A Unified Benchmark Suite for Retrieval-Augmented Speculative Decoding

Ada Ho, Tan Chien Hao, Lee Kwan Tze, Benn Tan

Background

Speculative decoding (SD) accelerates LLM inference by using a smaller model (drafter) to predict the next few tokens, then using a larger model (verifier) to verify all guesses in parallel. Recent work has explored allowing the drafter to retrieve from vector database as prior, giving rise to retrieval-augmented speculative decoding (RASD). While RASD has yielded promising results, lack of flexible configurability and the existence of multiple diverging techniques make it difficult to apply in practice. We propose **RASD-Bench**, a **plug-and-play software framework** for RASD that (a) allows convenient configuration of all parts of the RASD pipeline including drafter, verifier and vector database, and (b) reports useful metrics for evaluating throughput and generation quality.

Method

We adopt the RASD method proposed by RAPID [CFN25] because (a) it integrates with the SD functionality of Hugging Face Transformers, making production use simple; (b) it displays significant empirical improvements on long-context benchmarks LongBench v2 [BTZ24] and InfiniteBench [ZCH24], ensuring real-world usefulness.

We additionally make the following practical improvements:

- We port all evaluation code and config options from a standalone script to an importable RASDEngine class, enabling flexible configuration and interop with existing applications.
- During prefill, we exclude the output MLP from the forward pass, cutting VRAM usage by about one-third relative to RAPID on sequences >100k tokens long.
- We provide an RASDBenchmark class that abstracts away the complexity of evaluating an RASD pipeline on a custom dataset. The user needs only provide a sequence of samples, utilities for RAG context extraction, and a scoring function.

Evaluation

We evaluate our revised framework on RAPID's original datasets LongBench v2 and InfiniteBench, as well as an additional dataset Ada-LEval [WDZ24] consisting of two subtasks StackSelect and TextSort. StackSelect involves picking the most relevant answer to a StackOverflow question from up to 1000 candidates, while TextSort requires the model to output the correct order of four randomly permuted chapters from some contiguous section of a book.

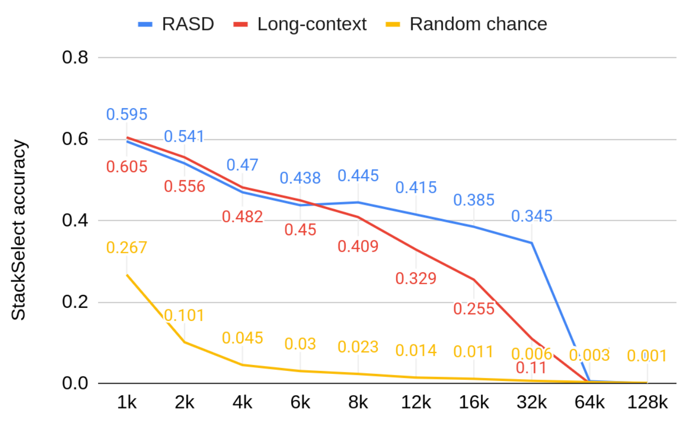
We choose Ada-LEval for the following reasons:

- **Deterministic evaluability.** Many benchmarks rely on LLM-as-judge, which our cost constraints preclude. Enforcing output format also helps evaluate instruction-following ability.
- **Diverse context lengths.** RAPID authors only plot performance gain against context length on a single dataset. Ada-LEval is explicitly split by context length, allowing us to check that their trend generalizes.
- **Diverse subtask difficulties.** StackSelect subtask is of comparable difficulty to InfiniteBench and LongBench, allowing 'in-distribution' validation of RAPID's results. TextSort subtask is extremely challenging—even frontier LLMs do no better than random chance. RAPID authors don't test their technique on such tasks; we fill that gap.

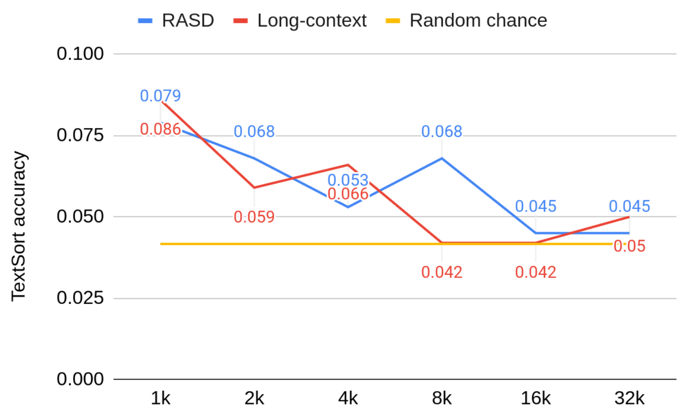
For all evaluations, we use Qwen2.5-7B-Instruct as both drafter and verifier, with a max context length of 98304 enforced through middle truncation.

Results

We reproduce RAPID's gains on LongBench v2 and InfiniteBench. Additionally, we find that they generalize to the StackSelect subtask of Ada-LEval, with the performance delta relative to vanilla long-context inference increasing remarkably with context length. A performance drop is observed above 32k tokens, likely because that is the default context window of Qwen2.5-7B. (Though the model does support YaRN for context window extension, we do not enable it in order to maintain methodological parity with RAPID.)



On the other hand, improvement on TextSort is marginal if at all existent. [WDZ24] hypothesize that the task is inherently too challenging for current LLMs at this scale; we show that this holds even when context is RAG-compressed. Without more specialized algorithms to tackle this task (e.g. issuing multiple retriever queries for *each* passage, then performing pairwise matching), TextSort remains a challenge even with RASD.



Limitations and possible extensions

Integration with LLM serving frameworks. Given the clear utility of RASD, we believe that integration with LLM serving frameworks (e.g. vLLM [KLZ23] and SGLang [ZYX23]) is a worthwhile next step. RASD-Bench proves that RASD can be made production-ready from a software engineering perspective; serving optimizations such as torch.compile, CUDA graphs, in-flight batching and asynchronous vector store queries will almost certainly make RASD sufficiently performant for real-world use.

Larger models and longer contexts. Compute constraints prevented us from evaluating RASD-Bench on larger models as either drafter or verifier. While [CFN25] prove RASD effective for at least two models with ~70B parameters, it might be worth evaluating on more models of such size, and perhaps even larger ones at 200B+ parameter scale.

RASD with reasoning models. [CFN25] show that RASD performance gains persist with CoT, consistently outperforming both non-CoT RASD and non-RASD CoT. With the advent of reasoning models specifically trained through RL to think step-by-step, we expect that RASD may synergize with reasoning capability to enhance accuracy on tasks requiring highly complex long-context understanding, such as TextSort.

References

[CFN25] RAPID: Long-Context Inference with Retrieval-Augmented Speculative Decoding

[BTZ24] LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks

[ZCH24] \inftyBench: Extending Long Context Evaluation Beyond 100K Tokens

[WDZ24] Ada-LEval: Evaluating long-context LLMs with length-adaptable benchmarks

[KLZ23] Efficient Memory Management for Large Language Model Serving with PagedAttention

[ZYX23] SGLang: Efficient Execution of Structured Language Model Programs