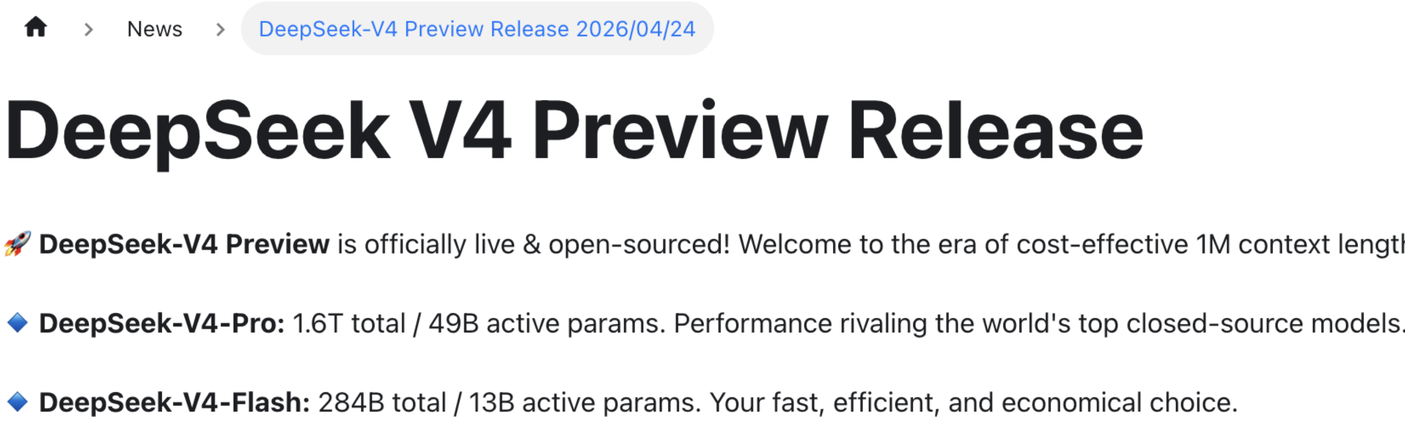
Accelerating Large-Scale Reasoning Model Inference with Sparse Self-Speculative Decoding

Yilong Zhao*, Jiaming Tang*, Kan Zhu, Zihao Ye, Chaofan Lin, Jongseok Park, Guangxuan Xiao, Alvin Cheung, Mingyu Gao, Baris Kasikci, Song Han, Ion Stoica

Background

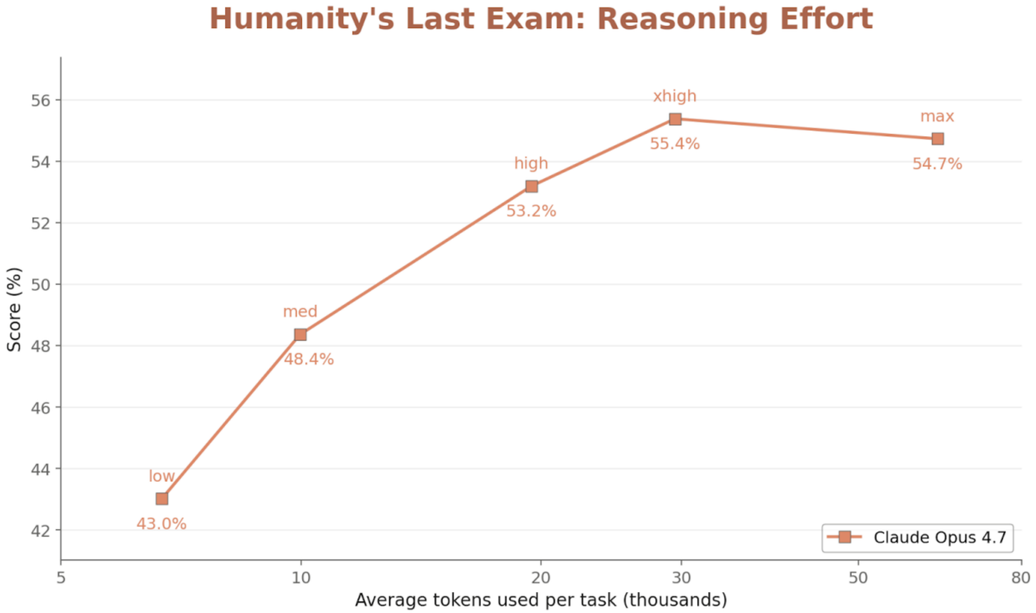
Large Reasoning Models (LRMs) Are Becoming Increasingly Prevalent



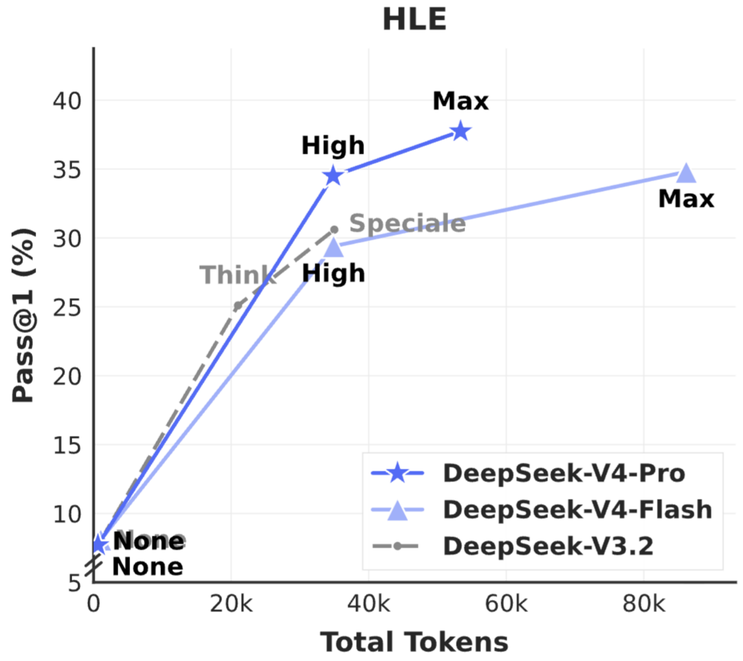
Challenge of LRM Serving and Training

Long Reasoning Makes LRM Serving and Training Slow

- **Challenge**: LRM serving and training rollout is slow due to the length of the **long chain of thoughts**.
- **Goal:** Accelerate **LRM serving** and **RLVR** with **lossless** generation via a **plug-and-play** way **without overhead**.



Claude Opus 4.7 uses 30~70K tokens on HLE

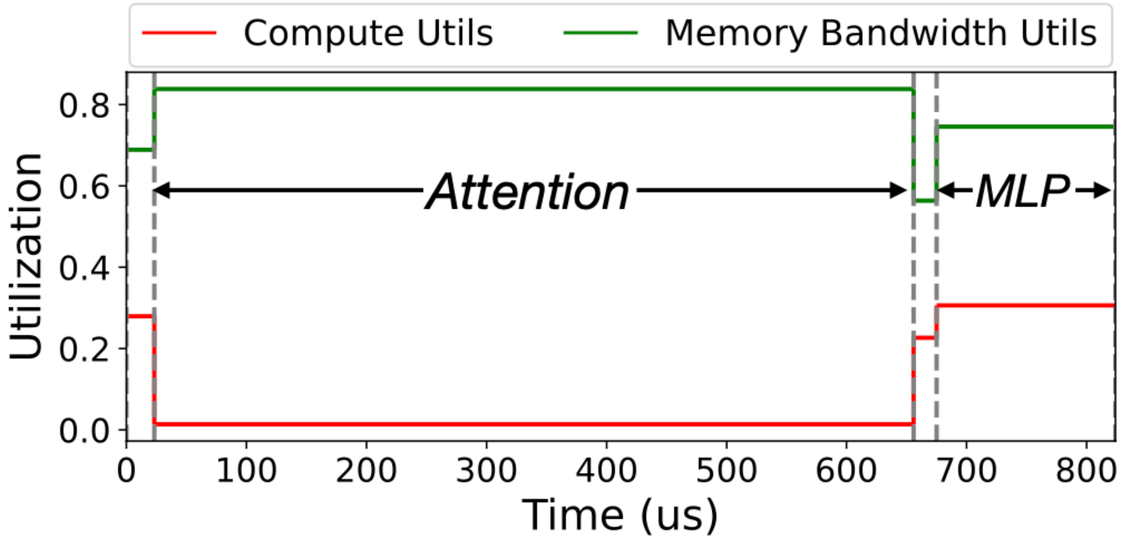


DeepSeek V4 uses 50~90K tokens on HLE

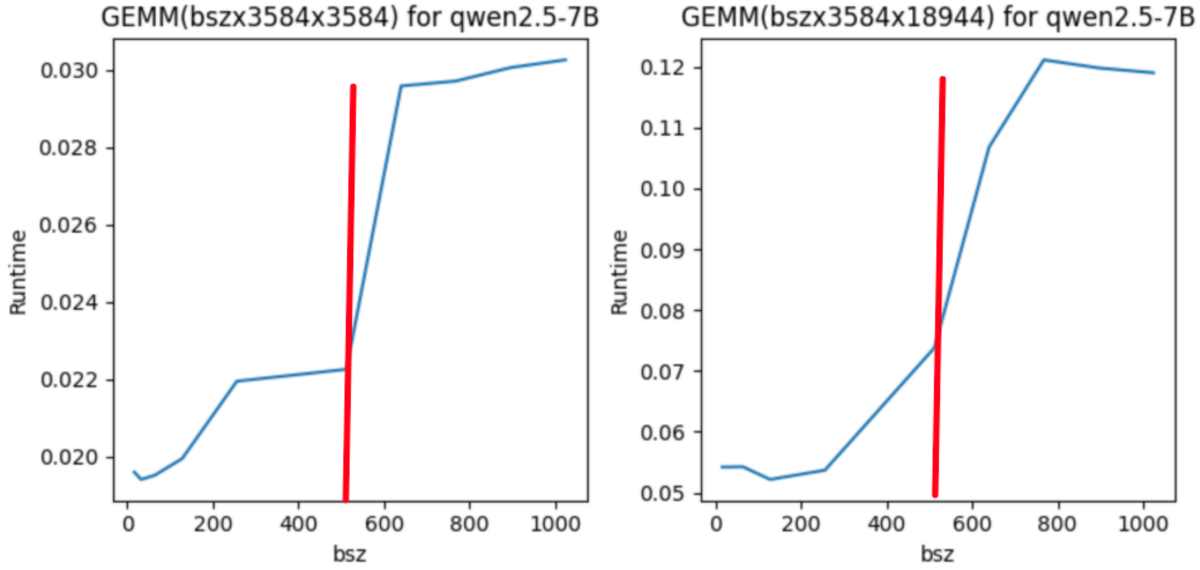
Motivation: Long Generation is Memory-bound

KV cache limits batch size, leaving substantial compute headroom

- **Long length limits the batch size, wasting computation.**
  - **7B on 2×H100-80GB**: ~1M-token KV capacity → only **~64 concurrent 16K sequences**
  - **32B on 8×H100-80GB**: ~1.9M-token KV capacity → only **~128 concurrent 16K sequences**
- **Potential:** utilize the wasted computation resources



H100 utilization during AIME decoding with Qwen-7B

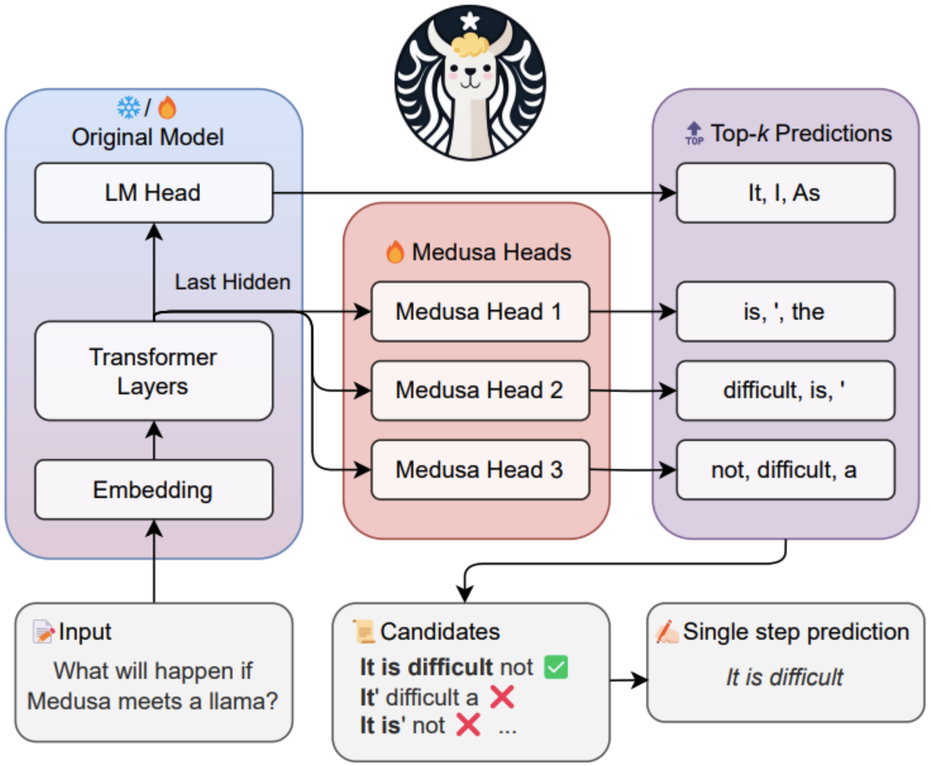


CUTLASS GEMM Runtime on H100 with Qwen-7B

Prior Speculative Decoding

Most SD methods either need a small draft model or fine-tuning.

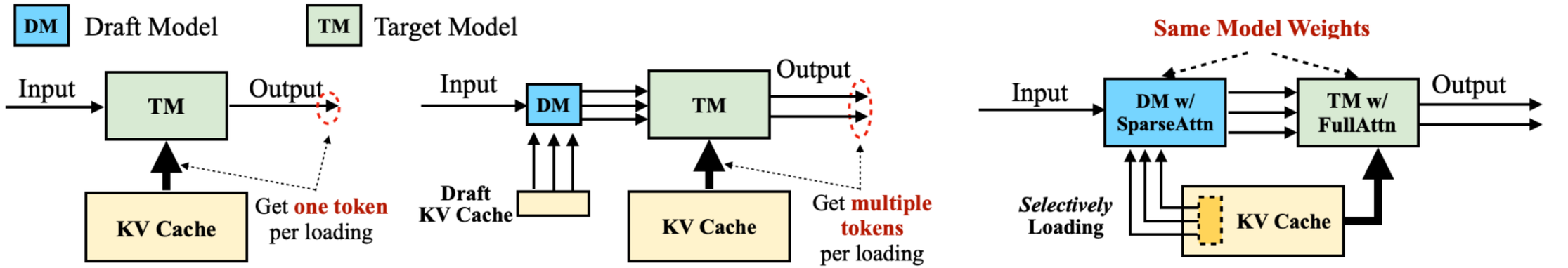
- **System Complexity**: Many SD methods require maintaining an extra draft model or auxiliary prediction heads.
- **Overhead**: Auxiliary modules often require fine-tuning or frequent updates during training.
- **Plug-and-play usability**: Cannot function in a plug-and-play way.



Solution

Self-speculative Decoding with Sparse Attention

- We can use the **same model weights with sparse attention as cheap speculation**.
- **Solution:** Self-speculative decoding with **sparse attention** as the **self-draft model**
  - **Lossless:** Speculative decoding.
  - **W/o overhead:** Self-speculative decoding without a draft model.
  - **Plug-and-play:** Sparse attention algorithm is training-free.
  - **Efficient:** Sparse attention is efficient and accurate for long-generation.

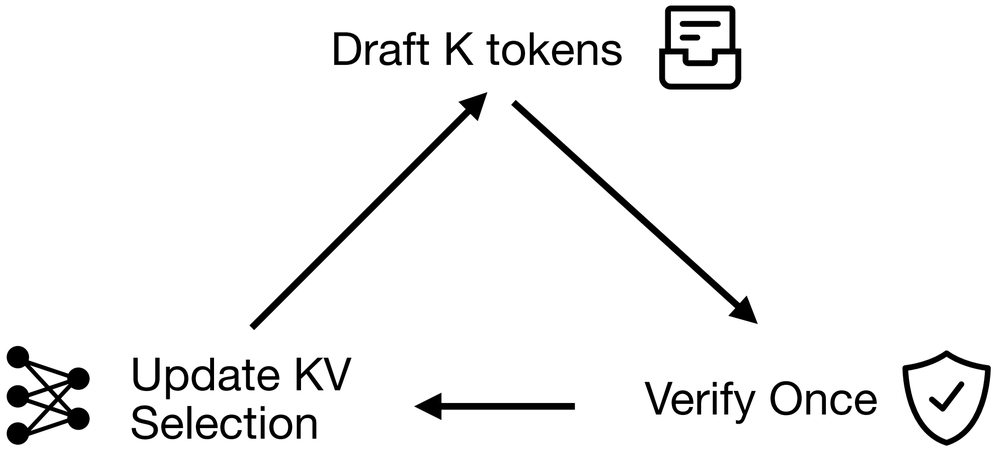


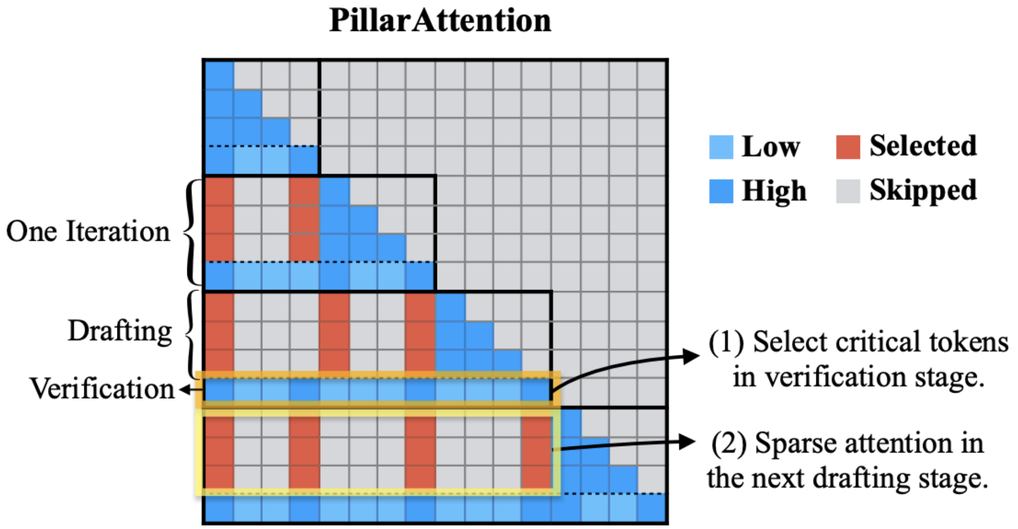
(a) Auto-regressive generation (b) Draft-model based speculative decoding (c) Ours (self-speculative decoding with sparse attention)

Algorithm Overview

Self-speculative Decoding with Sparse Attention

- **Draft cheaply with sparse attention, verify losslessly with full attention.**
  - **Draft** K tokens with **Sparse Attention**.
  - **Verify** the draft tokens using **Full Attention**.
  - **Pillar Attention:** Select the **Top-K** important KV tokens **by reusing the verification attention scores** for the next draft iteration.
  - Go back to step 1 until EoS is generated.





Key Takeaways

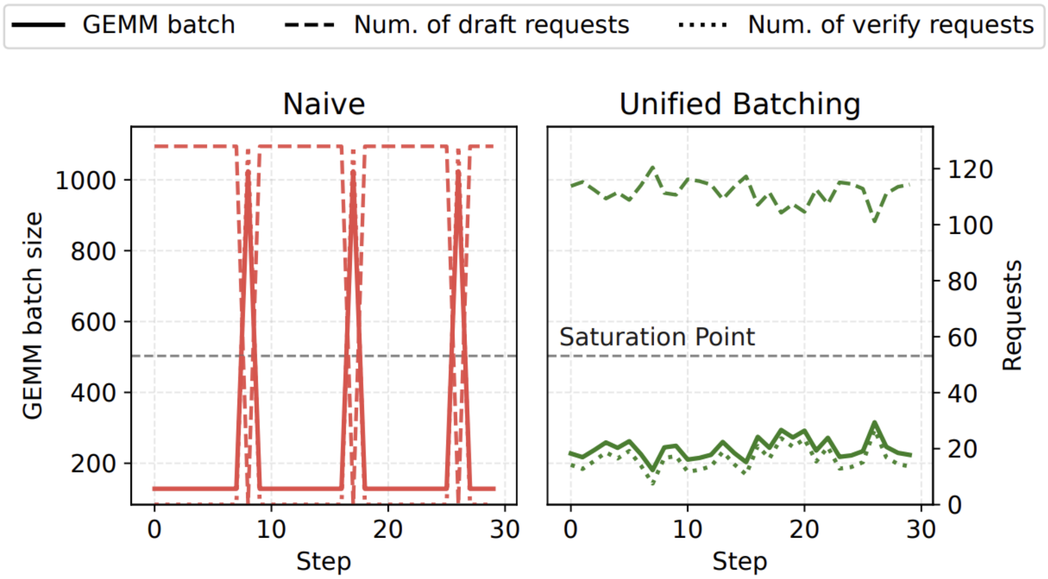
Self-speculative Decoding with Sparse Attention

- **Long generation is memory-capacity constrained:**
  - KV cache limits batch size before GEMM reaches saturation.
  - Example: **7B on 2×H100-80GB** supports ~1M KV tokens → only **~64 sequences at 16K average length**.
- **Sparse Speculative decoding increases effective GEMM batch size, but by <2×:**
  - **Effective batch size: (K + K) / (K + 1) * bsz < 2 * bsz.**
  - As long as the original batch is below GEMM saturation, this extra compute can be utilized.
- **In long-generation workloads, sparse self-speculation can turn underutilized compute into end-to-end speedup.**

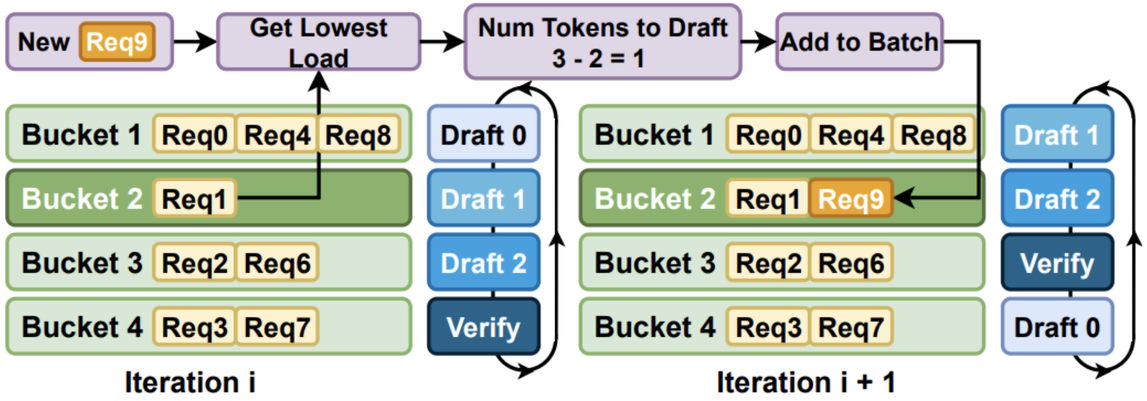
System Challenge: Scheduling Draft and Verification

Naïve scheduling synchronizes verification and causes GEMM saturation

- **Challenge**: If all requests are launched together, they enter the verification stage ***synchronously***
  - Verification has a larger effective GEMM batch (K+1 tokens) than drafting (1 tokens).
  - This creates bursty GEMM demand and can exceed the saturation point.
- **Unified batching**: scatter requests into **K+1 phase buckets**, so each batch **mixes K draft buckets with 1 verification bucket to keep GEMM demand stable**.



Visualization of resource usage with different scheduling policies

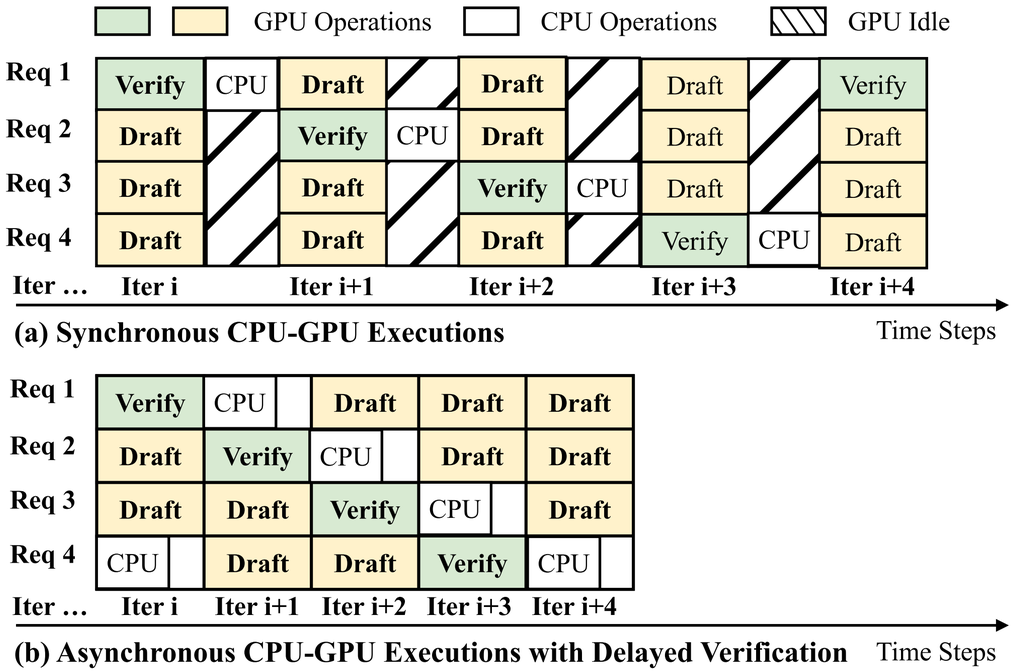


Min-loaded schedule: put request into the min-loaded phase

System Challenge: Hiding CPU Overhead

Draft depends on previous verification results, exposing CPU overhead

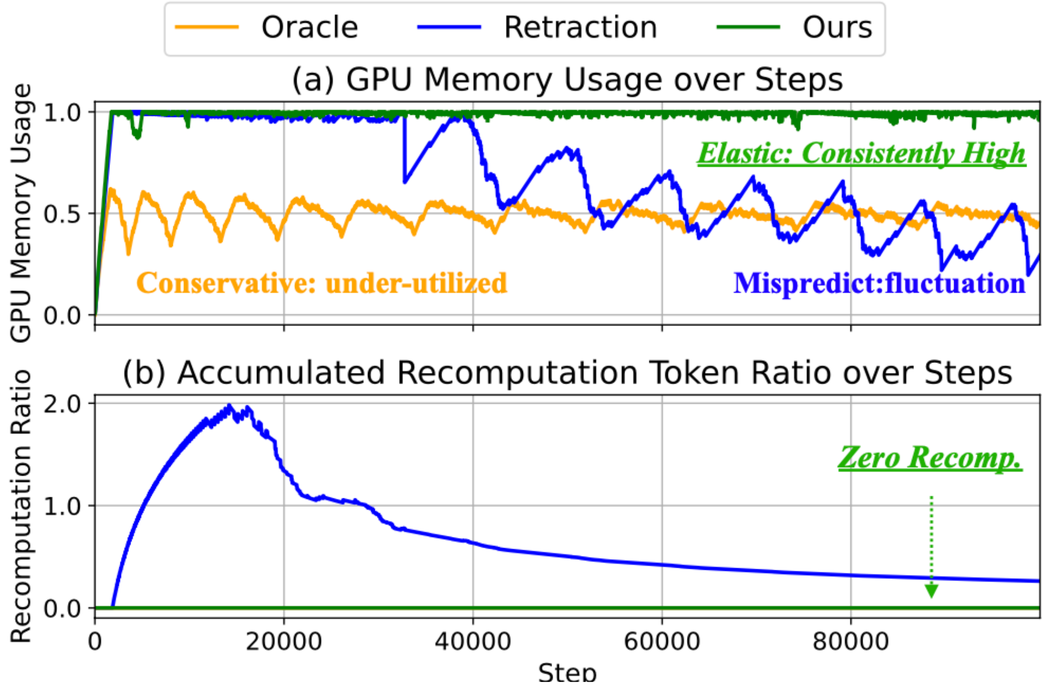
- **Challenge**: unlike standard LLM serving, self-speculative decoding has a **data dependency between verification and the next draft**.
- **Delayed verification:** after verification, **stall the request for one cycle before its next draft**.
  - CPU accept/reject and KV/sparsity updates run during the stalled cycle.
  - Stalling one cycle adds only 1/(K+1) overhead, which is small for typical K=8 or 16.



System Challenge: Memory Under-Utilization

In memory-bound decoding, KV capacity directly translates to throughput

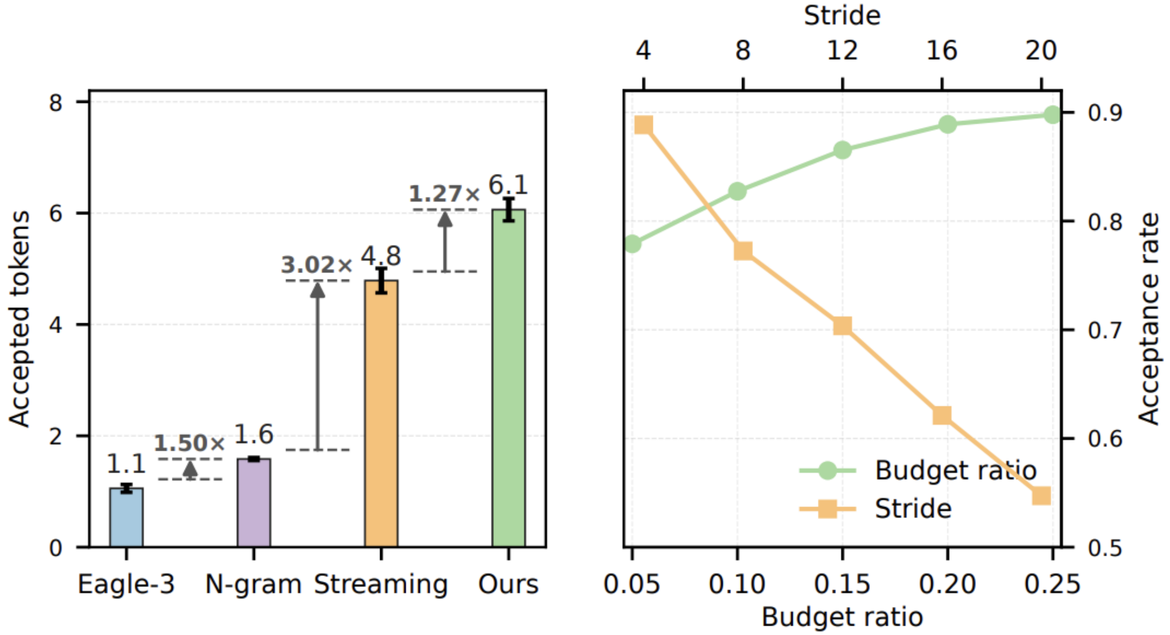
- **Problem**: GPU memory is conservatively reserved to avoid OOM, leaving usable KV capacity under-utilized.
- **Solution**: use KV-cache CPU offloading to safely pack more active
- **Policy**: **offload short requests first; restore long requests first** when memory allows.
- Our offloading policy **keep GPU KV usage near capacity**, enabling higher throughput **with zero recomputation**.



Acceptance Rate Evaluation

Pillar Attention improves draft quality

- Baselines: compare against common speculation policies (temp=0.6, w/o tree spec)
  - **Eagle-3**: drafting-model-based speculative
  - **N-gram**: training-free prompt lookup / n-gram speculation.
  - **Streaming**: sparse attention with fixed sink + recent tokens.
  - **Ours**: select by the top-k KV cache by the **attention scores from the previous iteration**.
- Result: Ours **accepts 6.1 tokens (out of 8 tokens) per verification**, outperforming baselines.

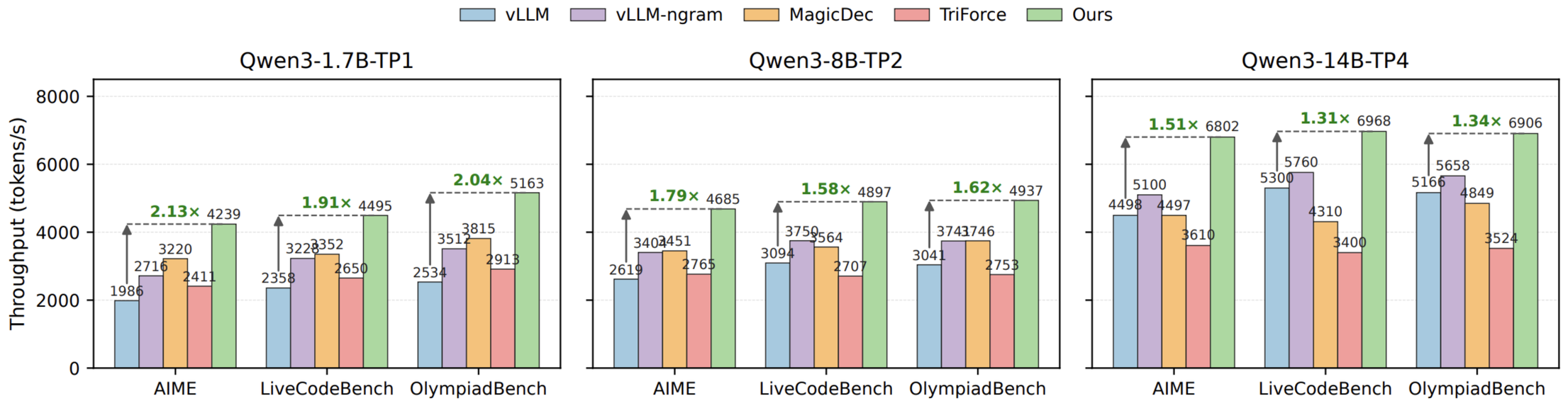


Average across [qwen3-1.7B, 8B, 14B] x [AIME, OlympaidBench, LiveCodeBench]

Efficiency Evaluation

End-to-end throughput across models and reasoning workloads

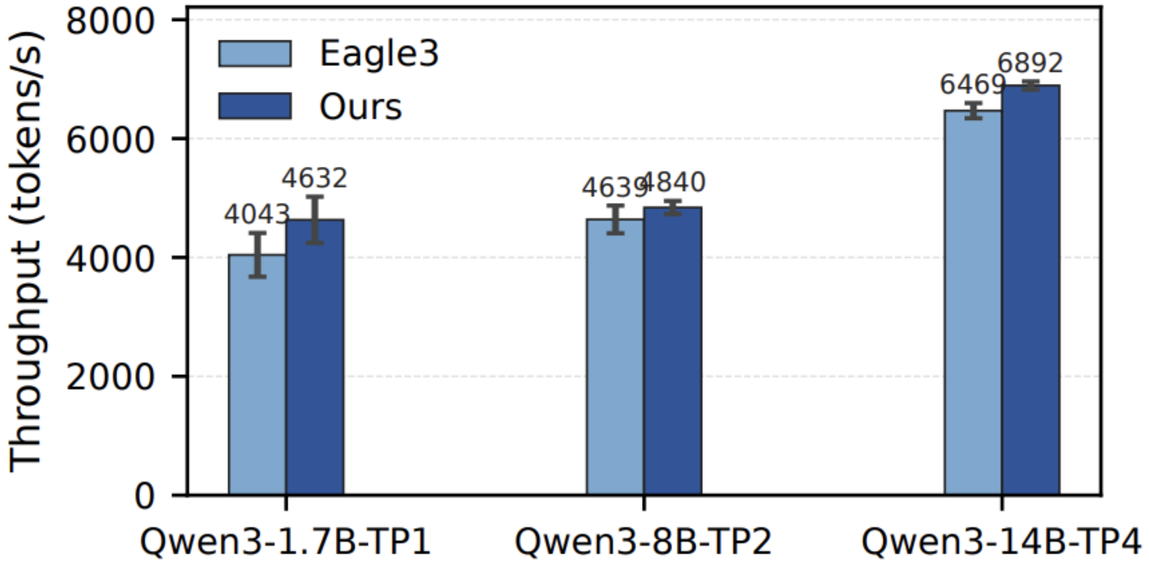
- **Setup:**
  - **Models:** Qwen3-1.7B (TP1), Qwen3-8B (TP2), and Qwen3-14B (TP4) on H100.
  - **Workloads:** AIME, LiveCodeBench, and OlympiadBench.
  - **Baselines:** vLLM/-NGram, TriForce, and MagicDec; with the best-performing speculative tokens
- **Results:**
  - **Up to 2.13x speedup over vLLM and outperforms baselines across models and workloads.**



Efficiency Evaluation

Efficiency vs. Draft-Model-Based Speculation

- **Setup:**
  - **Models:** Qwen3-1.7B (TP1), Qwen3-8B (TP2), and Qwen3-14B (TP4) on H100
  - **Workloads:** AIME, LiveCodeBench, and OlympiadBench
  - **Baselines:** Eagle3, a draft-model-based speculative decoding (w/ 3 draft tokens).
- Ours achieves **higher throughput than Eagle3 across all settings without separate draft model or fine-tuning**.



Conclusion

- **SparseSpec adopts sparse-attention as a draft model, achieving lossless and training-free acceleration.**
- **SparseSpec achieves 76.5% acceptance rate, achieving up to 2.13x speedup over baselines with several system co-designs.**

Q&A

- **Thank you!**
- **Contacts**:
  - yilongzhao@berkeley.edu
  - jmtang@mit.edu

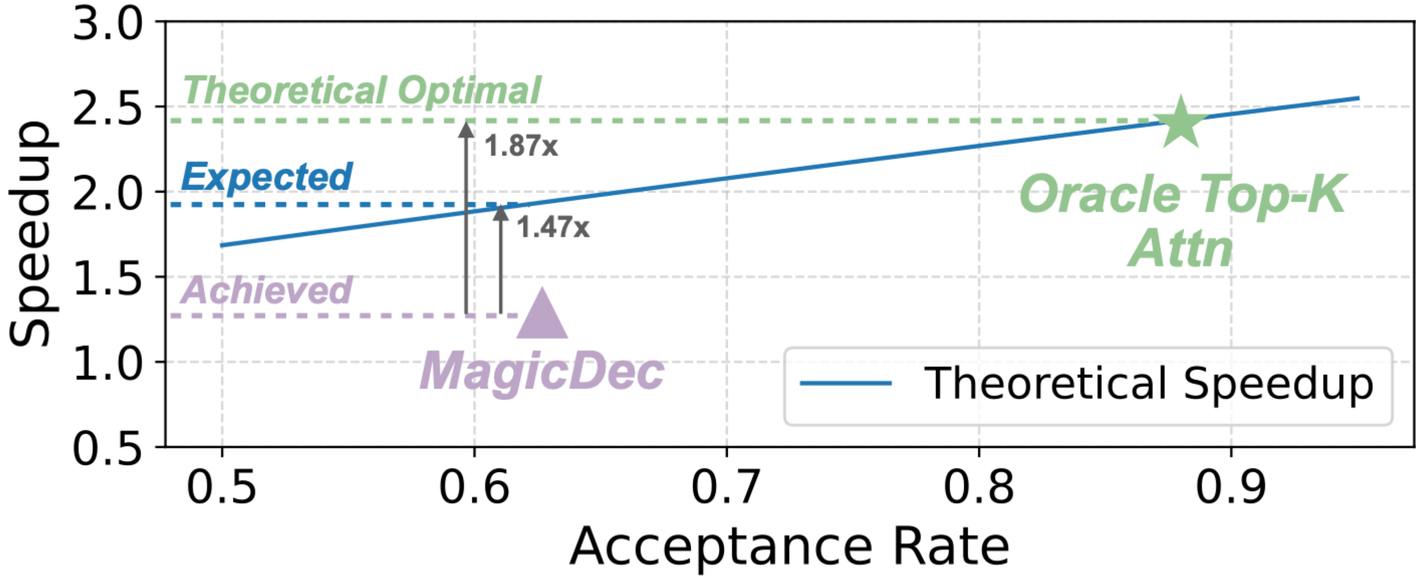
Paper

Code

Existing Self-Speculative Decoding

Naïve self-speculative decoding fails to achieve theoretical optimal speedup

- **Less accurate**: the acceptance rate is much worse than the oracle top-k attention
- **Less efficient:** assuming the same acceptance rate, the speedup is lower than expected



Ablation Study on System Optimizations

Each system optimization contributes to significant improvement

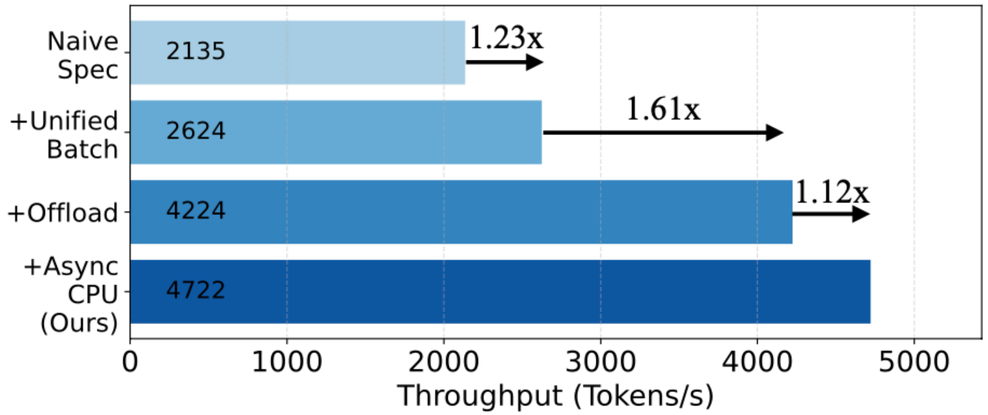


Figure 13. Ablation study of SpecGen on Qwen3-1.7B with AIME. Each component contributes to significant improvement.