SAEM: Stage-Aware Expert Management for Memory-Efficient MoE Inference in Chain-of-Thought Reasoning

Yujie Zhang, Bin Gao, and Tulika Mitra

***Abstract*—Chain-of-thought (CoT) prompting improves LLM reasoning by decomposing complex problems into intermediate steps, but its sequential nature increases decoding latency and memory usage. Mixture-of-Experts (MoE) models scale capacity through sparse expert activation, yet their full expert weights often exceed GPU memory and require costly GPU–CPU transfers. Existing runtimes treat all tokens uniformly, overlooking a key structural property of CoT traces: consecutive reasoning stages exhibit coherent and predictable expert activation patterns. Ignoring this stage-level regularity leads to inefficient caching and unnecessary data movement. We propose SAEM, a stage-aware MoE inference runtime that detects reasoning stage boundaries and exploits stage-level activation coherence to guide expert placement. SAEM combines stage-aware caching, expert-aligned token repacking, and in-situ CPU execution to reduce data transfer and kernel fragmentation. On mathematical and scientific reasoning workloads, SAEM achieves an average 1.33× throughput improvement over the strongest state-of-the-art caching and offloading baselines under constrained GPU memory, rising to 1.54× when calibration data matches the workload, demonstrating the effectiveness of stage-aware, locality-driven MoE inference for CoT reasoning.**

***Index Terms*—Mixture-of-experts inference, chain-of-thought, GPU–CPU cooperation, stage-aware caching.**

I. Introduction

LARGE language models (LLMs) have demonstrated strong reasoning capabilities through chain-of-thought (CoT) prompting, which decomposes complex problems into structured intermediate steps [1]–[3]. This approach is particularly effective in mathematically demanding domains, where explicit step-by-step reasoning substantially improves accuracy. However, CoT decoding incurs considerable computational cost. The autoregressive generation of long reasoning traces increases both inference latency and memory pressure; on challenging reasoning benchmarks, such traces frequently span hundreds or thousands of tokens, and evaluations of long-output reasoning models often permit generation budgets of up to 32,768 tokens [4]. Consequently, reasoning trace length tends to increase with task difficulty, further amplifying decoding cost and memory demand.

Mixture-of-Experts (MoE) architectures offer a promising direction for scalable inference by activating only a sparse subset of experts per token, enabling large model capacity without proportionally increasing computation [4], [5]. Yet deploying MoE models for CoT reasoning under resource constraints remains challenging. The full set of expert weights typically exceeds available GPU memory, forcing dynamic movement of expert parameters between CPU and GPU. Moreover, even when GPU memory is sufficient to hold all experts, the inherent sparsity of MoE routing means that most GPU-resident experts are seldom activated, leading to significant underutilization of scarce GPU memory and undermining the benefits of full on-device placement.

Quantization and sparsity approaches [6], [7] reduce computational and memory costs but often yield inconsistent performance across task domains. Recent MoE inference runtimes instead mitigate memory limitations through token-level expert caching [8]–[11] and expert prefetching [12]–[15]. Caching systems track expert usage and update the GPU cache based on recency or frequency. Prefetching complements this by predicting future expert requirements and proactively transferring expert weights to GPU memory. These approaches, however, treat generated tokens uniformly and overlook a key structural property of CoT reasoning: consecutive reasoning stages exhibit strong semantic coherence and consistently activate predictable expert subsets. When models explore alternative solution paths, perform self-correction, or validate intermediate results, they transition between distinct reasoning modes, each associated with stable and stage-specific expert activation patterns. Token-level expert management cannot capture these coarse-grained regularities, resulting in unnecessary GPU–CPU data movement.

We introduce SAEM, a data-aware MoE inference runtime for resource-constrained CoT reasoning that leverages semantic structure rather than token-level uniformity. SAEM builds on a key empirical observation: reasoning stages marked by discourse transitions (e.g., "alternatively," "instead,") exhibit coherent expert activation patterns that remain stable within stages but shift predictably across boundaries. This stage-level coherence provides actionable guidance for cache management that token-level approaches cannot exploit.

SAEM employs three coordinated mechanisms to optimize inference efficiency. First, it uses lightweight pattern matching to detect stage transitions in real time and aggregates

This work was supported by the National Research Foundation, Singapore, under its Competitive Research Program Award NRF-CRP23-2019-0003 and the Ministry of Education, Singapore, under Tier 3 grant MOE-MOET32024-0003. Yujie Zhang, Bin Gao, and Tulika Mitra are with the School of Computing, National University of Singapore, Singapore (e-mail: zyujie@comp.nus.edu.sg; bingao@nus.edu.sg; tulika@comp.nus.edu.sg).

A preliminary version of this work was **accepted for publication at the 63rd ACM/IEEE Design Automation Conference (DAC 2026)**, Long Beach, CA, USA; this article extends it with a detailed treatment of reasoning stage boundary detection, evaluation on AIME 2024 and GPQA-Diamond, and a prediction-oracle upper-bound analysis.

expert usage statistics at stage granularity to guide cache management. This coarse-grained strategy updates GPU expert residency only when reasoning semantics change, substantially reducing data movement compared to fine-grained token-level approaches while maintaining a high cache hit rate. Second, SAEM reorganizes tokens by expert assignment through expert-aligned token repacking, converting scattered memory accesses into contiguous batches that improve GPU utilization and reduce kernel launch overhead, particularly under highly skewed expert routing. Third, SAEM executes infrequently activated experts directly on the CPU via in-situ execution, avoiding redundant PCIe transfers and preventing cache pollution from experts unlikely to be reused within the current reasoning stage. These three mechanisms address complementary aspects of the inference challenge and thus form an integrated system rather than a set of independent optimizations.

Our main contributions are as follows:

- We identify key empirical characteristics of expert activation during CoT reasoning in MoE inference on complex reasoning tasks, and show how these insights can guide runtime hardware resource allocation.
- We present SAEM, a stage-aware inference runtime that reduces memory overhead through coordinated expert caching, token reorganization, and selective CPU execution while maintaining high throughput under constrained GPU memory.
- Through extensive evaluation across two MoE models, three reasoning benchmarks, and varying batch sizes and GPU cache budgets, we demonstrate that SAEM consistently outperforms state-of-the-art fine-grained expert caching and offloading baselines, with an average 1.33× throughput improvement overall and 1.54× under calibration-matched conditions.

II. Background

A. Mixture-of-Experts Inference

1) Mixture-of-Experts Inference: MoE models replace dense feed-forward layers with sparsely activated expert networks. A routing mechanism selects only a small subset of experts (e.g., top-k) per token, enabling substantial parameter scaling without proportional growth in computation. This sparse activation mechanism allows parameter growth far beyond what dense models can practically support, while keeping per-token computation relatively stable. At inference time, however, expert activations are often highly skewed and strongly dependent on the input domain, despite load-balancing regularization during training. This skew amplifies the need for effective expert placement and scheduling.

2) Resource-Efficient MoE Inference: To address these challenges, prior systems employ various dynamic expert management strategies, including GPU-side caching [8]–[11], CPU-side execution of non-resident experts [16], and sequence-level expert prediction and prefetching [12], [13], [17]. These techniques reduce transfer overhead and improve memory utilization, but they generally operate at token-level, calibration-based, or sequence-level granularity and thus overlook the semantic structure of generated outputs. In multi-step reasoning tasks, CoT traces exhibit distinct reasoning stages with stable, predictable expert activation patterns—regularities that finer-grained management fails to exploit.

B. Chain-of-Thought Reasoning

1) CoT as Structured Multi-Step Reasoning: Chain-of-thought (CoT) reasoning enables step-by-step "thinking" by decomposing complex problems into intermediate subgoals and deductions [5]. Rather than producing answers directly, models articulate their reasoning process, often involving self-correction, backtracking, and exploration of alternatives. This structured approach is widely adopted in advanced LLMs such as OpenAI o1 [5] and DeepSeek-R1 [4] to improve accuracy and robustness on mathematical and logical reasoning tasks.

2) Internal Structure of CoT Traces: CoT traces are not linear streams of tokens but structured sequences composed of distinct reasoning stages. These stages are typically signaled by discourse-level transition cues such as "alternatively," "on second thought," or "therefore," which indicate shifts in reasoning strategy, verification, or conclusion (Fig. 1). These cues segment the reasoning process into semantically coherent units. Crucially, different stages—such as exploring alternatives, validating assumptions, or finalizing conclusions—exhibit distinct computational patterns and activate different expert subsets within the model (Fig. 2). This reveals exploitable structural and temporal regularities in expert usage that current MoE inference systems do not leverage.

III. Motivation

The Opportunity: Conventional inference systems treat CoT decoding as a uniform token stream and manage expert placement at token granularity, overlooking semantic regularities across reasoning stages. As shown in Fig. 2, different stages (exploring alternatives, validating assumptions, or finalizing conclusions) activate distinct expert subsets. Recognizing these stage transitions enables more efficient expert utilization in MoE architectures. This section presents empirical observations that motivate our design for resource-constrained MoE inference in CoT reasoning.

Motivation 1: Sparse and Skewed Expert Activation in CoT Reasoning Enables Targeted Caching. Although MoE models are trained with load-balancing objectives to encourage uniform expert utilization [18], domain-specific reasoning tasks exhibit highly skewed routing behavior, resulting in sparse and imbalanced expert usage. Fig. 3 shows layer-wise expert activation patterns of Qwen3-30B-A3B (Qwen3) on MATH-500, averaged over 100 samples. Under a top-8 gating policy and assuming uniform routing across 128 experts, each expert would be expected to be selected with probability ∼6.25%. In practice, however, only a small subset of experts consistently exceeds this baseline, while the vast majority are rarely or never selected. This pronounced skew reveals a strong expert-selection bias: during CoT reasoning, the model repeatedly routes tokens to a narrow set of experts aligned with semantic or task-specific subspaces. Such sparsity and concentration create a clear opportunity for runtime optimization. By caching frequently activated experts and deprioritizing

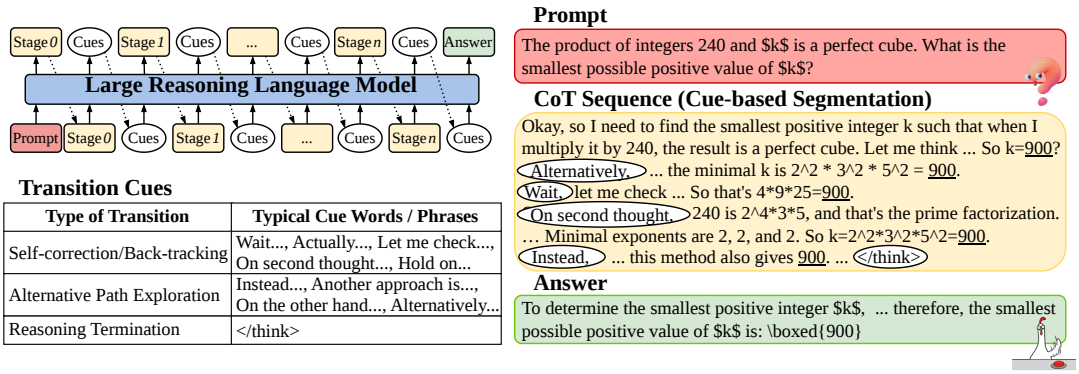

Fig. 1. CoT reasoning employs transition cues to structure multi-step problem solving, facilitating self-correction and the exploration of alternative solution paths.

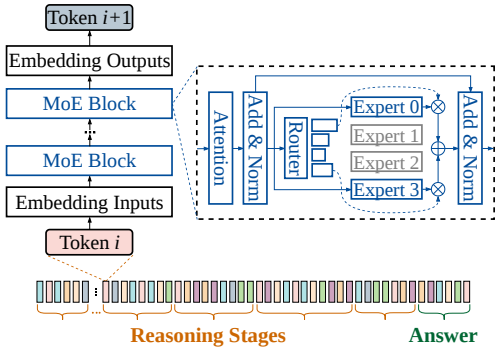

Fig. 2. Illustration of CoT reasoning in a MoE model, where each token activates its top-2 selected experts.

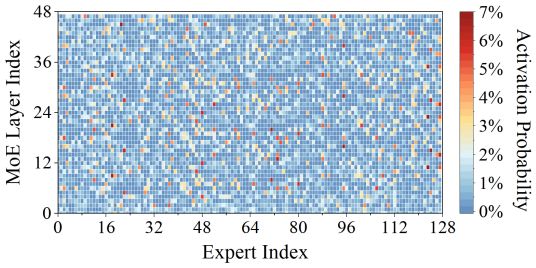

Fig. 3. Layer-wise expert activation pattern in Qwen3 on MATH-500 under CoT reasoning, averaged over 100 samples.

inactive ones, the inference system can reduce memory traffic, avoid unnecessary expert loading, and thereby improve overall computational efficiency.

Motivation 2: Temporal Coherence Across Reasoning Stages Enables Predictive Scheduling. Expert routing remains remarkably stable across adjacent CoT stages, with transitions indicated by linguistic or semantic cues that signal shifts in reasoning strategy, such as exploring alternatives (e.g., "Alternatively,", "Instead,").

To formalize this observation, let $R^p \in \mathbb{R}^{L\times E}$ denote the expert activation matrix for stage p, where L is the number of MoE layers and E the number of experts per layer. Each entry $R^p_{i,j}$ represents the activation probability of expert j in layer i, computed as the ratio of tokens routed to that expert relative to all tokens processed by layer i. To quantify stage-to-stage consistency, we compute the average layer-wise cosine similarity between the activation matrices of consecutive stages: $sim\left(R^p, R^{p+1}\right) = \frac{1}{L}\sum_{l=1}^{L}\cos\left(R^p_l, R^{p+1}_l\right)$.

TABLE I
TEMPORAL COHERENCE SCORES (TC_{seq}) OF EXPERT ACTIVATION PATTERNS ACROSS REASONING STAGES, SEGMENTED BY TRANSITION CUES, MEASURED ON QWEN3 AND ERNIE-4.5.

	MATH-500	AIME 2024	GPQA-Diamond
Qwen3	89.78%	89.50%	92.01%
ERNIE-4.5	89.35%	89.87%	85.27%

Aggregating over a multi-stage reasoning trace of length P yields the sequence-level temporal coherence metric:

$$TC_{seq} = \frac{1}{P-1}\sum_{p=1}^{P-1} sim\left(R^p, R^{p+1}\right), \quad (1)$$

where higher values indicate more stable routing across stage transitions. Table I reports TC_{seq} scores for Qwen3 and ERNIE-4.5 across three datasets, with an average of 89.30%.

This coherence holds despite substantial variation in stage structure. Across Qwen3-generated reasoning traces on MATH-500, the number of stages ranges from 1 to 49 (median 5, average 8), while stage length ranges from 55 to 7129 tokens (median 271, average 484). This suggests that SAEM benefits from local expert-activation regularity between adjacent stages rather than from uniformly short or homogeneous traces. Accordingly, stage-aware scheduling can use the activation pattern of stage p to proactively place experts for stage $p+1$, reducing cache updates and expert transfer overhead. Although scalability to substantially longer contexts remains future work, the same principle should apply as long as adjacent-stage coherence persists.

Motivation 3: Sequential Execution in Resource-Constrained MoEs Amplifies Kernel Launch Overhead. Limited GPU memory often forces MoE layers to execute experts nearly sequentially rather than in parallel, amplifying the inefficiencies of the naive expert-centric gather–compute–scatter process—referred to here as

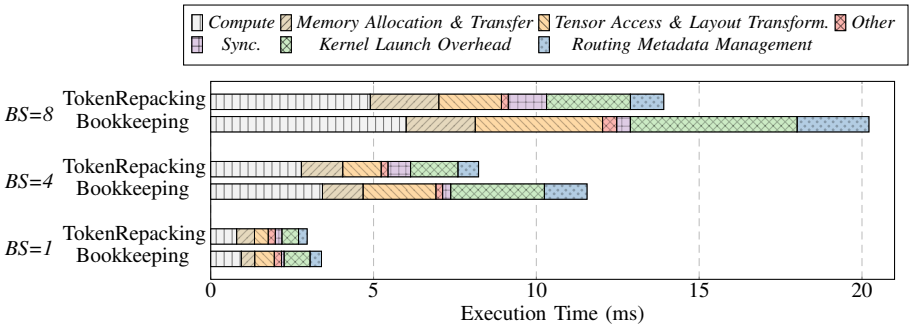

Fig. 4. Operation-level breakdown of *Bookkeeping* and *Token Repacking* strategies in one MoE layer during decoding, isolating execution overhead (excluding routing) on Qwen3.

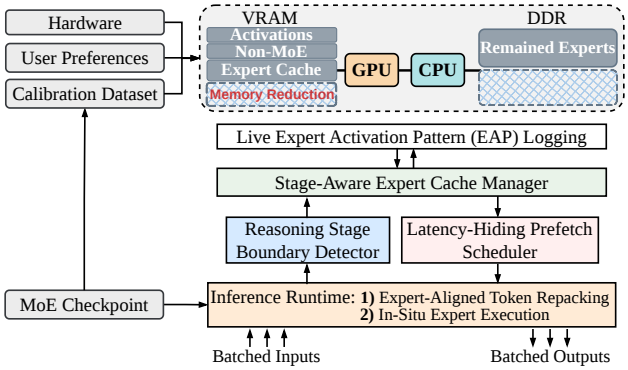

Fig. 5. Design Overview of SAEM.

Bookkeeping. By constructing a global routing mask and identifying token subsets for each expert, this procedure introduces highly scattered token access operations. The problem is exacerbated in reasoning-optimized architectures. For example, Qwen3 employs top-8 routing (versus the typical top-2 in standard MoEs), which quadruples the number of kernel invocations devoted to scattered token-access operations. Consequently, a single forward pass through one MoE layer requires more than 128 kernel launches per token solely for expert execution, with scattered token access kernels accounting for 18.75% of these calls.

Fig. 4 compares *Bookkeeping* and *TokenRepacking* in a single MoE layer across batch sizes. Under *Bookkeeping*, kernel launch overhead, routing metadata management, and token access/layout transformation collectively account for 54.2% of total execution time on average. With *TokenRepacking*, this fraction drops to 40.0%, driven by fewer token access/layout transformation kernels (–4.7%) and lower kernel launch overhead (–6.7%). These inefficiencies motivate *TokenRepacking*: grouping tokens by expert assignment enables batched expert execution, reduces scattered token access kernels, and improves efficiency in resource-constrained CoT reasoning.

IV. SAEM: System Design

A. Overview

We now present SAEM, a data-aware MoE inference runtime that exploits stage-level regularities in CoT reasoning. Fig. 5 shows its overall architecture and data flow. SAEM comprises three core components: (i) a reasoning-stage boundary detector that identifies semantic transitions in CoT traces; (ii) a stage-aware expert cache manager that determines GPU–CPU expert placement based on historical activation patterns; and (iii) a latency-hiding prefetch scheduler that overlaps data movement with computation. In addition, SAEM incorporates two always-on optimizations: expert-aligned token repacking to improve memory coalescing and GPU utilization, and in-situ expert execution to avoid unnecessary weight transfers by executing CPU-resident experts directly on the host.

B. Reasoning Stage Boundary Detection

1) Lightweight Pattern Matching for Stage Transitions:

a) Customized Transition Pattern Extraction: SAEM identifies stage boundaries from the generated token stream using lightweight lexical pattern matching. CoT traces are often organized into semantically coherent stages, such as initial derivation, alternative path exploration, self-correction, verification, and finalization. In this work, we focus on alternative path exploration, while extending detection to broader reasoning strategies is left for future work. These stage transitions are frequently indicated by discourse-level cues, such as "Alternatively," "Instead," "Another way is," and "On second thought," which provide a low-cost signal of shifts in the model's reasoning strategy.

SAEM avoids heavyweight semantic parsing or additional neural boundary classifiers, as they would introduce extra computation on the critical decoding path. Instead, cue-based detection only maintains a compact set of transition patterns and matches them against recently generated tokens. This design is sufficient for SAEM because the detector is not intended to produce perfect human-interpretable segmentation; rather, it only needs to identify coarse transition points where expert activation patterns are likely to change, enabling expert placement updates at stage granularity.

To build the transition patterns, SAEM first collects a global transition set $\mathcal{T}_{\text{global}}$ from common CoT traces. Directly using all markers in this set may increase matching overhead and introduce false positives. Moreover, different reasoning LLMs exhibit distinct stylistic tendencies shaped by their training and post-training pipelines. For example, Qwen3 tends to use more formal transition phrases, whereas ERNIE-4.5 often

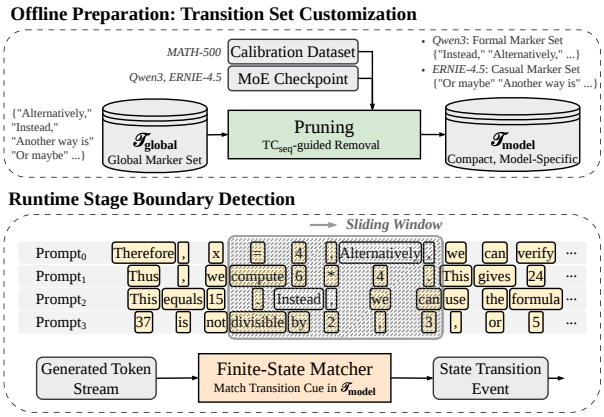

Fig. 6. Lightweight pattern matching for stage transitions.

produces more conversational cues. Therefore, as illustrated in Fig. 6, SAEM customizes the transition set for each target MoE model. Given a calibration dataset and model checkpoint, SAEM evaluates candidate markers according to their contribution to stage-level expert activation coherence, measured by the temporal coherence metric TC_{seq} introduced in Section III. Markers that rarely appear, produce noisy segmentation, or do not improve adjacent-stage coherence are iteratively removed. This offline pruning yields a compact model-specific set $\mathcal{T}_{\text{model}}$, reducing runtime overhead while retaining high-impact cues associated with expert-activation shifts.

b) Sliding-Window Pattern Matching: At runtime, SAEM maintains a configurable sliding window over recently generated tokens for each active sequence. After each decoding step, the detector updates the window and checks whether its suffix matches any cue in $\mathcal{T}_{\text{model}}$. The matcher is implemented as a finite-state machine to track partial matches across token boundaries, which is necessary because transition cues may span multiple subword tokens. Once a complete cue is matched, SAEM emits a stage transition event, finalizes the routing statistics of the completed stage, and starts collecting a new activation profile for the next stage. Thus, the detector provides a lightweight runtime trigger that links semantic shifts in CoT reasoning to coarse-grained expert management.

2) Activation Pattern Aggregation at Boundaries: When a stage transition event is triggered, SAEM aggregates expert activation statistics for the completed stage across all queries in the batch. Specifically, it computes per-layer, per-expert usage frequencies from the observed routing decisions, summarizing the stage's overall computational demand rather than reacting to individual token-level fluctuations.

SAEM then uses the aggregated activation pattern to guide expert placement for the upcoming stage. Based on the temporal coherence described in Section III, frequently activated experts in the completed stage are likely to remain useful in the adjacent stage. SAEM therefore prioritizes these experts for GPU residency and deprioritizes rarely activated ones. By updating the cache only at detected stage boundaries, SAEM avoids the overhead of per-token expert migration while preserving adaptivity to structured changes in the reasoning trajectory.

C. Stage-Aware Expert Cache Management

1) Memory Initialization and Allocation: At system startup, non-MoE layers, including attention mechanisms and layer normalization, are placed directly on the GPU due to their small footprint and consistent activation across all tokens. For MoE layers, SAEM allocates a fixed-size GPU memory region for expert weights, with the cache capacity determined by the user-specified Expert Cache Ratio (ECR). This fixed allocation ensures predictable memory usage throughout inference and prevents memory fragmentation that could arise from dynamic allocation. Optionally, the system can warm the cache with a default hot set of experts identified through dataset-level calibration on representative reasoning tasks. The ECR-defined cache budget is evenly distributed across layers and filled first with the highest-priority experts; any remaining capacity is assigned to globally frequent experts. Once finalized, the per-layer cache slot layout remains fixed. This initialization provides a strong starting point, but expert placement is continuously refined at runtime as stage-specific activation patterns emerge during actual inference.

2) Dynamic Expert Placement Using Historical Patterns: As inference progresses, SAEM maintains two key data structures: a live activation log that tracks the cumulative activation frequency of each expert within the current reasoning stage, and a residency map indicating which experts occupy the fixed per-layer GPU cache slots. At each stage boundary, the residency planner updates expert placement within each layer's allocated cache budget. Experts in a given layer are ranked by their activation frequency in the just-completed stage, and the least-used experts in that layer's cache slots are evicted to make room for higher-priority experts currently residing on the CPU. Only replacements compatible with the fixed per-layer slot layout are permitted, preserving the memory organization defined at initialization. After the update, the activation log is reset to begin collecting statistics for the next stage. This layer-local, stage-aware placement strategy leverages the observed temporal coherence: consecutive reasoning stages exhibit over 90% overlap in activated expert subsets, allowing SAEM to maintain an efficient and predictive caching policy.

3) Latency-Hiding Prefetch and Overlap Optimization: SAEM reduces expert weight transfer latency by overlapping data movement with computation. Expert weight transfers between CPU and GPU are orchestrated using asynchronous DMA operations that overlap with ongoing kernel execution. Lightweight GPU event mechanisms coordinate these asynchronous operations by signaling transfer completion, ensuring that the required expert weights become available exactly when needed without stalling the pipeline. Upon detecting a stage boundary, the scheduler immediately issues prefetch requests, allowing weight transfers to proceed in parallel with the final decoding steps of the current stage and the initial computations of the next stage. This prefetch-and-compute overlap minimizes cache update costs and helps maintain high GPU utilization throughout inference. Crucially, SAEM performs expert migration exclusively at stage boundaries rather than at token granularity, reducing scheduling overhead and aligning data movement with natural transitions in expert

activation patterns.

D. In-Situ CPU Expert Execution

1) Avoiding Unnecessary Data Movement via Local Execution: A key insight in SAEM is that not every expert must reside on the GPU. When a requested expert is not GPU-resident but remains available in host memory, SAEM compares two execution paths: transferring the expert to the GPU for execution or executing it directly on the CPU. It then selects the lower-latency option for that expert. CPU execution is often preferable when only a few tokens are assigned to the expert, as PCIe transfer latency dominates in this regime. As the expert-specific token count increases, however, GPU execution becomes more favorable because the transfer cost is amortized over more tokens. In this way, CPU execution avoids the true bottleneck—PCIe transfer latency—rather than introducing one. Moreover, migrating infrequently activated experts to the GPU would unnecessarily pollute the cache and risk evicting more valuable, frequently used experts.

2) Hybrid CPU–GPU Execution Pipeline: The in-situ execution workflow proceeds as follows. After the routing mechanism assigns tokens to experts under the top-k gating policy, SAEM partitions the token batch into GPU-bound and CPU-bound groups according to the current expert residency map. GPU-resident experts are executed using high-throughput GPU kernels that exploit tensor-core acceleration, while CPU-resident experts are processed in parallel using optimized GEMM backends such as Intel MKL or OpenBLAS.

To reduce overheads associated with remote memory access and scheduling interference, SAEM supports NUMA-aware CPU execution by co-locating expert computation threads and their associated data within the same CPU socket. CPU-resident experts are executed by threads pinned to cores on a single socket, and expert weights and activation buffers are allocated on the corresponding NUMA node. This locality-aware execution model minimizes cross-socket memory traffic and reduces access latency during hybrid GPU–CPU inference.

The outputs from GPU and CPU execution paths are merged through a lightweight synchronization step before advancing to the next layer. This hybrid strategy offers two key benefits: it avoids unnecessary transfers for infrequently activated experts that would otherwise pollute the GPU cache, and it improves overall throughput by using CPU compute resources alongside the GPU—transforming the CPU from a passive storage device into an active computation engine.

E. Expert-Aligned Token Repacking

1) Expert-Aligned Token Repacking: Motivated by the inefficiencies identified in Section III, SAEM introduces a token repacking mechanism that reorganizes tokens by expert assignment in each MoE layer. By grouping tokens by expert IDs and placing them contiguously in memory, repacking removes the scattered token access operations inherent in the naive *Bookkeeping* approach, which repeatedly gathers non-contiguous tokens and launches fragmented kernels. The temporary workspace required for repacking is allocated once and reused across all MoE layers, and the cost of grouping tokens by expert is negligible because it involves only lightweight reindexing over the current batch, resulting in minimal overhead even for deep models.

This alignment enables each expert to execute its computation in a single batched kernel launch rather than multiple small invocations, directly reducing kernel launch overhead and layout transformation costs (Fig. 4). Fig. 7 illustrates the mechanism. Consider four tokens with top-2 routing: x_0 is routed to experts E_0 and E_2, x_1 to E_1 and E_2, x_2 to E_1 and E_3, and x_3 to E_0 and E_2. Without packing, token–expert pairs are scattered across memory, requiring each expert to extract its tokens via separate layout transformation kernels followed by multiple fragmented launches. Token repacking removes this fragmentation by placing all tokens assigned to a given expert contiguously—for instance, x_0 and x_3 for E_0—allowing the expert to operate on a single, dense input buffer.

2) Applicability and Performance Impact: Token repacking is applied at every MoE layer and operates independently of CoT stage boundaries. By converting sparse token-to-expert mappings into dense expert-aligned batches, it simplifies memory access, reduces layout transformation overhead, and decreases kernel launch frequency—yielding higher GPU throughput, particularly at small batch sizes or imbalanced routing.

V. Experimental Evaluation

A. Experimental Setup

1) Models and Datasets: We evaluate SAEM on two reasoning-oriented MoE models: Qwen3-30B-A3B [19] (Qwen3) and ERNIE-4.5-21B-A3B-Thinking [20] (ERNIE-4.5). Their model configurations, including expert count and routing strategy, are summarized in Table II. We use MATH-500 [21] as the primary benchmark, which contains 500 Olympiad-style mathematics problems spanning seven domains. To assess generality beyond this setting, we also evaluate on AIME 2024 [22], a compact but challenging set of 30 competition-level mathematics problems covering topics such as algebra, geometry, number theory, combinatorics, and probability, and GPQA-Diamond [23], a 198-question graduate-level multiple-choice benchmark covering physics, chemistry, and biology. Together, these datasets evaluate SAEM across mathematical reasoning and scientific question answering, allowing us to assess both performance and temporal coherence of expert activations across diverse knowledge domains.

2) Hardware: Experiments are conducted on a single-node setup equipped with a single NVIDIA A100 GPU (80 GB HBM2e, 1.93 TB/s). We use an Intel Xeon Gold 6326 CPU (16 cores, 2.9 GHz) to emulate practical deployment scenarios outside of data centers. The host system provides 512 GB DDR4 memory, with CPU–GPU communication over PCIe 4.0×16 (64 GB/s). To ensure stable and reproducible performance measurements, CPU-based expert execution is confined to a single CPU socket, with threads pinned to cores within that socket and memory allocations restricted to the corresponding NUMA node. This configuration avoids cross-socket memory accesses and minimizes variability from OS

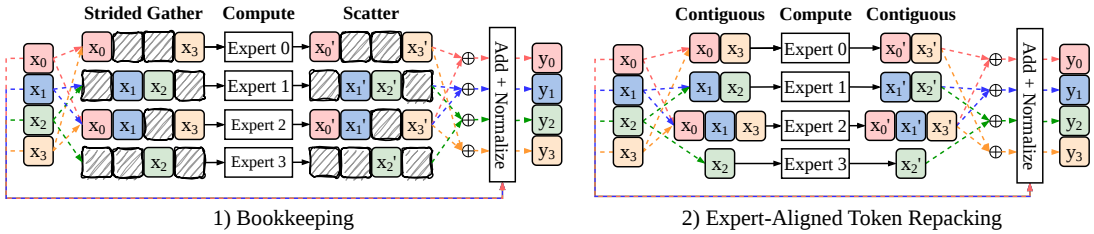

Fig. 7. Illustration of expert-aligned token repacking for a single MoE layer during decoding with a batch size of four. For clarity, the routing decision is assumed but not shown.

TABLE II
ARCHITECTURAL DETAILS OF MOE MODELS OPTIMIZED FOR MULTI-STEP REASONING TASKS.

Model	#Blocks	#Shared Experts	#Routed Experts	Top-k	Experts Params.	Total Params.
Qwen3-30B-A3B (Qwen3)	48	0	128	8	29.0B	30.5B
ERNIE-4.5-21B-A3B (ERNIE-4.5)	28	2	64	6	21.0B	21.8B

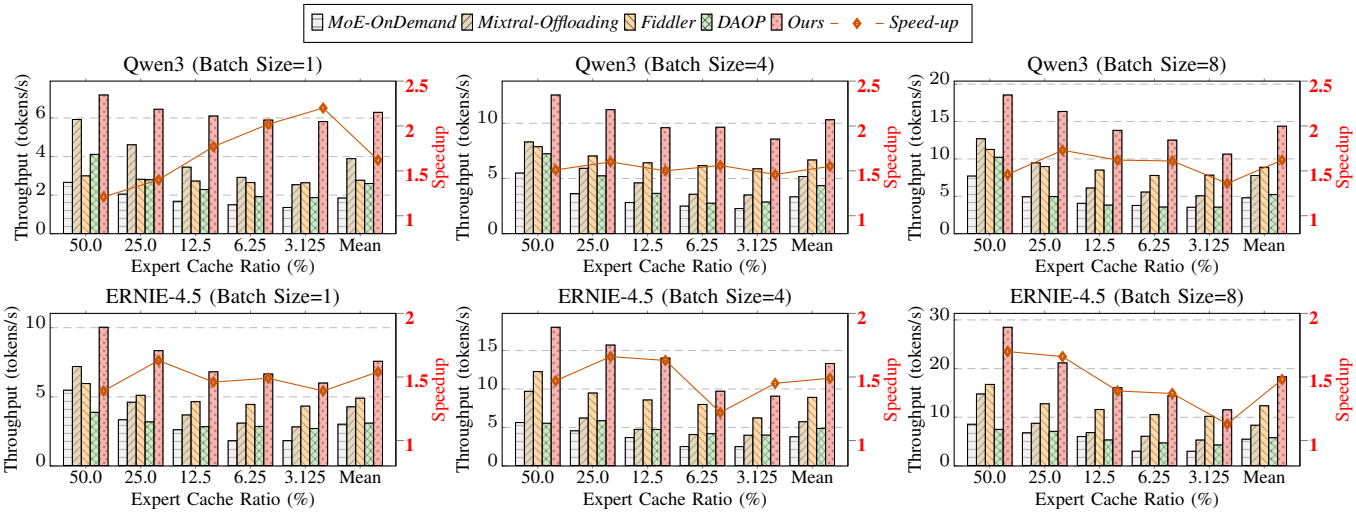

Fig. 8. Throughput comparison over batch sizes and ECRs for **MATH-500**. Overlaid lines denote speedup relative to the strongest baseline.

scheduling and remote NUMA traffic, enabling isolation of SAEM's algorithmic behavior from hardware-induced noise.

3) Implementation: We implement SAEM atop the Hugging Face Transformers library [24] using its PyTorch backend. We vary the batch size from 1 to 8. To evaluate the trade-off between memory and performance, we adopt the **Expert Cache Ratio (ECR)** metric [17], defined as the ratio of GPU-resident experts to the total number of routed experts. Expert caches are initialized using statistics of dominant experts gathered from MATH-500. Inference efficiency is measured by end-to-end throughput, reported in tokens per second.

4) Baselines: We compare SAEM with several representative MoE inference baselines: MoE-OnDemand, Mixtral-Offloading [8], Fiddler [16], and DAOP [17]. MoE-OnDemand keeps non-MoE layers (e.g., attention and normalization) and dominant experts on the GPU, while storing others in CPU memory. Non-resident experts are fetched on demand, often incurring substantial data transfer overhead. Mixtral-Offloading employs an LRU-style token-level cache, dynamically migrating frequently used experts between CPU and GPU. Fiddler reduces data movement by executing non-resident experts directly on the CPU whenever they are activated. DAOP partitions experts between CPU and GPU based on per-sequence activation patterns and predicts future expert usage to precompute selected experts on the CPU.

B. Speedup

Fig. 8, Fig. 9, and Fig. 10 report end-to-end throughput of SAEM against state-of-the-art baselines across varying batch sizes and ECRs on MATH-500, AIME 2024, and GPQA-Diamond, respectively. SAEM outperforms the strongest baseline in nearly every configuration tested, with average speedups of **1.60**× (Qwen3) and **1.47**× (ERNIE-4.5) on MATH-500, **1.20**× and **1.21**× on AIME 2024, and **1.14**× and **1.34**× on GPQA-Diamond.

The margin is largest for MATH-500, the dataset used to calibrate both components of SAEM's offline preparation: the dominant-expert statistics that warm the initial cache and the pruned transition set $\mathcal{T}_{\text{model}}$. AIME 2024 and GPQA-Diamond are therefore held out, and their results characterize SAEM under calibration mismatch rather than under matched conditions. GPQA-Diamond additionally shifts domain, from mathematical derivation to graduate-level multiple-choice science, so its discourse-cue distribution differs from the distribution used to

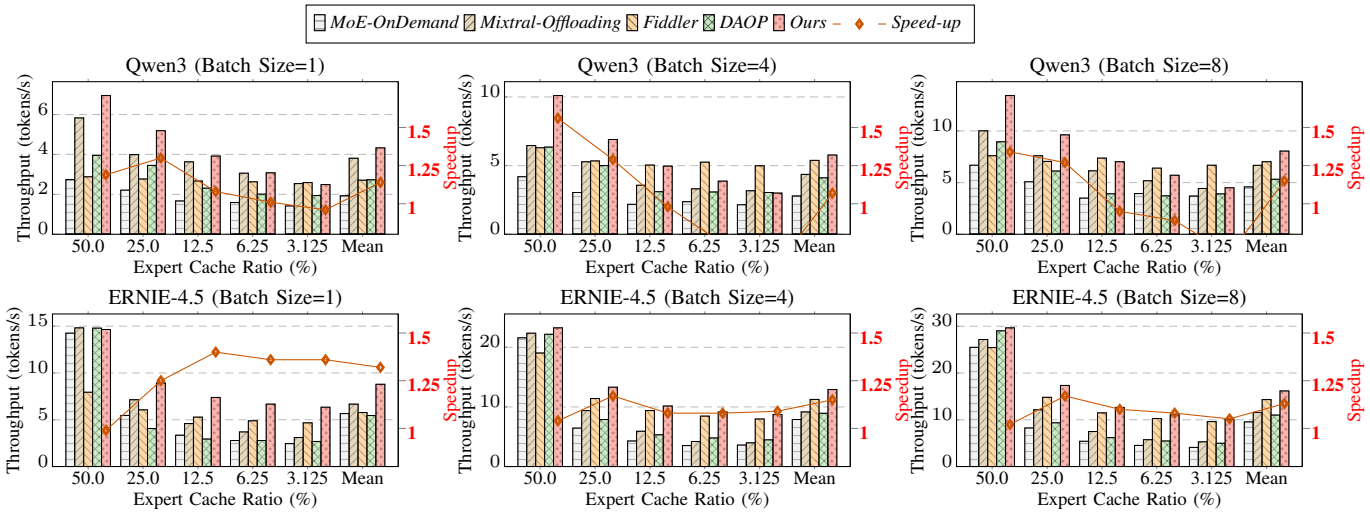

Fig. 9. Throughput comparison over batch sizes and ECRs for **AIME 2024**. Overlaid lines denote speedup relative to the strongest baseline.

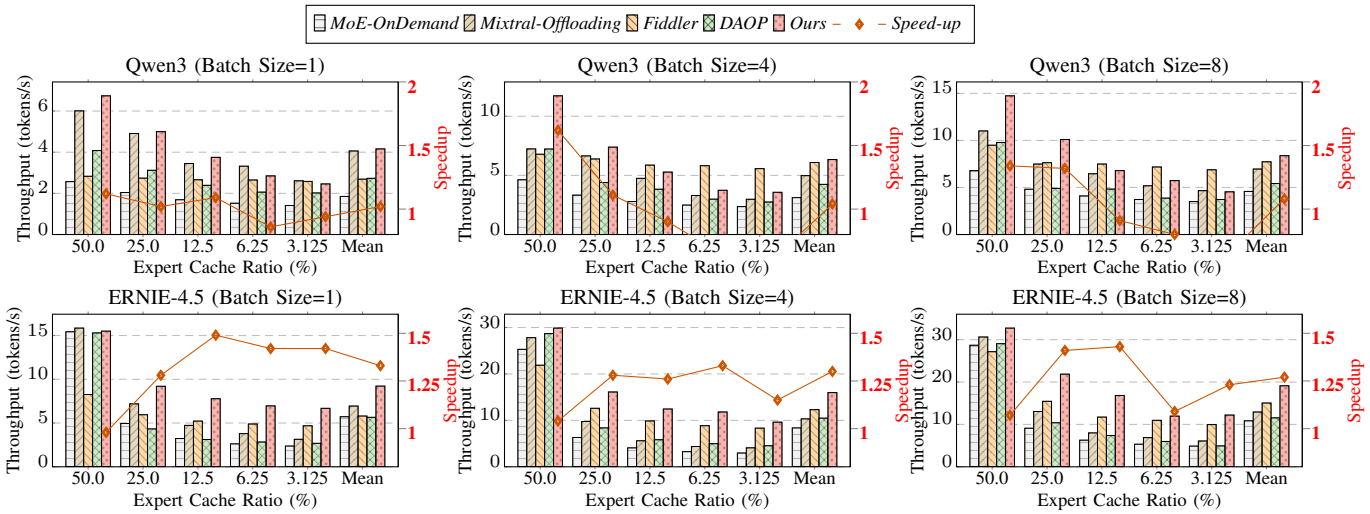

Fig. 10. Throughput comparison over batch sizes and ECRs for **GPQA-Diamond**. Overlaid lines denote speedup relative to the strongest baseline.

prune $\mathcal{T}_{\text{model}}$. That SAEM retains a 1.14–1.34× advantage with no target-domain calibration indicates the stage-locality mechanism transfers across reasoning domains, while the MATH-500 margin indicates the additional headroom available when calibration data is representative of the workload. In the remainder of this section, we examine MATH-500 in detail, as it spans the widest range of cache-pressure conditions and isolates SAEM's behavior when calibration is representative; unless otherwise noted, the following analysis refers to Fig. 8.

1) Single-batch Regime: In the single-batch setting, aggressive LRU-driven token-level expert migration allows Mixtral-Offloading to maintain high GPU utilization by frequently refreshing the active expert set. In contrast, SAEM updates its cache only at CoT boundaries, resulting in a slightly lower steady-state cache hit rate (e.g., 90.08% vs. 94.53% under 50% ECR on Qwen3). However, SAEM offsets this disadvantage with expert-aligned token repacking, which recovers tensor-core efficiency lost to irregular expert placement. As a result, SAEM still achieves a **1.62**× throughput improvement over Mixtral-Offloading despite less frequent cache updates.

2) Concurrent Multi-batch Regime: Under multi-batch execution, the performance gap widens further in favor of SAEM. Fine-grained LRU migration incurs substantial redundant weight transfers under multi-batch execution, an overhead that grows rapidly as batch size increases and cache capacity becomes constrained. SAEM avoids these inefficiencies through stage-aware expert caching, which selectively retains experts aligned with the CoT reasoning structure. This strategy consistently yields higher cache hit ratios and improved GPU utilization. For instance, on Qwen3 with batch size 8, SAEM increases the cache hit ratio relative to Mixtral-Offloading by 53.06% at 12.5% ECR and by **187.95%** at 3.125% ECR. These improvements translate into tangible runtime benefits: SAEM achieves a **2.10**× throughput gain over Mixtral-Offloading at batch size 8 and 3.125% ECR. Importantly, SAEM sustains its advantage even under low-ECR conditions, where the active expert set is small and cache thrashing severely limits LRU-based methods.

3) Comparison with DAOP and Fiddler: These improvements enable SAEM to outperform DAOP and Fiddler by

2.70× and 1.60× on average across all tested models and batch sizes on MATH-500. DAOP uses prediction-based expert precomputation to accelerate CPU-side expert execution and reduce CPU–GPU synchronization overhead, while Fiddler executes non-resident experts on the CPU whenever they are needed. Both approaches construct GPU caches using sentence-level statistics or calibration-dataset statistics to identify dominant experts. However, neither method leverages semantic continuity across CoT reasoning stages. Their expert-selection mechanisms are static at the sentence level or calibration-dataset level and thus cannot adapt dynamically as reasoning unfolds. Consequently, both DAOP and Fiddler often fail to retain experts that are likely to be reused across multi-step reasoning, leading to weak cache reuse and significantly lower throughput compared with SAEM.

C. Prediction-Oracle Upper-Bound Analysis

SAEM performs prediction-guided cache updates: at each detected stage boundary, it uses the expert-activation pattern (EAP) of the completed stage as a proxy for the upcoming stage. To isolate the performance headroom associated with this one-stage prediction lag, we construct a clairvoyant oracle that replaces the predictive proxy with perfect knowledge of the upcoming stage while preserving the cache capacity, update events, and cache-enforcement mechanism.

Let EAP_s denote the per-layer expert activation counts observed during stage s. We first execute the model normally and record the generated token sequence, stage-boundary events, and EAP_s for every stage. We then replay the recorded sequence under two policies. The online predictive policy updates the cache for stage s using EAP_{s-1}, whereas the oracle uses the recorded EAP_s. Both policies enforce the same ECR and invoke the same cache-update and expert-execution mechanisms at identical boundary events; they differ only in the activation profile supplied to the residency planner. Because expert routing is logically independent of physical cache placement, the recorded trace can be reused across the evaluated ECRs.

Fixed-sequence replay is necessary for a controlled comparison. Atomic accumulation in the MoE combine operation can introduce small run-to-run numerical differences, causing unconstrained autoregressive generation to diverge in its output tokens and, consequently, its routing and stage boundaries. Replaying the recorded token sequence fixes the prompts, decoding length, and boundary events across the online predictive and oracle executions. We additionally verify that replayed cue events match the recorded events and disable profiling instrumentation during timing. This protocol isolates the effect of prediction quality while holding cache capacity and update cadence fixed.

We quantify the distance to the oracle using cache-hit-ratio efficiency and throughput efficiency:

$$\eta_{\mathrm{CHR}} = \frac{\mathrm{CHR}_{\mathrm{SAEM}}}{\mathrm{CHR}_{\mathrm{Oracle}}}, \qquad \eta_{\mathrm{TP}} = \frac{T_{\mathrm{SAEM}}}{T_{\mathrm{Oracle}}}, \tag{2}$$

where CHR denotes the expert cache hit ratio and T denotes end-to-end decoding throughput. Values approaching 100% indicate that the online predictive policy performs close to the prediction oracle. Both terms in each ratio are measured using the same fixed-sequence replay protocol. We report ratios rather than the individual replay throughputs because the latter are not directly comparable to the free-running results in Fig. 8. For context, Table III also reports the ECR = 100% fully GPU-resident reference and SAEM's free-running throughput.

TABLE III
PREDICTION-ORACLE ANALYSIS ON QWEN3. FULL GPU IS THE ECR = 100% REFERENCE, AND SAEM REPORTS THE FREE-RUNNING THROUGHPUT FROM FIG. 8. CHR AND TP EFF. ARE DEFINED IN EQ. (2).

Batch Size	Full GPU (tokens/s)	ECR	SAEM (tokens/s)	CHR Eff. (%)	TP Eff. (%)
1	9.04	50.0%	7.19	90.36	87.87
		25.0%	6.45	81.40	90.82
		12.5%	6.10	76.09	92.80
8	20.90	50.0%	18.55	97.80	95.69
		25.0%	16.35	96.43	98.46
		12.5%	13.82	96.75	100.78

TABLE IV
INFERENCE SPEEDUP BREAKDOWN OF PROPOSED TECHNIQUES UNDER 12.5% CACHE RATIO ON QWEN3.

Batch Size	Technique	Throughput (tokens/s)	Speedup
1	Best-performing baseline	3.45	–
	Cache update only	4.81	1.39×
	In-situ CPU execution only	2.73	0.79×
	Token repacker only	5.02	1.46×
	All	6.10	1.77×
8	Best-performing baseline	8.51	–
	Cache update only	10.44	1.23×
	In-situ CPU execution only	8.51	1.00×
	Token repacker only	9.09	1.07×
	All	13.82	1.62×

Table III measures SAEM's distance from the prediction oracle in cache placement and end-to-end performance. At batch size 1, CHR efficiency decreases from 90.36% to 76.09% as ECR decreases, indicating that perfect prediction becomes increasingly beneficial for cache placement under tighter memory constraints. In contrast, TP efficiency increases from 87.87% to 92.80%, limiting the oracle throughput improvement to 1.08–1.14×. This divergence shows that higher cache accuracy does not translate proportionally into throughput because only part of the end-to-end latency is affected by expert residency.

At batch size 8, CHR and TP efficiencies remain above 96% and 95%, respectively, indicating that SAEM already operates close to the oracle under higher concurrency. The 100.78% TP efficiency at ECR = 12.5% reflects sub-1% measurement variation and is treated as parity. Overall, perfect prediction provides the greatest cache-placement improvement at low ECR and batch size 1, but only modest throughput gains. The substantially larger gap between SAEM and Full GPU therefore arises primarily from limited GPU cache capacity rather than prediction inaccuracy.

D. Ablation Study

We perform an ablation study to quantify the contribution of each SAEM component. Inference speedups are measured on Qwen3 at 12.5% ECR and reported relative to the strongest

baseline, as summarized in Table IV. Stage-aware cache updates and expert-aligned token repacking each accelerate inference in isolation, addressing complementary bottlenecks: the former retains experts likely to be reused across adjacent reasoning stages, reducing cache churn and expert migration, while the latter groups routed tokens into contiguous, hardware-friendly batches, improving memory locality and kernel efficiency. In-situ CPU execution alone, however, reaches only 0.79× at batch size 1 and merely matches the baseline at batch size 8: without stage-aware placement, the GPU cache retains a suboptimal expert set, forcing many tokens onto the slower CPU path—effectively Fiddler-style execution without the mechanisms that make hybrid execution profitable. It is thus an enabling mechanism rather than a standalone accelerator. Combining all three components yields the highest throughput (1.77× and 1.62×), confirming that the mechanisms are complementary.

To isolate the contribution of in-situ execution *within* the full system, we disable CPU-side expert execution while preserving the stage-aware cache policy and token repacking; non-resident experts are instead transferred to the GPU on demand through a reserved temporary slot. Throughput drops from 6.10 to 3.22 tokens/s at batch size 1 and from 13.82 to 9.00 tokens/s at batch size 8, confirming that avoiding on-demand PCIe transfers remains essential under constrained GPU memory. Together with the isolation results, this shows that SAEM's gains arise from coordinating hybrid CPU–GPU execution with stage-aware expert placement and kernel-efficient token organization, rather than from any single mechanism.

VI. Discussion & Limitations

SAEM improves MoE inference throughput by exploiting reasoning-stage locality for expert management. We discuss its runtime overhead and remaining limitations below.

A. Practical Overhead

SAEM introduces runtime overhead from transition-cue matching, token repacking, and stage-level expert placement. These costs are bounded by design and remain lightweight relative to MoE computation and data movement. Transition detection performs pattern matching over a short window of recently generated tokens, avoiding additional neural classifiers or semantic parsing during decoding. Token repacking uses index-based reordering with preallocated workspaces, avoiding repeated memory allocation and limiting layout-transformation overhead. Expert placement is updated only at detected stage boundaries rather than at every decoding step, amortizing cache management cost over the following reasoning stage. As shown in the ablation study, these costs are outweighed by reductions in PCIe expert transfers, cache churn, and fragmented token-access kernels, resulting in net throughput improvement.

B. Stage Boundary Detection

SAEM currently detects reasoning-stage boundaries using explicit linguistic transition cues in generated CoT traces, such as "Alternatively," and "Instead,". This strategy is effective when reasoning traces contain clear structural markers. However, some queries may lack explicit cues due to variation in generation style or task-specific reasoning patterns. In such cases, boundary detection may become less precise, reducing the effectiveness of stage-level cache updates.

To improve robustness beyond explicit textual cues, future work could explore statistical boundary signals derived from expert activations. One promising direction is to track changes in the entropy of expert activation distributions across successive tokens or short windows. Pronounced entropy shifts may indicate implicit transitions between reasoning stages even when discourse markers are absent. Such signals could complement lexical cues and improve SAEM's generality across models, reasoning styles, and workloads.

VII. Conclusion

SAEM demonstrates that semantics-aware execution strategies for reasoning workloads can significantly improve the efficiency of MoE inference for CoT reasoning. By leveraging stage-level activation patterns and applying coordinated caching, scheduling, and token repacking, SAEM reduces memory pressure and accelerates decoding. These results highlight the potential of structurally informed runtime designs in advancing scalable and efficient CoT reasoning under resource constraints.

References

[1] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[2] Q. Lyu, S. Havaldar, A. Stein, L. Zhang, D. Rao, E. Wong, M. Apidianaki, and C. Callison-Burch, "Faithful chain-of-thought reasoning," in *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 305–329.

[3] E. Yeo, Y. Tong, M. Niu, G. Neubig, and X. Yue, "Demystifying long chain-of-thought reasoning in llms," *arXiv preprint arXiv:2502.03373*, 2025.

[4] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[5] A. Jaech, A. Kalai, A. Lerer, A. Richardson, A. El-Kishky, A. Low, A. Helyar, A. Madry, A. Beutel, A. Carney *et al.*, "OpenAI o1 system card," *arXiv preprint arXiv:2412.16720*, 2024.

[6] K. T. Chitty-Venkata, J. Ye, and M. Emani, "MoPEQ: Mixture of mixed precision quantized experts," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025, pp. 4023–4032.

[7] X. Lu, Q. Liu, Y. Xu, A. Zhou, S. Huang, B. Zhang, J. Yan, and H. Li, "Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 6159–6172.

[8] A. Eliseev and D. Mazur, "Fast inference of mixture-of-experts language models with offloading," *arXiv preprint arXiv:2312.17238*, 2023.

[9] S. Zhong, Y. Sun, L. Liang, R. Wang, R. Huang, and M. Li, "HybriMoE: Hybrid CPU-GPU scheduling and cache management for efficient moe inference," in *62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.

[10] E. Yu, Z. Zhang, D. Dong, Y. Wu, and X. Liao, "PreScope: Unleashing the power of prefetching for resource-constrained MoE inference," *arXiv preprint arXiv:2509.23638*, 2025.

[11] P. Tang, J. Liu, X. Hou, Y. Pu, J. Wang, P.-A. Heng, C. Li, and M. Guo, "HOBBIT: A mixed precision expert offloading system for fast MoE inference," *arXiv preprint arXiv:2411.01433*, 2024.
[12] Z. Du, S. Li, Y. Wu, X. Jiang, J. Sun, Q. Zheng, Y. Wu, A. Li, H. H. Li, and Y. Chen, "Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 224–238, 2024.
[13] L. Xue, Y. Fu, Z. Lu, L. Mai, and M. Marina, "MoE-Infinity: Efficient MoE inference on personal machines with sparsity-aware expert cache," *arXiv preprint arXiv:2401.14361*, 2024.
[14] Z. Fang, Z. Hong, Y. Huang, Y. Lyu, W. Chen, Y. Yu, F. Yu, and Z. Zheng, "Accurate expert predictions in moe inference via cross-layer gate," *arXiv e-prints*, pp. arXiv–2502, 2025.
[15] R. Hwang, J. Wei, S. Cao, C. Hwang, X. Tang, T. Cao, and M. Yang, "Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference," in *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1018–1031.
[16] K. Kamahori, T. Tang, Y. Gu, K. Zhu, and B. Kasikci, "Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models," in *International Conference on Learning Representations*, vol. 2025, 2025, pp. 56 099–56 115.
[17] Y. Zhang, S. Aggarwal, and T. Mitra, "DAOP: Data-aware offloading and predictive pre-calculation for efficient MoE inference," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
[18] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022.
[19] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.
[20] Baidu-ERNIE-Team, "Ernie 4.5 technical report," 2025.
[21] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.
[22] Mathematical Association of America, "American invitational mathematics examination (AIME)," 2024. [Online]. Available: https://maa.org/maa-invitational-competitions
[23] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman, "GPQA: A graduate-level google-proof Q&A benchmark," in *First Conference on Language Modeling*, 2024.
[24] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6/