoE	34.73	63.73	54.26	37.60	67.30	53.83	51.91
	MoQE-INT8	45.99	72.26	70.28	45.00	76.22	64.01	62.29
	D^2MoE	45.82	71.99	70.28	44.00	75.73	63.77	61.93
	MoLE	48.21	74.98	71.60	43.80	76.66	65.59	63.47
	RcMoE	47.95	74.40	71.58	44.60	77.26	65.51	63.55

Task-Level LongBench Results. Table 9 extends the main comparison with task-level LongBench results across methods under the same zero-shot protocol and 8,192-token input limit. The suite covers multi-hop question answering (2WikiMQA, HotpotQA, and MuSiQue), long-document question answering (MultiFieldQA-en and Qasper), narrative comprehension (NarrativeQA), query-focused summarization (QMSum), and Chinese reading comprehension (DuReader). At the smaller scale, RcMoE reaches 9.89 on average, compared with 9.74 for the in-memory MoE and 9.86 for MoLE. At the larger scale, RcMoE reaches 11.44, exceeding the in-memory MoE by 0.89 points and showing its largest gains on DuReader, HotpotQA, MultiFieldQA-en, and Qasper. RcMoE improves over the in-memory MoE on seven of the eight tasks, with the largest gain on MultiFieldQA-en, while

remaining competitive across multi-hop question answering, document question answering, narrative comprehension, and summarization. These task-level results provide a more precise view of the aggregate improvement reported in the main paper.

Table 9: Per-task LongBench results across methods. 2W denotes 2WikiMQA; DuR, MultiF., MuSi, Narr., and Qasp. denote DuReader, MultiFieldQA-en, MuSiQue, NarrativeQA, and Qasper, respectively. Avg. is the mean across tasks.

Model	Method	2W	DuR	Hotpot	MultiF.	MuSi	Narr.	Qasp.	QMSum	Avg.
Dense-360M	In-Memory	8.71	11.58	7.54	14.43	3.46	2.70	8.74	19.40	9.56
MoE-598M	In-Memory	8.00	10.95	7.85	15.24	3.85	2.86	9.45	19.76	9.74
	EdgeMoE	2.41	0.85	1.40	7.98	0.77	1.68	3.52	10.96	3.69
	MoQE-INT8	7.76	11.21	7.60	15.66	3.56	2.94	8.86	19.80	9.67
	D^2MoE	7.75	11.71	7.39	15.52	4.17	2.74	9.05	19.31	9.70
	MoLE	6.85	12.16	7.58	17.19	3.22	2.92	8.81	20.17	9.86
	RcMoE	7.32	11.78	8.16	16.52	3.64	3.18	8.66	19.88	9.89
Dense-1.7B	In-Memory	9.77	12.61	7.55	16.77	4.62	3.08	8.68	20.98	10.51
MoE-2.9B	In-Memory	9.99	12.89	8.19	15.50	4.37	3.21	9.41	20.87	10.55
	EdgeMoE	5.44	4.25	4.06	11.08	2.37	1.91	5.35	18.22	6.59
	MoQE-INT8	8.14	13.17	8.02	19.07	4.76	3.56	10.16	20.44	10.91
	D^2MoE	8.63	12.95	8.51	18.06	4.97	3.48	9.90	20.64	10.89
	MoLE	9.04	12.42	7.92	18.46	4.52	2.71	11.11	20.59	10.85
	RcMoE	9.60	14.41	8.91	19.58	4.81	3.42	9.70	21.11	11.44

E Additional Ablation and System Analysis

SC-LUT Precision. Table 10 provides a task-level view of the accuracy changes introduced by SC-LUT quantization. For RcMoE-360M, NF4 remains within 0.32 points of BF16 on every benchmark, indicating that the 0.11-point average decrease is not caused by a pronounced failure on any individual task. The largest reductions occur on WinoGrande and BoolQ, while OpenBookQA slightly improves after NF4 quantization. At the 1.7B scale, the effect is more task dependent: NF4 improves BoolQ and PIQA by 0.88 and 0.44 points, respectively, but decreases OpenBookQA and WinoGrande by 0.60 and 0.74 points. These changes largely offset each other, yielding an average score nearly identical to BF16. Compared with NF4, INT4 shows a more consistent degradation on RcMoE-360M, including a 0.84-point decrease on OpenBookQA, while NF3 exhibits a similar task-level pattern despite its lower storage precision. On RcMoE-1.7B, both INT4 and NF3 remain competitive on average, but incur larger losses than NF4 on ARC-Challenge. Overall, the per-task results show that NF4 does not introduce systematic degradation across benchmarks and provides the most stable accuracy profile across the two model scales.

Table 10: Per-task zero-shot accuracy across SC-LUT precisions. ARC-C and OBQA denote ARC-Challenge and OpenBookQA.

Model	Precision	ARC-C	BoolQ	HellaSwag	OBQA	PIQA	WinoGrande	Average
RcMoE-360M	BF16	39.08	63.98	56.71	38.20	71.87	58.80	54.77
	NF4	38.91	63.73	56.62	38.40	71.82	58.48	54.66
	INT4	38.75	63.89	56.41	37.36	71.74	58.55	54.45
	NF3	39.03	63.89	56.48	37.35	71.82	58.49	54.51
RcMoE-1.7B	BF16	47.87	73.52	71.58	45.20	76.82	66.25	63.54
	NF4	47.95	74.40	71.58	44.60	77.26	65.51	63.55
	INT4	46.59	73.94	71.51	45.00	77.09	66.57	63.45
	NF3	47.01	74.01	71.72	45.20	76.99	66.14	63.51

Table 11 further reports the task-level LongBench results across SC-LUT precisions. For RcMoE-360M, all compressed variants remain close to the BF16 average, with differences within 0.18 points; NF4 has the smallest average difference at 0.08 points. NF4 decreases HotpotQA by 0.10 points while showing moderate variations on the remaining tasks. At the 1.7B scale, NF4 achieves the same average score as BF16 after rounding, despite small task-level fluctuations. INT4 and NF3 exhibit larger variations, particularly on multi-hop QA tasks, whereas NF4 maintains the most stable performance across both model scales. Together with the zero-shot accuracy results, these findings support NF4 as the default SC-LUT precision.

Table 11: Per-task LongBench results across SC-LUT precisions. 2W denotes 2WikiMQA; DuR, MultiF., MuSi, Narr., and Qasp. denote DuReader, MultiFieldQA-en, MuSiQue, NarrativeQA, and Qasper, respectively. Avg. is the mean across tasks.

Model	Precision	2W	DuR	Hotpot	MultiF.	MuSi	Narr.	Qasp.	QMSum	Avg.
RcMoE-360M	BF16	7.60	11.92	8.26	16.40	3.73	3.25	8.63	19.97	9.97
	NF4	7.32	11.78	8.16	16.52	3.64	3.18	8.66	19.88	9.89
	INT4	7.37	11.68	7.88	16.44	3.53	3.13	8.47	19.82	9.79
	NF3	7.39	11.74	8.22	16.31	3.45	3.15	8.53	19.85	9.83
RcMoE-1.7B	BF16	10.21	13.55	9.00	19.66	4.90	3.53	9.84	20.85	11.44
	NF4	9.60	14.41	8.91	19.58	4.81	3.42	9.70	21.11	11.44
	INT4	9.47	13.36	8.68	19.25	3.89	3.59	9.57	20.67	11.06
	NF3	9.97	14.04	8.61	19.50	4.21	3.33	9.55	21.07	11.29

Semantic Pattern Vocabulary Size. Table 12 studies the effect of semantic pattern vocabulary size on RcMoE-360M using the final 1,000-step checkpoint for each setting. The 50K vocabulary achieves the highest LongBench score. Expanding it to 75K slightly improves average accuracy, but reduces the LongBench score and increases storage. Accordingly, we use 50K patterns as the default setting because it provides the best balance between model quality and SC-LUT footprint.

Table 12: Semantic pattern vocabulary size ablation on RcMoE-360M. Model-quality results use BF16 SC-LUTs; the final column reports NF4 storage.

Patterns	Avg. Acc.	Avg. Long.	NF4 (GB)
1K	54.61	9.69	5.02
5K	54.73	9.76	5.42
10K	54.62	9.76	5.92
25K	54.73	9.75	7.43
50K	54.77	9.97	9.93
75K	54.84	9.84	12.43

SC-LUT Construction Cost. SC-LUT construction is a one-time offline procedure over 99,152 retrieval identifiers, including 49,152 token identifiers and 50K semantic patterns. Storage values in this analysis use the RcMoE-1.7B layer and hidden dimensions. As shown in Table 13, both construction time and storage scale nearly linearly with the number of retrieval experts, while NF4 consistently reduces the SC-LUT footprint by about four times. On the RTX 4090, the total time increases from 44.0 seconds for three experts to 499.0 seconds for 32 experts, indicating predictable offline scaling. On the Jetson Orin Nano, NF4 quantization dominates the overhead, accounting for most of the 746.7-second total time.

Table 13: Offline SC-LUT construction overhead with 50K NPMI semantic patterns.

E	Device	BF16 (GB)	NF4 (GB)	Build (s)	Quant. (s)	Total (s)
3	RTX 4090	27.23	6.81	25.4	18.6	44.0
	Jetson Orin Nano	27.23	6.81	91.0	655.7	746.7
8	RTX 4090	72.62	18.16	68.0	49.8	117.8
16	RTX 4090	145.24	36.31	133.7	100.7	234.4
32	RTX 4090	290.48	72.62	280.3	218.7	499.0

SC-LUT Scalability. We evaluate the scalability of retrieval-centric MoE inference with increasing retrieval expert counts (E) on an NVIDIA Jetson Orin Nano with 8 GB unified memory. RcMoE constructs SC-LUT entries for 49,152 token identifiers and 50K NPMI-selected semantic patterns using the RcMoE-1.7B layer and hidden dimensions. As shown in Table 14, the enlarged BF16 SC-LUT rapidly becomes impractical as E increases. NF4 reduces its storage footprint by approximately $4\times$. Although semantic-conditioned retrieval requires both token and context entries, kernel fusion and asynchronous prefetching limit the resulting latency increase. Consequently, RcMoE scales more efficiently than the uncompressed MoLE execution path as the expert pool grows.

Table 14: SC-LUT scalability with 50K NPMI semantic patterns.

E	SC-LUT (GB)		Latency (ms)		Speedup
	BF16	NF4	RcMoE	MoLE	
3	27.23	6.81	4.60	4.61	1.00×
8	72.62	18.16	5.22	5.65	1.08×
16	145.24	36.31	6.08	7.46	1.23×
32	290.48	72.62	7.81	11.06	1.42×
64	580.97	145.24	11.29	18.28	1.62×
128	1161.94	290.48	18.22	32.70	1.79×

Energy Efficiency. We measure energy on a Jetson Orin Nano using the `tegrastats` VDD_IN sensor at 200 ms intervals. Each method uses deterministic decoding (128 generated tokens), FP16 execution, and three trials. MoLE accesses a BF16 token-indexed LUT through memory mapping; RcMoE uses a 50K NPMI-enhanced NF4 SC-LUT with fused retrieval and asynchronous prefetching. As shown in Table 15, RcMoE lowers latency and energy across prompt lengths. For a 96-token prompt, latency falls from 30.5 s to 25.6 s and energy from 365.3 J to 327.8 J (10.3%). The offloaded LUT footprint falls from 13.50 GB for MoLE to 6.81 GB for RcMoE. Fused execution and prefetching hide much of the semantic-matching and dequantization cost, so reduced storage traffic and execution time outweigh the modest increase in average power.

Table 15: Energy comparison between MoLE and RcMoE on Jetson Orin Nano with 128 generated tokens.

Prompt Length	Latency (s)		Energy (J)		Reduction
	MoLE	RcMoE	MoLE	RcMoE	
1	15.8	14.3	185.9	173.0	6.9%
27	18.6	16.9	229.0	213.1	6.9%
79	27.4	23.7	333.6	303.0	9.2%
96	30.5	25.6	365.3	327.8	10.3%