ASYNCSPADE: EFFICIENT TEST-TIME SCALING WITH ASYNCHRONOUS SPARSE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling (TTS) boosts LLM reasoning via long chain-of-thought (CoT), but the linear KV-cache growth amplifies the memory-bound bottleneck of LLM decoding. Query-aware page-level sparse decoding can achieve state-of-the-art performance under constrained FLOPs budgets, but is limited by both sequential-dependent page filtering and coarse-grained token selection, hampering serving efficiency and model performance on TTS tasks under high concurrency and long CoT scenarios (consuming even higher runtime than the forward pipeline itself). In this paper, we first find that the current-step query state can be accurately approximated in a unified manner from a short window of recent queries, enabling training-free query-aware sparsity without waiting in the decoding loop. We propose **`AsyncSpade`**, an asynchronous framework for efficient TTS built on two core components: (1) **a novel light-weight temporal-regressive module** that predicts the next-token query state; (2) **an asynchronous and disaggregated framework** that decouples the KV cache filtering from the auto-regressive decoding loop, overlapping the token-level KV selection with the forward inference computation through asynchronism. To our knowledge, `AsyncSpade` is the first to eliminate the sequential dependence without sacrificing model performance. We validate the effectiveness of `AsyncSpade` on common LLM serving setups with an A100 node, where `AsyncSpade` fully overlaps KV-cache operations with the inference pipeline, **achieving theoretical optimal time-per-output-token (TPOT)**. Specifically, `AsyncSpade` delivers over 20% reduction on TPOT compared to SoTA baseline (*i.e.* Quest) and at least 50% TPOT reduction compared to full attention on Qwen3-8B and Qwen3-32B models, while matching or surpassing their accuracy on various TTS benchmarks (AIME-24/25, GPQA-Diamond, MATH-500).

1 INTRODUCTION

Recent advances in large language models (LLMs) Jaech et al. (2024); Ren et al. (2025) have demonstrated remarkable capabilities in tackling complex reasoning tasks across multiple domains, including mathematical problem solving Ren et al. (2025); Wang et al. (2025); Huang & Yang (2025), code generation Ahmad et al. (2025); Huang & Yang (2025); Guo et al. (2025), and scientific discovery Huang et al. (2025); Wu et al. (2025), marking a pivotal advancement in the AI frontier. One of the most powerful paradigms that drives these advances is **Test-time scaling (TTS)**, which significantly unleashed the reasoning capabilities of LLM. Leading exemplars such as GPT-o1 Jaech et al. (2024), DeepSeek-R1 Guo et al. (2025), and QwQ Team (2024) have established that, by allocating additional computation during inference, most notably through extended chain-of-thought (CoT) Wei et al. (2022) reasoning, TTS can unlock state-of-the-art performance on a broad spectrum of challenging tasks.

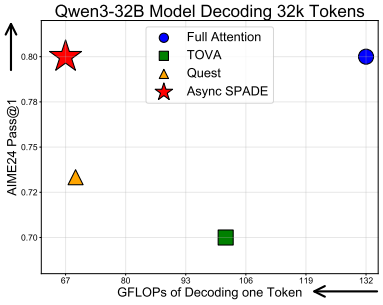
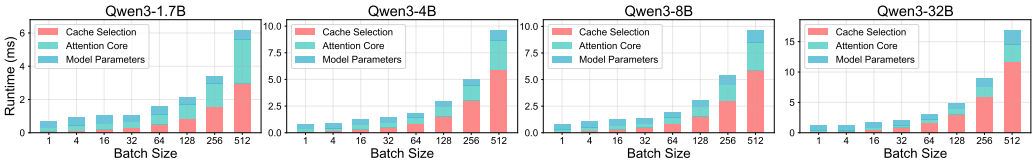
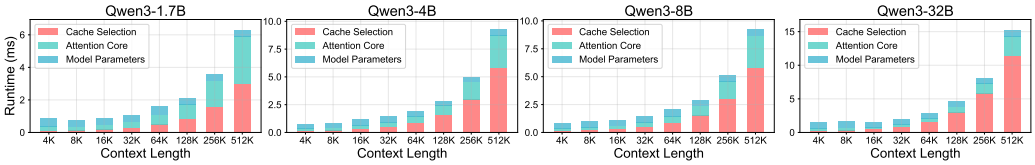


Figure 1: **Performance of Qwen3-32B on AIME24 with Long Decoding**. `AsyncSpade` minimizes the decoding FLOPs while maintaining high performance.



(a) **Latency Breakdown for Varied Concurrency**. Context length is fixed to $32k$ in all cases.



(b) **Latency Breakdown for Varied Context Length**. The batch size is fixed to 32 in all cases.

Figure 2: **Runtime Profiling for Page-level Sparse Decoding**. We benchmark the latency breakdown of a single Transformer block in the decoding stage on Qwen3 dense models Yang et al. (2025) with an NVIDIA A100 GPU with configurations in Fig. 3. We set page size to 16 following the default setting of FlashInfer Ye et al. (2025b), and select $1/16$ tokens from the full KV cache. (a) reports results for varied batch sizes (1–512) to emulate different serving concurrency, while (b) reports results for varied context lengths ($4k$–$512k$) to emulate long chain-of-thought decoding.

A critical challenge, however, is that TTS substantially prolongs the decoding stage. Each newly generated token must attend to the key-value (KV) cache of all previous tokens, resulting in the cost of attention computation growing linearly with the increase in decoding length. This linear expansion of the KV cache and memory footprint also intensifies the I/O pressure between GPU high bandwidth memory (HBM) and shared memory (SRAM), becoming a critical performance bottleneck and leading to exacerbated time-per-output-token (TPOT). In long-CoT reasoning tasks, this results in attention core Dao et al. (2022), rather than the parameter computation, emerging as the dominant performance bottleneck Sadhukhan et al. (2025), hindering the deployment of LLM in TTS scenarios under high concurrency Agrawal et al. (2024).

Figure 3: **Configurations of the profiled Qwen3 dense models**.

Model	Hidden Size	# Attn Heads	# KV Heads	Intermediate Size	# Layers
`Qwen3-1.7B`	2048	16	8	6144	28
`Qwen3-4B`	2560	32	8	9728	36
`Qwen3-8B`	4096	32	8	12288	36
`Qwen3-32B`	5120	64	8	25600	64

One promising solution is sparse decoding, *i.e.*, approximating full attention by retaining only a small, critical fraction of tokens in the KV cache during the prolonged LLM decoding process. Previous approaches exploit fixed structural heuristics, such as preserving the "attention-sink" token Xiao et al. (2024b) or leveraging historical patterns, exemplified by accumulated attention scores in H2O Zhang et al. (2023). Despite their simplicity, these query-agnostic strategies cannot fully capture factual token relevance. Recent work has highlighted that the criticality of a token strongly depends on the current query Tang et al. (2024), leading to query-aware sparsity methods Xiao et al. (2024a); Ribar et al. (2024); Tang et al. (2024). By directly leveraging current query embedding to dynamically filter relevant KV entries, these methods can achieve superior accuracy. However, they exhibit an inherent drawback, *i.e.*, KV selection creates a sequential dependence before attention computation, as it depends on the current query state. Fig. 2 demonstrates that cache selection turns out to be the dominant bottleneck of TPOT under either high concurrency or long context scenarios. Furthermore, current leading query-aware sparse attention approaches, such as Quest Tang et al. (2024) and MoBA Lu et al. (2025), adopt page or block-level selection strategies rather than token-level fine-grained granularity, which may ignore critical tokens and hamper model performance. This design stems from the deployment constraints. As the KV cache is stored on the same GPU for inference, modern GPU kernels such as FlashInfer Ye et al. (2025b) are highly optimized for reading and writing large, contiguous chunks of memory, which makes block- or page-wise access efficient. In contrast, token-level selection incurs massive, small, and irregular memory accesses that hamper runtime performance, emphasizing the necessity to advance both runtime efficiency and reasoning performance on TTS tasks.

In this paper, we propose `AsyncSpade`, a novel asynchronous sparse decoding framework for ultimate test-time scaling efficiency through decoupling the KV cache management & filtering from the

inference pipeline. To manage these two operations separately, we introduce two specialized ranks: *Inference Rank* dedicated for forward computation, and *Cache Rank* exclusively responsible for KV management and fine-grained token selection. During inference, the *Inference Rank* asynchronously transmits query, key, and value embeddings to the *Cache Rank*. Upon receiving these embeddings, the *Cache Rank* effectively regresses the next query embedding from a sliding window of previous queries, thus enabling token-level KV selection to be prepared ahead of use. The selected KV entries are then immediately transferred back to the *Inference Rank* for efficient attention computation. This duo-rank architecture brings dual benefits. First, it eliminates the sequential selection bottleneck of existing approaches. By properly batching the inter-rank communications, both the communication and the selection operation overhead can be fully pipelined with forward inference computation. Second, it unlocks the finest possible granularity through **token-level selection** by delegating token selection to a dedicated Cache GPU. This is because the associated memory reorganization overhead can be handled asynchronously and fully overlapped, without blocking the critical inference path.

In summary, our contributions are as follows:

- **Asynchronous and disaggregated design for efficient test-time scaling**: `AsyncSpade` first parallelizes the forward inference pipeline with token-level KV cache selection, enabling theoretically optimal TPOT within the context of query-aware sparse decoding.
- **Simple and effective next-query prediction**: Based on the insights of locality and linear correlation for adjacent consecutive query states, we introduce a lightweight, temporal locality-aware query-prediction algorithm that effectively forecasts the next query state.
- **High performance on both efficiency and test-time scaling tasks**: `AsyncSpade` achieves comparable performance compared to full attention while consistently reducing the TPOT across common LLM serving scenarios by over 20% compared to the strong baseline of Quest and over 50% compared to the full-attention baseline.

2 Related Works

Test-Time Scaling is a effective paradigm to substantially improved the reasoning ability of LLMs by allocating extra computation at inference Zhang et al. (2025). Existing efforts mainly follow two strategies: (1) *Sequential scaling* prolongs reasoning trajectories before producing final answers, exemplified by Long-CoT Wei et al. (2022), and widely adopted in models such as GPT-o1 Jaech et al. (2024), DeepSeek-R1 Guo et al. (2025), QwQ Team (2024), Qwen3 Yang et al. (2025), GPT-OSS Agarwal et al. (2025), and LIMO Ye et al. (2025a). (2) *Parallel scaling* instead expands the solution space via multiple generations. Multi-sample decoding Sun et al. (2024) and self-consistency Wang et al. (2023) instantiate this strategy by sampling diverse reasoning paths. Beyond sampling-based methods, search-based algorithms (e.g., tree search, Monte Carlo Tree Search) Chaffin et al. (2022); Yao et al. (2023) explicitly structure reasoning into combinatorial trajectories. While effective at boosting reasoning quality, these approaches increase inference latency and memory cost, motivating complementary strategies from the system side.

Dynamic KV Cache Sparsity exploits context-aware strategies to approximate attention scores. H2O Zhang et al. (2023) utilize accumulated history attention scores to select the critical tokens. FastGen Ge et al. (2024) adaptively compresses the KV cache by profiling attention heads and evicting tokens according to their contextual focus. Loki Singhania et al. (2024) employs offline principal component analysis calibration to reduce key vector dimensions along with top-k selection. However, these approaches may still prune truly important tokens, since they approximate relevance without conditioning on the actual query at the current decoding step.

Query-aware KV Cache Sparsity addresses this limitation by explicitly leveraging the query embedding. SparQ Ribar et al. (2024) instantiates this method by selecting top-r dimensions of the query and pruning tokens accordingly. Quest Tang et al. (2024) exploits query-aware estimates of attention score bounds to selectively load relevant KV cache pages and improve efficiency. Moba Lu et al. (2025)introduces a mixture-of-experts inspired block-wise attention that dynamically routes queries to different blocks. However, these methods either suffer from sequential dependency between token selection and inference computation, which incurs non-trivial latency, or adopt block-level coarse granularity that limits accuracy. In contrast, `AsyncSpade` achieves token-level selection without sequential dependency through a novel asynchronous and disaggregated KV cache selection mechanism.

3 OBSERVATION

In this section, we present two key observations on query embeddings and KV cache that motivate our approach. To quantify the similarity between two query states, we use their selectivity on the KV cache as a proxy, and introduce a metric called *overlap ratio*. For two queries q_i and q_j, we use $S(q_i)$ and $S(q_j)$ to denote their respective selected token sets, where $|S(q)|$ denotes the number of tokens in the set $S(q)$. The *overlap ratio* is then defined as

$$\mathcal{O}_{i,j} = \frac{|S(q_i) \cap S(q_j)|}{|S|} \tag{1}$$

where $|S|$ is the fixed number of selected tokens for each query. We assert that both q_i and q_j select the same number of tokens in the KV cache, *i.e.*, $|S|$=$|S(q_i)|$=$|S(q_j)|$.

3.1 TEMPORAL LOCALITY OF QUERY STATES

ShadowKV Sun et al. (2025) demonstrates that the KV cache exhibits strong temporal locality, *i.e.*, the sets of KV entries retrieved by consecutive query states share a high proportion of intersection. Inspired by this, we conduct additional profiling experiments using the proposed metric of *overlap ratio* to analyze the temporal locality of the KV cache on test-time reasoning tasks.

We illustrate in Fig. 4 that there is strong temporal locality in the filtered KV sets, where the most recent 16 tokens consistently maintain high overlap ratios of over 40% throughout the generation process. The tendency of queries at adjacent decoding steps to select similar KV subsets demonstrates that attention patterns of nearby queries are highly correlated. These empirical results imply that the attention distribution of a query carries predictive information about its successors, suggesting *the feasibility of approximating the future query attention patterns by historical query information.*

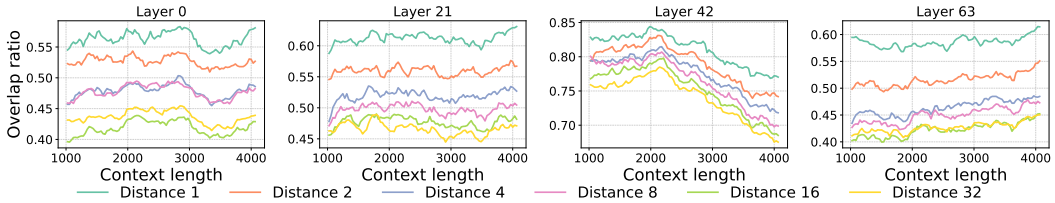


Figure 4: ***Overlap ratio* for query states across different token distances**. The *overlap ratio* for distance d and token t is examined with $\mathcal{O}_{t-d,t}$. We use Qwen3-32B and AIME24 with full attention for the profiling experiments, where the *overlap ratio* is averaged over the sample and attention head dimensions, and 4 layers are examined. $1/8$ tokens from the KV cache are selected.

3.2 LINEAR CORRELATION OF ADJACENT QUERIES

We further investigate whether this temporal locality can be modeled in a unified perspective and find that the historical query states possess a strong linear correlation with the current query state. We denote the sliding query states as $\{Q_{t-W}, \ldots, Q_t\}$, where W is the window size. To demonstrate the linear correlation between Q_t and its adjacent queries $\{Q_{t-W}, \ldots, Q_{t-1}\}$, we regress Q_t from the W predecessors by solving a ridge regression problem. Assuming that Q_t can be expressed with a group of softmax-normalized weights $\{\omega_{t-W}, \ldots, \omega_{t-1}\}$ from the windowed historical queries:

$$\omega^\star = \arg\min_{\omega \in \mathbb{R}^W} \left\| \sum_{i=1}^{W} \frac{\exp(\omega^\star_{t-i})}{\sum_{j=1}^{W} \exp(\omega^\star_{t-j})} Q_{t-i} - Q_t \right\|_2^2 + \epsilon \sum_{i=1}^{W} \omega_{t-i}^2, \tag{2}$$

where $\epsilon > 0$ is used for regularization and ω_{t-i} corresponds to the historical query state Q_{t-i}. To demonstrate the effectiveness of this approximation, we further apply $\omega^\star$ also on the inputs:

$$\tilde{Q}_t = \sum_{i=1}^{W} \frac{\exp(w^\star_{t-i})}{\sum_{j=1}^{W} \exp(w^\star_{t-j})} \cdot Q_{t-i}, \quad i = 1, \ldots, W. \tag{3}$$

We then compare the attention score distribution induced by $\tilde{Q}_t$ and the ground-truth Q_t by visualizing the *overlap ratio* of their top-k token selection within the full KV cache, as demonstrated in

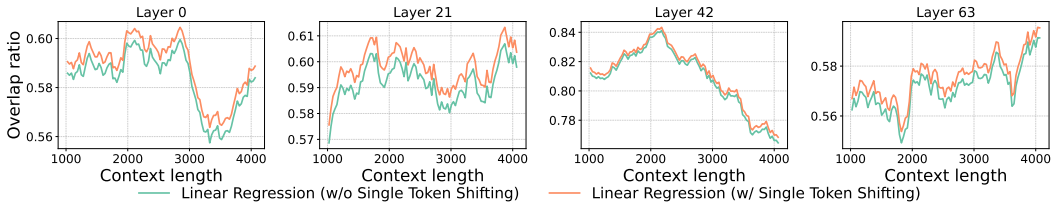


Figure 5: ***Overlap ratio* for linear regression w/ & w/o single-token shifting.** We follow the same settings as Fig. 4. Given window size W, the *overlap ratio* of token t for linear regression w/o single token shifting is examined by first regressing token t with token $\{t-W,\ldots,t-1\}$ and then apply the solved weights also on token $\{t-W,\ldots,t-1\}$, while the *overlap ratio* of token t w/ single token shifting is examined by first regressing token $t-1$ with token $\{t-W-1,\ldots,t-2\}$ and then apply the solved weights on token $\{t-W,\ldots,t-1\}$. $W=16$ is used for profiling.

the green line in Fig. 5. The relatively high *overlap ratio* indicates that the attention pattern of Q_t is possible to be modeled by a linear combination of its consecutive preceding query states.

4 METHODOLOGY

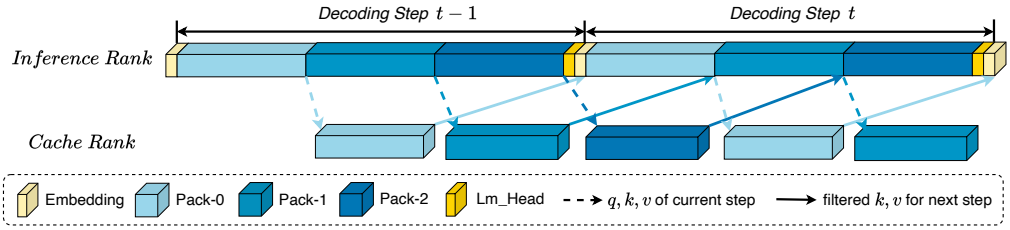


Figure 6: **Workflow of `AsyncSpade`**. We illustrate the overall workflow of `AsyncSpade` with 2 consecutive decoding steps, where LLM parameters are assembled into 3 packs, with fully overlapped cross-device communication and cache management, delivering theoretically optimal TPOT.

In this section, we introduce `AsyncSpade`, an algorithm-system co-design approach that optimizes TPOT for serving LLM on test-time scaling tasks through rank disaggregation, asynchronism, and fine-grained sparsity. Sec. 4.1 introduces the overall design principles of `AsyncSpade`, including the proposed disaggregated architecture and the workflow. Sec. 4.2 describes the algorithmic design for asynchronous cache selection with token-level granularity. Sec. 4.3 further provides the implementation details that support the ultimate decoding efficiency of `AsyncSpade`.

4.1 ASYNCHRONOUS AND DISAGGREGATED FRAMEWORK FOR SPARSE DECODING

In conventional query-aware sparse decoding methods, KV selection must wait for the current query embedding to be computed, and the attention core can only be launched after the selected KV is obtained. `AsyncSpade` first breaks this dependency by decoupling the KV selection operation from the inference pipeline, eliminating redundant operations in the inference pipeline. To achieve this, `AsyncSpade` predicts the query state of the next token and filters KV entries at token-level granularity based on it. This process can prepare the most relevant KV candidates for the next decoding step in advance, and is conducted in parallel with the inference computation pipeline.

Inference Rank* and *Cache Rank To fully overlap the KV cache filtering operations, we employ two specialized logical ranks to handle separate operations: *Inference Rank* for the forward inference pipeline, including attention core and parameter computation, and *Cache Rank* for managing and filtering the KV caches, coordinated through P2P asynchronous communication for efficient LLM decoding. *Inference Rank* transmits the computed query, key, and value states to the *Cache Rank* for management and selection, while *Cache Rank* returns the filtered KV cache to the *Inference Rank*, enabling parallelized execution and fine-grained cache granularity.

Communication and Computation Workflow During LLM inference, once the generation length exceeds a predefined threshold (denoted as step θ_t), the model transitions to sparse decoding mode. At step θ_t-1, immediately after computing the query state, the *Inference Rank* transfers

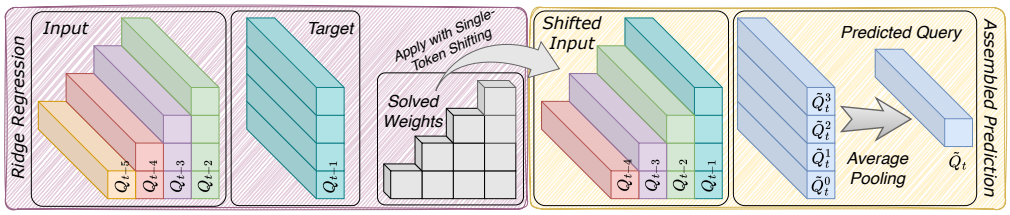


Figure 7: **Assembled Regression with Average Pooling in `AsyncSpade`**. We use ridge regression to learn from each historical query window. Here we set the window size to 4 for illustration.

all previous KV entries along with several recent query embeddings to the *Cache Rank* for initialization. While the *Inference Rank* continues processing forward inference for the token at step $\theta_t - 1$, the *Cache Rank* operates in parallel to predict the query embedding for the next step θ_t using a sliding window over recent queries, and applies this predicted query to filter the KV cache. The selected KV pairs are subsequently transferred back to the *Inference Rank* and participate in the attention computation of the decoding step θ_t. For the subsequent decoding steps, the query & key & value states of the current step are packed and passed to the *Cache Rank*. The *Cache Rank* appends KV states to the KV cache and enqueues the query state to the sliding window, then proactively performs **token-level KV cache filtering** required for the next decoding step.

4.2 Token Criticality Estimation with Historical Sliding Query

A central challenge in sparse decoding is identifying the most critical set of key-value (KV) pairs to maintain model performance. Conventional query-aware methods Tang et al. (2024); Lu et al. (2025) directly use the query state at the current decoding step for cache selection to adhere to the definition of attention core Vaswani et al. (2017). However, this sequential dependency and coarse-grained granularity can hamper both LLM serving efficiency under heavy concurrency and the model reasoning performance on TTS tasks Łańcucki et al. (2025). To tackle these drawbacks, `AsyncSpade` effectively predicts the query in advance to approximate the token criticality.

Temporal-Regressive Prediction with Adjacent Query States Based on the insights in Sec. 3, `AsyncSpade` exploits the observation that consecutive query states exhibit strong temporal locality and can be well approximated as a linear combination of their predecessors. Based on these observations, we directly apply the solved weights to a single-token-shifted sliding query window to predict the query states for the next token. At current decoding step t, we first obtain the regression weights $\{\omega_{t-W}, \ldots, \omega_{t-1}\}$ by solving the ridge regression problem in Eq. (2), also with the softmax-normalization in Eq. (3). Each weight ω_{t-i} corresponds to the historical query Q_{t-i}, reflecting its contribution to regressing Q_t. These weights are then *applied* to the shifted query sequence $\{Q_{t-W+1}, \ldots, Q_t\}$ to *predict* the next query state $\hat{Q}_{t+1}$ at step $t+1$:

$$\hat{Q}_{t+1} = \sum_{i=1}^{W} w_{t-i}\, Q_{t+1-i}. \tag{4}$$

Fig. 5 demonstrates the practicality of this temporal-regressive prediction strategy, where the *overlap ratio* with single-token shifting can even surpass that of the counterpart without shifting.

Assembled Regression with Average Pooling To better integrate the historical queries within a certain temporal range in a unified manner, we further extend the ridge regression by first assembling the solved weights from multiple window sizes and then pooling the obtained states. Rather than relying on a single window size, we perform regression across all window sizes $k \in \{1, \ldots W\}$. For each window size k, we use the states $\{Q_{t-k}, \ldots, Q_{t-1}\}$ to regress Q_t by first solving the ridge regression problem defined in Eq. (2), followed by the softmax normalization in Eq. (3) to yield the corresponding weights $W_k = \{w_{t-k}^{(k)}, \ldots, w_{t-1}^{(k)}\}$. Each weight $w_{t-i}^{(k)}$ corresponds to the historical query Q_{t-i} These weights are then applied to the shifted sequence $\{Q_{t-k+1}, \ldots, Q_t\}$ and yield a candidate estimation of the next query embedding $\hat{Q}_{t+1}^{(k)} = \sum_{i=1}^{k} w_{t-i}^{(k)} Q_{t+1-i}$. This process generates m complementary estimates $\{\hat{Q}_{t+1}^{(1)}, \ldots, \hat{Q}_{t+1}^{(m)}\}$, each capturing temporal locality at different scales.

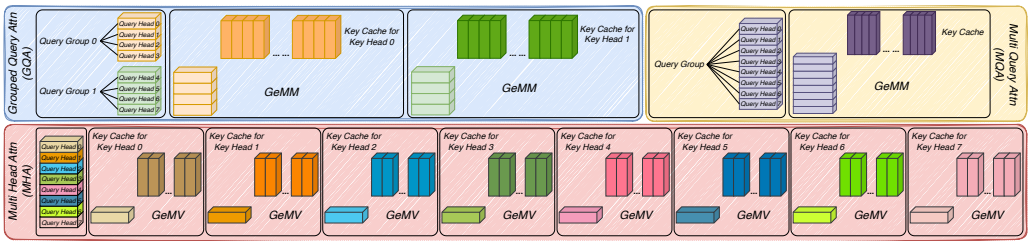


Figure 8: **Apply `AsyncSpade` to Different Attention Architectures through Batched MatMul**. While conventional Multi-Head Attention (MHA) is restricted to implementing criticality estimation only through GeMV, the Group Query Attention (GQA) and Multi-Query Attention (MQA/MLA) architectures prevalent in modern LLMs can achieve this through GeMM.

Finally, these estimates are aggregated through average pooling to obtain the final estimation of the next query embedding:

$$\hat{Q}_{t+1} = \frac{1}{m}\sum_{k=1}^{m} \hat{Q}_{t+1}^{(k)} \tag{5}$$

Since this assembled regression problem is very simple and lightweight, it can be efficiently solved on GPUs with negligible runtime.

4.3 Hardware-Efficient Implementation

Depth-wise Parallelism on Cache Rank To fully overlap the communication overhead across *Inference & Cache Ranks* as well as better utilize the parallel computation resources on the *Cache Rank*, we assemble several consecutive transformer decoder blocks in the LLM into a packed unit to conduct asynchronous transmission between the *Inference* and *Cache Ranks* at unit granularity rather than performing separate communications for each layer. In this way, we can reduce the launch times for cross-device communication and better utilize the bandwidth. Upon receiving the bundled states from multiple blocks sent by the *Inference Rank*, the *Cache Rank* can perform filtering operations with parallelism along the depth dimension.

Batched Matmul for Different Attention Architectures We investigate the adaptation of `AsyncSpade` on all the softmax attention variants in modern LLMs, including Multi-Head Attention (MHA Vaswani et al. (2017)), Grouped Query Attention (GQA Ainslie et al. (2023)), and Multi-Query Attention (MQA Shazeer (2019)). Specifically, the Multi-Head Latent Attention in the DeepSeek series Liu et al. (2024a;b); Guo et al. (2025) and Kimi-K2 Team et al. (2025) can be exactly transformed into MQA during the inference stage through matrix absorption. Here, we treat all these architectures as variants of GQA. Denoting the number of attention (query) heads as N_{q} and the number of key & value heads as N_{kv}. To be specific,

- MHA possesses N_{q} query groups, and $N_{\text{q}} = N_{\text{kv}}$.
- GQA possesses $N_{\text{q}}/N_{\text{kv}}$ query groups, and N_{q} should be divisible by N_{kv}.
- MQA/MLA possesses only 1 query group, and $N_{\text{kv}} = 1$.

We present a comprehensive investigation on applying `AsyncSpade` to these attention variants using native PyTorch interfaces in Fig. 8. Denoting the batch size as bs, head dimension as D_{h}, and the number of tokens in the KV cache as N_{t}. Since the query state only contains one token during the decoding stage, the tensor shape can be formulated as $(bs, N_{\text{kv}}, N_{\text{q}}/N_{\text{kv}}, D_{\text{h}})$, and the key states can be correspondingly formulated as $(bs, N_{\text{kv}}, N_{\text{t}}, D_{\text{h}})$. `AsyncSpade` for MHA can only be implemented with flattened GeMV, while other architectures can be implemented with flattened GeMM, which can better utilize the tensor core, the matrix computation unit on NVIDIA GPUs, for better runtime efficiency. Since most of the modern LLMs are built on GQA or MLA, the *Cache Rank* GPUs can therefore be effectively utilized for token criticality estimation.

Communication-Computation Overlap We illustrate the overall workflow in Fig. 6, which presents the communication-computation overlapping strategies in `AsyncSpade`. The hardware requirements for `AsyncSpade` lie in two aspects:

- The latency of a P2P communication cycle, including (1) sending the packed key & value (& query) states from *Inference Rank* to *Cache Rank*, and (2) sending the

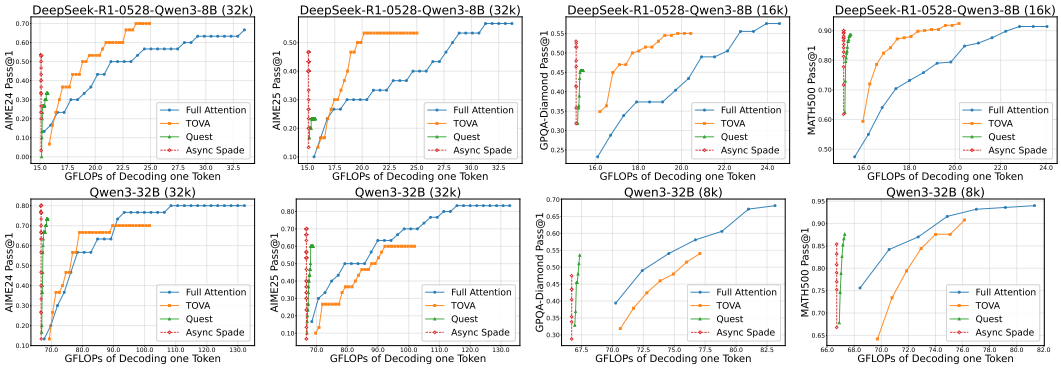


Figure 9: **Performance Comparison on TTS Benchmarks**. We examine the Pass@1 (y-axis) and the FLOPs of decoding one token (x-axis) for 8B and 32B models with $2k$ selected tokens for sparse methods, and `AsyncSpade` consistently outperforms the strong baselines and achieves the highest performance while consuming the least FLOP budgets.

packed filtered KV cache from *Cache Rank* to *Inference Rank*, should be less than the corresponding forward computing latency on the *Inference Rank*.

- The latency of processing a packed state on the *Cache Rank*, including (1) token criticality estimation, (2) top-k selection, and (3) KV cache re-organization for filtered tokens, should also be less than the corresponding forward computing latency on the *Inference Rank*.

5 Experiments

5.1 Setups

Models, Datasets, and Baselines We evaluate `AsyncSpade` across four popular and challenging test-time reasoning benchmarks: AIME24, AIME25 Art of Problem Solving (2025), GPQA-Diamond Rein et al. (2024) and MATH500 Hendrycks et al. (2021). We employ state-of-the-art open-source LLMs in our experiments, specifically Qwen3-32B Yang et al. (2025) and DeepSeek-R1-0528-Qwen3-8B Guo et al. (2025). To make a rational and comprehensive comparison, we benchmark against leading query-aware sparse attention algorithms, including TOVA Oren et al. (2024) and Quest Tang et al. (2024). Our experiments are conducted on two nodes: N_1 equipped with 8× NVIDIA A100 (80 GB) GPUs and N_2 with 8× NVIDIA H100 (80 GB) GPUs. The hardware specs are provided in Table. 1.

Table 1: **Hardware Specs for the 2 Nodes**.

Node Specs	N_1	N_2
GPU	8×A100 SXM	8×H100 SXM
GPU Memory	80GB	80GB
Inter-GPU Bandwidth	250 GB/s	350 GB/s
PCIe	4.0 ×16	5.0 ×16
NVLink Generation	3rd	4th

5.2 Performance Comparison

We systematically evaluate the performance of `AsyncSpade` on popular test-time scaling benchmarks, using three strong baselines: Quest Tang et al. (2024) with a page size of 16, TOVA Oren et al. (2024), and vanilla full attention. We present the comprehensive performance comparison in Fig. 9, where the Pass@1 solving rate is used as the y-axis and the FLOPs of decoding one token is used as the x-axis. We provide the definition details of Decoding FLOPs in appendix B. We select $2k$ tokens for each sparse decoding method. `AsyncSpade` consistently outperforms other approaches with minimized FLOPs budget and comparable or better solving rate.

5.3 Efficiency Comparison

We benchmark the TPOT performance of `AsyncSpade` and other baselines under the same decoding context length and concurrency level, visualized in Fig. 10. `AsyncSpade` achieves theoretically minimized TPOT on cutting-edge data-center GPUs. To be specific, **(1) Compared to Quest**, `AsyncSpade` can fully overlap the KV cache filtering overheads; meanwhile, `AsyncSpade` can also be deployed with any LLM inference backends rather than restricted to the paged inference

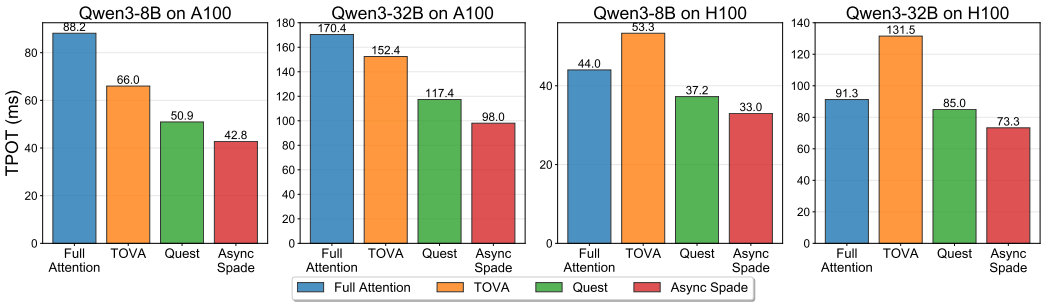


Figure 10: **Runtime Comparison Across Multiple Configurations**. TPOT is examined with batch size 8 on $32k$ decoding context. `AsyncSpade` consistently outperforms other strong baselines with minimized TPOT under concurrent serving and long decoding scenarios.

Table 2: **Ablation Studies for `AsyncSpade`**. We investigate the cache management latency and inter-GPU communication bandwidth required for fully overlapping communication overheads on both Qwen3-8B and 32B models, using both A100 and H100 nodes.

	Device	Packing Config	Inference	Cache Management	Minimal Bandwidth
Batch Size 8 Qwen3-8B 32k tokens select 2k	A100	Packing 6 Layers	7.13 ms	6.42 ms	107.71 GB/s
		Packing 12 Layers	14.26 ms	12.75 ms	107.71 GB/s
	H100	Packing 6 Layers	5.47 ms	3.92 ms	140.40 GB/s
		Packing 12 Layers	10.94 ms	7.85 ms	140.40 GB/s
Batch Size 16 Qwen3-8B 16k tokens select 1k	A100	Packing 6 Layers	7.30 ms	7.02 ms	105.20 GB/s
		Packing 12 Layers	14.60 ms	14.43 ms	105.20 GB/s
	H100	Packing 6 Layers	5.91 ms	5.18 ms	129.94 GB/s
		Packing 12 Layers	11.82 ms	11.14 ms	129.94 GB/s
Batch Size 8 Qwen3-32B 32k tokens select 2k	A100	Packing 4 Layers	6.30 ms	5.16 ms	159.49 GB/s
		Packing 8 Layers	12.60 ms	8.32 ms	159.49 GB/s
	H100	Packing 4 Layers	4.39 ms	3.74 ms	228.88 GB/s
		Packing 8 Layers	8.78 ms	7.48 ms	228.88 GB/s
Batch Size 16 Qwen3-32B 16k tokens select 1k	A100	Packing 4 Layers	7.77 ms	7.02 ms	129.32 GB/s
		Packing 8 Layers	15.54 ms	14.01 ms	129.32 GB/s
	H100	Packing 4 Layers	4.37 ms	3.72 ms	229.93 GB/s
		Packing 8 Layers	8.74 ms	7.48 ms	229.93 GB/s

engine Ye et al. (2025b) in Quest. **(2) Compared to TOVA**, which can achieve better performance than Quest but is significantly less practical, `AsyncSpade` makes this fine-grained strategy deployable through the innovative asynchronous and disaggregated design.

5.4 Ablation on the Latency and Bandwidth

We tests overheads of different components in `AsyncSpade`. Table. 2 shows that the cache management latency is consistently lower than inference time across all configurations. We also compute the minimal bandwidth required to achieve a fully overlapped workflow for each configuration. Our analysis shows that all the bandwidth requirements can fall well within the capabilities of the corresponding hardware specifications listed in Table. 1. This demonstrates that `AsyncSpade` can effectively overlap communication and KV selection overheads under practical hardware constraints.

6 Conclusions and Future Work

We introduce `AsyncSpade`, an algorithm-system co-design approach to optimize TPOT for test-time-scaled LLM decoding. By asynchronously disaggregating KV-cache management from the forward pass, `AsyncSpade` eliminates redundant operations within the model inference pipeline, delivering both heavy concurrency support and high performance on test-time scaling tasks, especially achieving theoretically optimal TPOT on common LLM serving scenarios. As a preliminary work on training-free, high-performance, and efficient test-time scaling, `AsyncSpade` opens multiple avenues to further boost LLM decoding efficiency without sacrificing model performance.

7 Reproducibility Statement

We provide all implementation details necessary to reproduce our results. All experiments use publicly available models (DeepSeek-R1-0528-Qwen3-8B and Qwen3-32B) and public benchmarks (AIME24, AIME25, GPQA-Diamond, MATH500). Hardware configurations, detailed pseudocode for the core algorithms and experimental settings are specified in the paper.

8 Ethics Statement

This work does not involve human subjects, sensitive personal data, or information that could raise privacy concerns. The datasets and benchmarks we use are publicly available, and our methods do not introduce discrimination or unfair bias. We believe our research poses no foreseeable ethical risks and adheres to the ICLR Code of Ethics.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*, 2025.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=hmOwOZWzYE.

Art of Problem Solving. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2025. Accessed: September 18, 2025.

Antoine Chaffin, Vincent Claveau, and Ewa Kijak. Ppl-mcts: Constrained textual generation through discriminator-guided mcts decoding. In *NAACL 2022-Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1–15, 2022.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uNrFpDPMyo.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Xiaoke Huang, Juncheng Wu, Hui Liu, Xianfeng Tang, and Yuyin Zhou. m1: Unleash the potential of test-time scaling for medical reasoning with large language models. *CoRR*, 2025.

Yichen Huang and Lin F Yang. Gemini 2.5 pro capable of winning gold at imo 2025. *arXiv preprint arXiv:2507.15855*, 2025.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *CoRR*, 2024.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo M Ponti. Inference-time hyper-scaling with kv cache compression. *arXiv preprint arXiv:2506.05345*, 2025.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient LLM inference. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL https://openreview.net/forum?id=Ue8EHzaFI4.

Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen. Kinetics: Rethinking test-time scaling laws. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37: 16692–16723, 2024.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. Fast best-of-n decoding via speculative rejection. *Advances in Neural Information Processing Systems*, 37:32630–32652, 2024.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=oa7MYAO6h6.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 47901–47911, 2024.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown. *Hugging Face*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Juncheng Wu, Wenlong Deng, Xingxuan Li, Sheng Liu, Taomian Mi, Yifan Peng, Ziyang Xu, Yi Liu, Hyunjin Cho, Chang-In Choi, et al. Medreason: Eliciting factual medical reasoning steps in llms via knowledge graphs. *arXiv preprint arXiv:2504.00993*, 2025.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=bTHFrqhASY.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=NG7sS51zVF.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025a.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. In *Eighth Conference on Machine Learning and Systems*, 2025b.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, et al. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

A LLM USAGE CLARIFICATION

Large Language Models were used in this work solely for grammatical correction and language editing purposes.

B DEFINITION OF DECODING FLOPS FOR ONE TOKEN

FLOPs for Different Decoding Strategies We provide the theoretical computation cost in terms of floating-point operations (FLOPs) for different decoding strategies, following the parameter settings of DeepSeek-R1-0528-Qwen3-8B and Qwen3-32B. The total computation cost is decomposed into two parts: (1) parameter compute cost C_{param}, which comes from the dense linear layers (attention projection, key/value projection, and feed-forward network), and (2) attention-related cost C_{attn}, which varies with the decoding strategy.

The parameter compute cost can be written as

$$C_{\text{param}} = l \cdot \big(2 \cdot 2 \cdot H \cdot q \cdot h + 2 \cdot 2 \cdot kv \cdot h \cdot H + 3 \cdot 2 \cdot H \cdot i\big), \tag{6}$$

where l is the number of layers, H is the hidden size, q is the number of attention heads, kv is the number of KV heads, h is the head dimension, and i is the intermediate dimension of the feed-forward network.

The attention cost depends on the sparsity strategy:

- **Full Attention**:
$$C_{\text{attn}} = l \cdot (4 \cdot q \cdot h \cdot T), \tag{7}$$
where T is the total sequence length.
- **TOVA**:
$$C_{\text{attn}} = l \cdot (4 \cdot q \cdot h \cdot C \ + \ 2 \cdot q \cdot h \cdot T), \tag{8}$$
where C is the number of selected tokens.
- **Quest**:
$$C_{\text{attn}} = l \cdot (4 \cdot q \cdot h \cdot C \ + \ 2 \cdot q \cdot h \cdot (T/P)), \tag{9}$$
where P is the page size for block-level sparsity.
- **`AsyncSpade` (Our Method)**:
$$C_{\text{attn}} = l \cdot (4 \cdot q \cdot h \cdot C). \tag{10}$$

Finally, the total FLOPs is obtained as

$$C_{\text{FLOPs}} = C_{\text{param}} + C_{\text{attn}}. \tag{11}$$

Theoretically Optimal TPOT If the cache management & filtering operations in `AsyncSpade` can be fully overlapped by the forward inference pipeline on the *Inference Rank*, then we only need to take (1) the parameter computation and (2) the attention core computation with the selected tokens into consideration, which consumes exactly fewer FLOPs than all the other counterparts, *i.e.*, achieving theoretically optimal TPOT performance.

C SOLVING SOFTMAX-NORMALIZED RIDGE REGRESSION

We provide the detailed pseudo code for solving the softmax-normalized ridge regression problem in `AsyncSpade` in algorithm 1. This solving program consumes negligible runtime even under heavy concurrency because it is very lightweight.

Algorithm 1 Assembled KV Cache Filtering

Require:
Cached query window: $\mathbf{Q}_{\text{cache}} \in \mathbb{R}^{B\times H\times W\times D}$
Current query: $\mathbf{q}_{\text{curr}} \in \mathbb{R}^{B\times H\times 1\times D}$
Key states: $\mathbf{K} \in \mathbb{R}^{B\times H\times L\times D}$
Regularization parameter: ϵ
Window size: W
Ensure: Attention logits: $\mathbf{L} \in \mathbb{R}^{B\times H\times L}$
procedure ASSEMBLEDFILTER($\mathbf{Q}_{\text{cache}}, \mathbf{q}_{\text{curr}}, \mathbf{K}, \epsilon, W$)
    Step 1: Input Preparation
    $\mathbf{Q}_{\text{hist}} \leftarrow \text{reshape}(\mathbf{Q}_{\text{cache}}[:,:,:W-1,:], [B\cdot H, W-1, D])$
    $\mathbf{q}_{\text{prev}} \leftarrow \text{reshape}(\mathbf{Q}_{\text{cache}}[:,:,-1:,:], [B\cdot H, 1, D])$
    Step 2: Mask Construction
    $\mathbf{M} \leftarrow \text{lower\_triangular\_mask}(W) \in \{0,1\}^{W\times W}$
    $\mathbf{M} \leftarrow \text{broadcast}(\mathbf{M}, [B\cdot H, W, W])$
    Step 3: Ridge Regression
    $\mathbf{G} \leftarrow \mathbf{Q}_{\text{hist}}\mathbf{Q}_{\text{hist}}^{\top} + \epsilon\mathbf{I}$ ▷ Gram matrix with regularization
    $\mathbf{b} \leftarrow \mathbf{Q}_{\text{hist}}\mathbf{q}_{\text{prev}}^{\top}$
    $\mathbf{w}_{\text{base}} \leftarrow \text{softmax}(-\mathbf{G}^{-1}\mathbf{b})$ ▷ Solve ridge regression problem
    Step 4: Weight Matrix Construction
    $\mathbf{W}_{\text{matrix}} \leftarrow \text{broadcast}(\mathbf{w}_{\text{base}}, [B\cdot H, W, W-1])$
    $\mathbf{W}_{\text{matrix}} \leftarrow \mathbf{W}_{\text{matrix}} \odot \mathbf{M}$ ▷ Apply mask
    $\mathbf{W}_{\text{matrix}} \leftarrow \text{softmax}(\mathbf{W}_{\text{matrix}})$ ▷ Normalize
    Step 5: Query Prediction
    for $j = 1$ **to** W **do**
        $\mathbf{Q}_j \leftarrow \mathbf{Q}_{\text{cache}}[:,:,-1-j:-1,:]$
        $\mathbf{w}_j \leftarrow \mathbf{W}_{\text{matrix}}[:,j,:j]$
        $\mathbf{q}_{\text{candidate}}^{j} \leftarrow \sum_{k=1}^{j} w_j^k \cdot \mathbf{q}_j^k$ ▷ Shifted weighted sum
    end for
    Step 6: Query Average Pooling
    $\mathbf{q}_{\text{final}} \leftarrow \frac{1}{W}\sum_{j=1}^{W} \mathbf{q}_{\text{candidate}}^{j}$ ▷ Average over all candidate next query
    Step 7: KV Cache Selection
    $\mathbf{L} \leftarrow \mathbf{q}_{\text{final}}\mathbf{K}^{\top}$ ▷ Select significant KV pairs
    $\mathbf{L} \leftarrow \text{reshape}(\mathbf{L}, [B, H, L])$
    return $\mathbf{L}$
end procedure