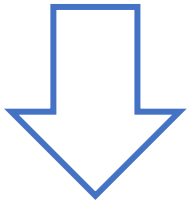
Stream2LLM: Overlap Context Streaming and Prefill for Reduced Time-to-First-Token

Rajveer Bachkaniwala, Chengqi Luo, Richard So, Divya Mahajan, Kexin Rong

Georgia Institute of Technology

Georgia Tech

LLMs benefit from context retrieval



parametric knowledge

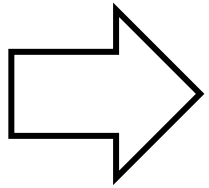
LLMs benefit from context retrieval

External Knowledge Source

- Dense retrieval (e.g., vector search)
- Sparse retrieval (e.g., BM25)
- Tool use (e.g., web crawling)
- …

parametric knowledge

I have a question…



Retrieval

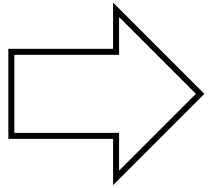
LLMs benefit from context retrieval

External Knowledge Source

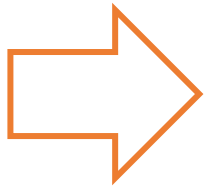
- Dense retrieval (e.g., vector search)
- Sparse retrieval (e.g., BM25)
- Tool use (e.g., web crawling)
- …

parametric knowledge

I have a question…

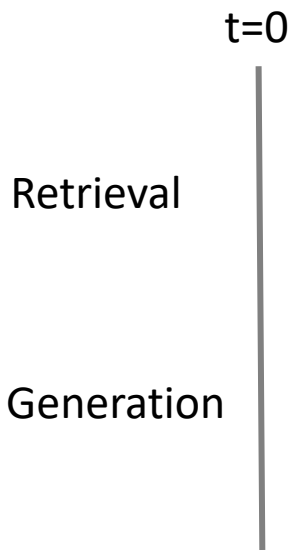


Retrieval



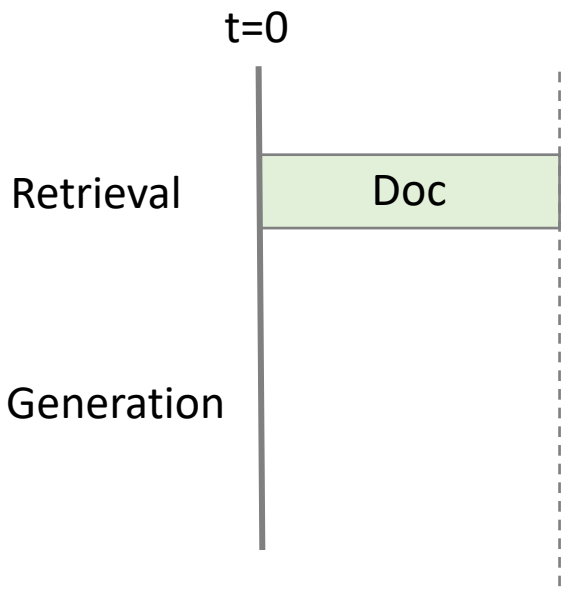
External Context

Context retrieval hurts request latency



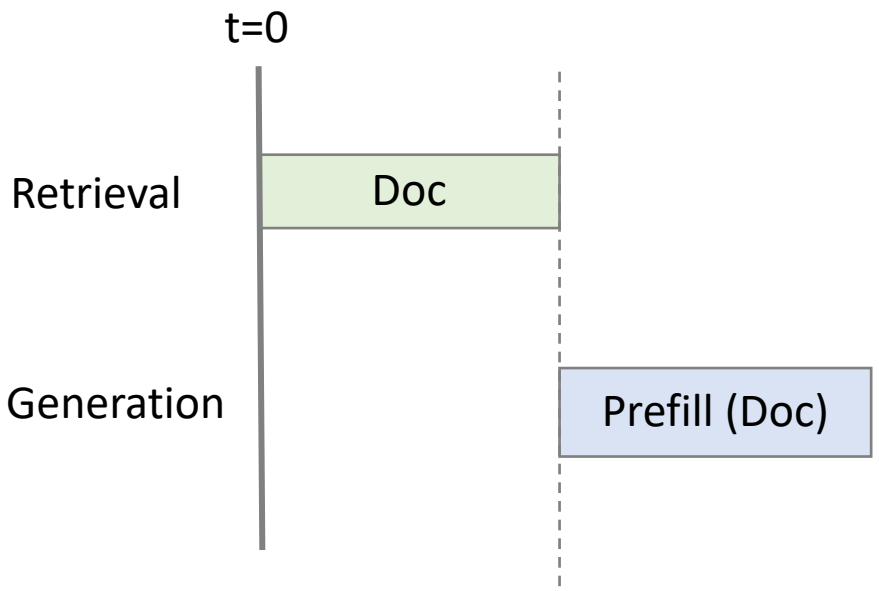
Context retrieval hurts request latency

Retrieval latency



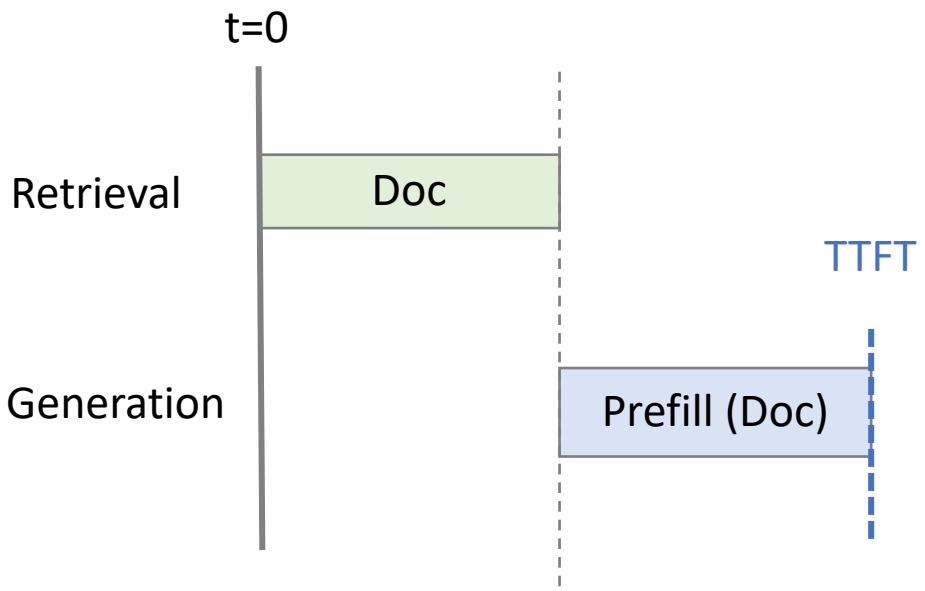
Context retrieval hurts request latency

Retrieval latency + *Prefill latency*



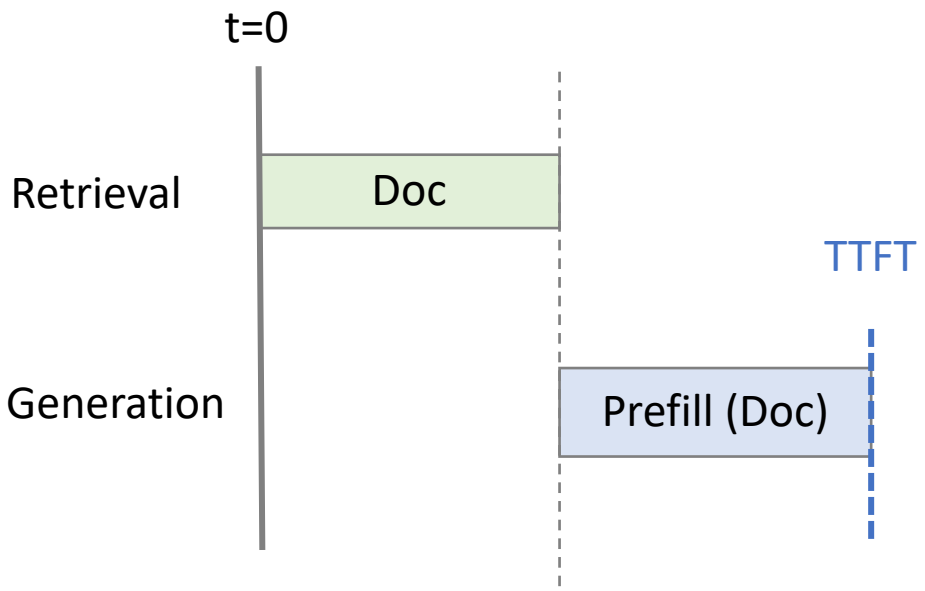
Context retrieval hurts request latency

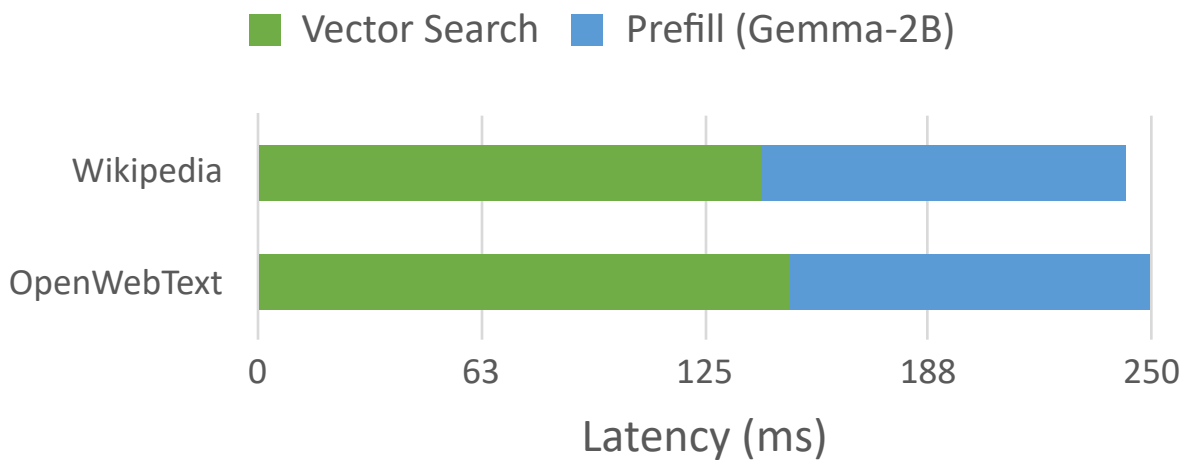
Time to First Token = Retrieval latency + Prefill latency



Context retrieval hurts request latency

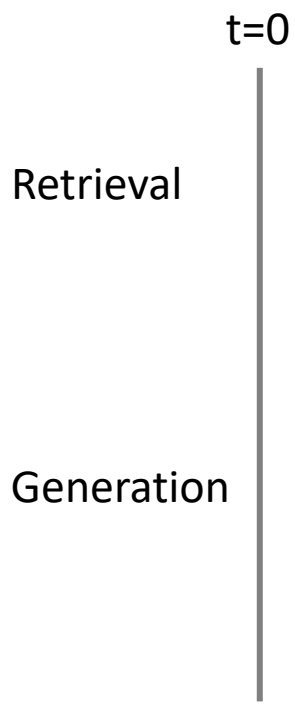
Time to First Token = Retrieval latency + Prefill latency





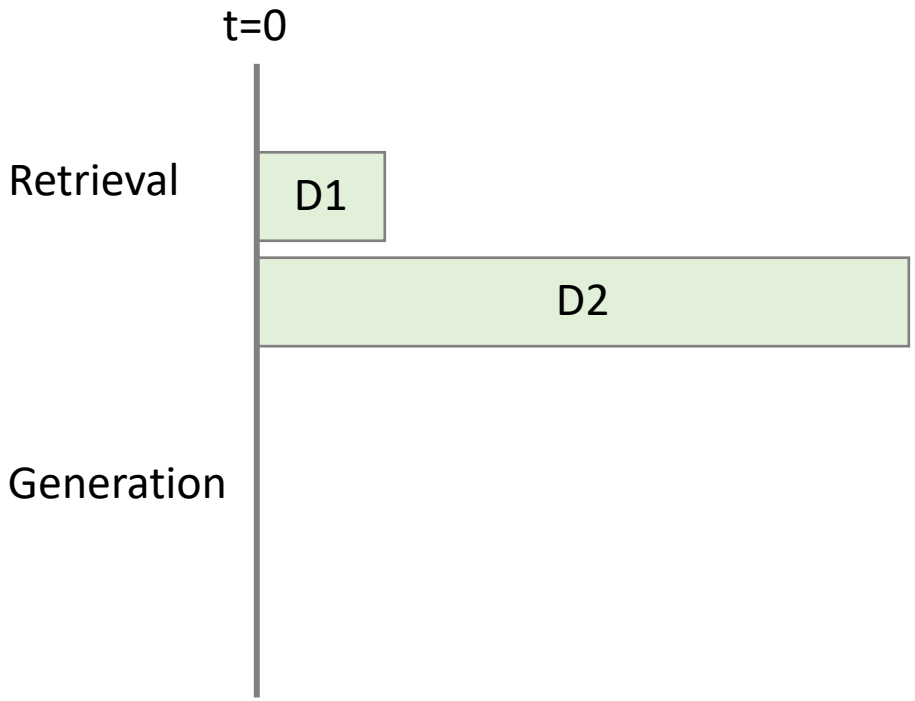
Stream2LLM: Streaming LLM Inference

Goal: Overlap retrieval and prefill to reduce TTFT



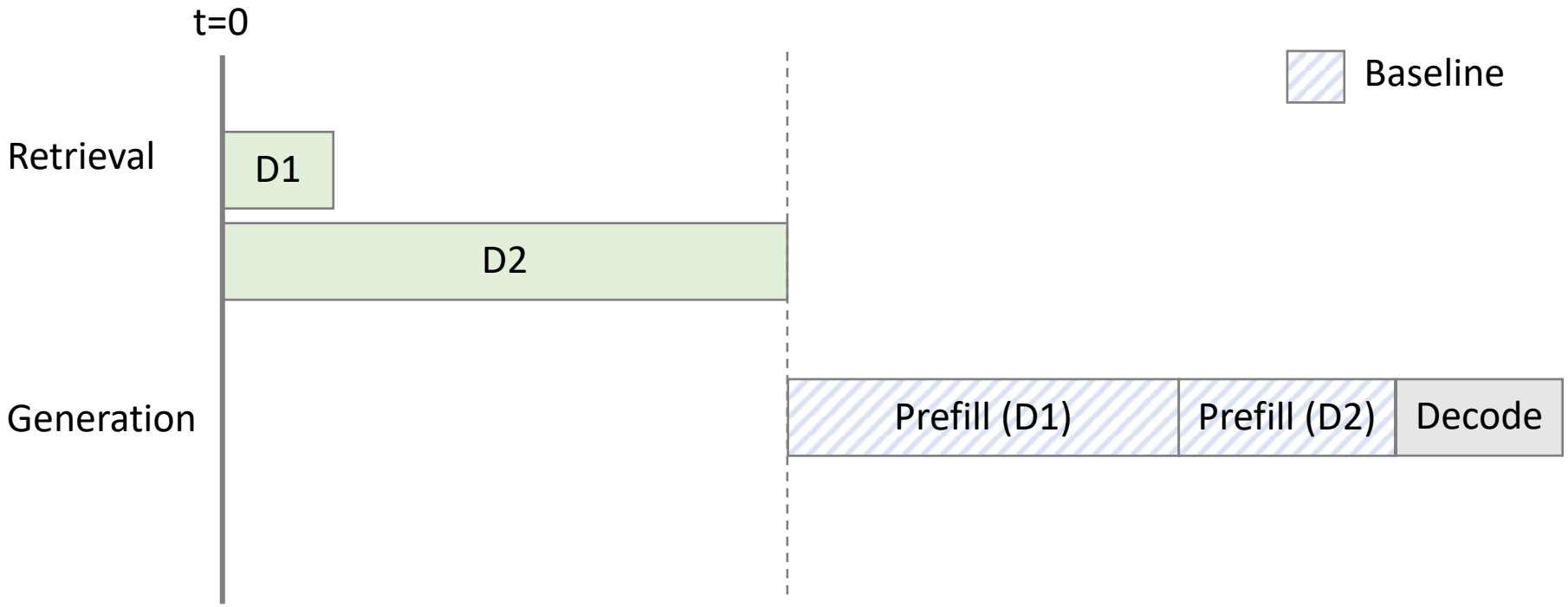
Stream2LLM: Streaming LLM Inference

Goal: Overlap retrieval and prefill to reduce TTFT



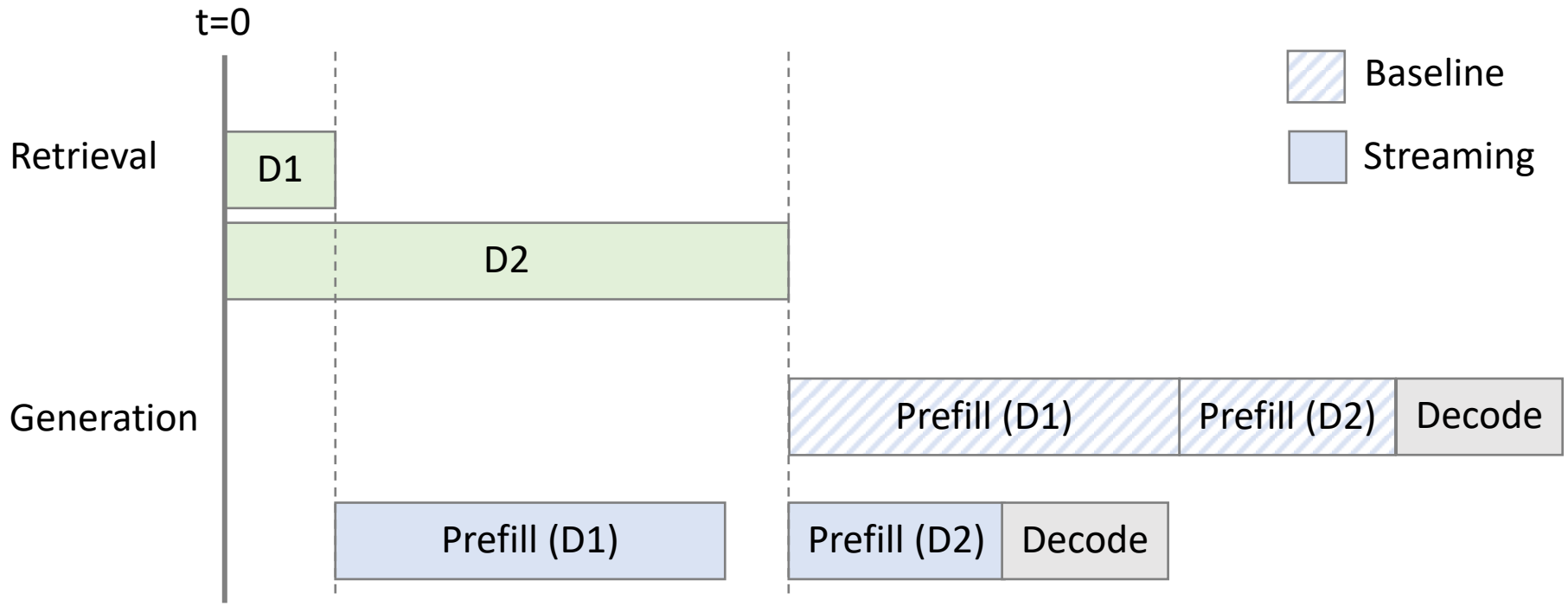
Stream2LLM: Streaming LLM Inference

Goal: Overlap retrieval and prefill to reduce TTFT



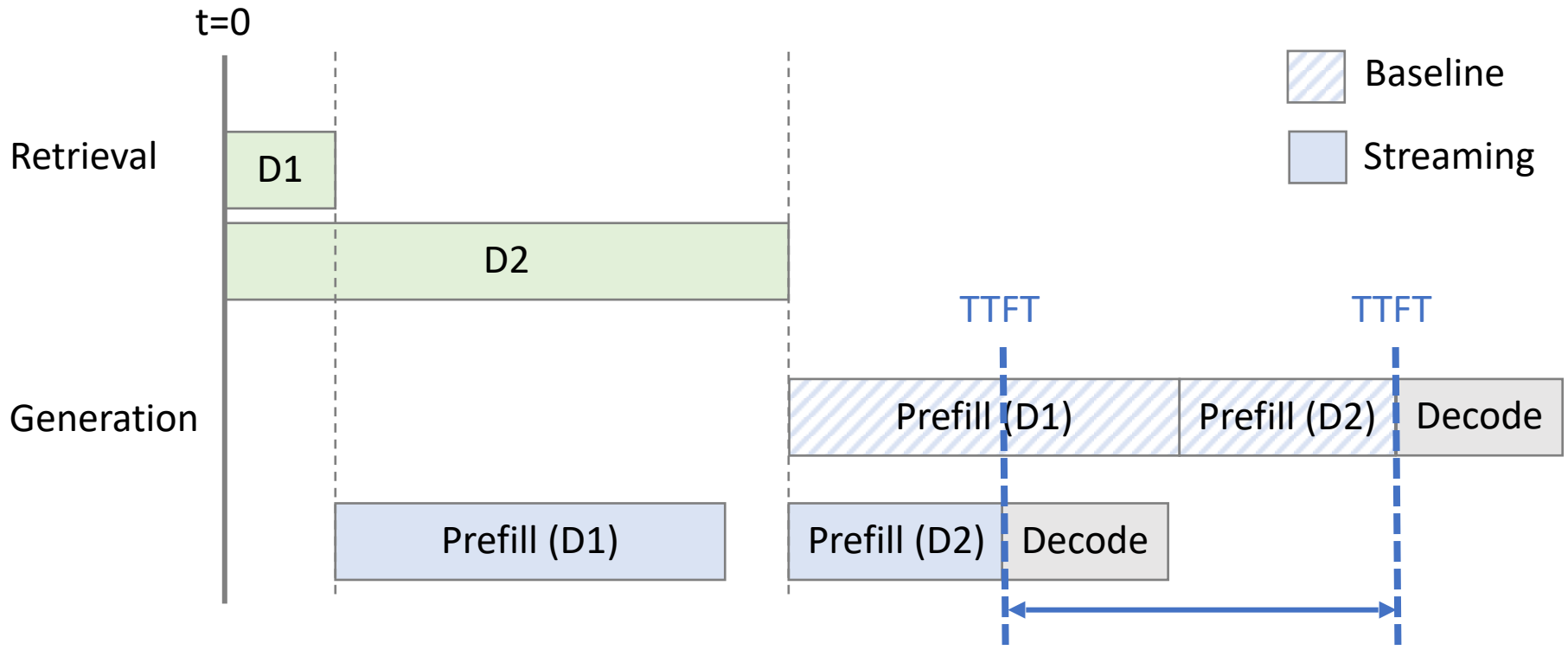
Stream2LLM: Streaming LLM Inference

Goal: Overlap retrieval and prefill to reduce TTFT



Stream2LLM: Streaming LLM Inference

Goal: Overlap retrieval and prefill to reduce TTFT

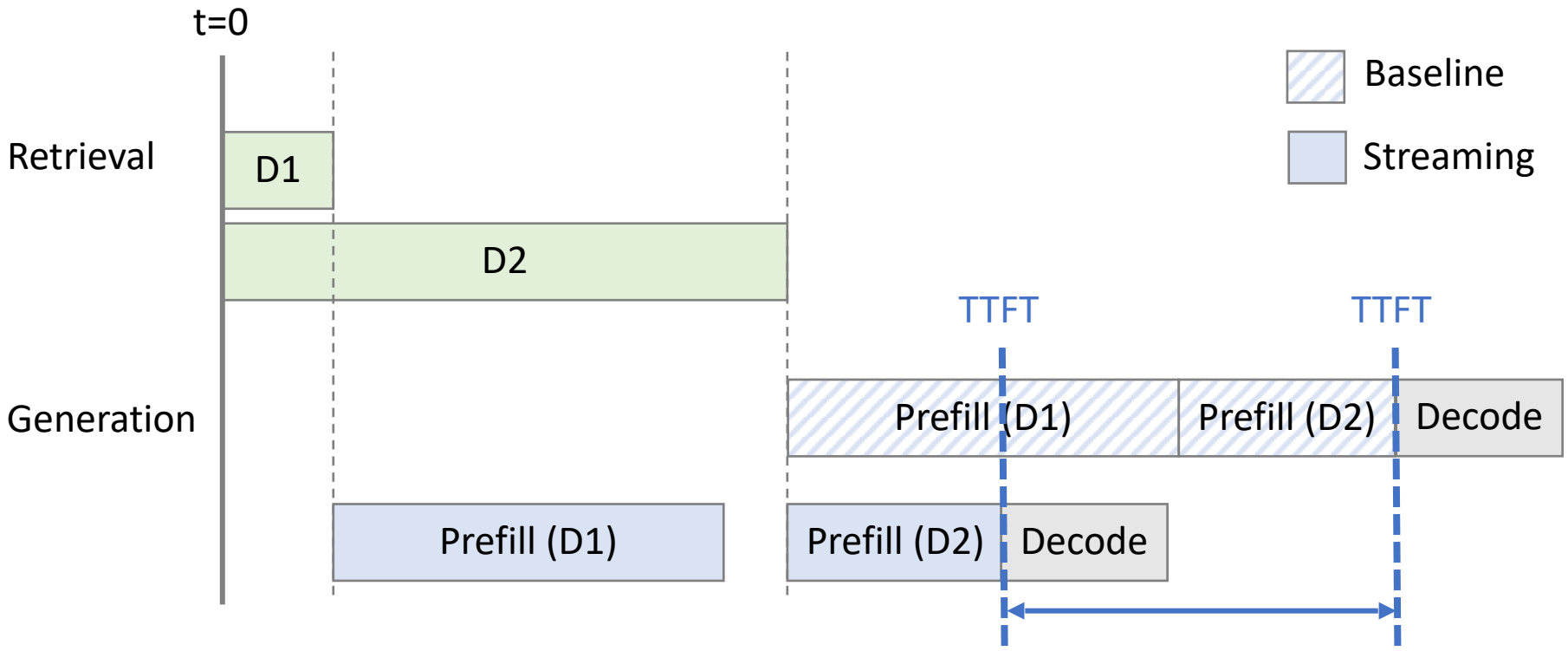


Stream2LLM: Streaming LLM Inference

Goal: Overlap retrieval and prefill to reduce TTFT

Deployment setting: prefill instances in P/D disaggregation

* Implemented on top of vLLM



Challenges with Context Streaming

Each iteration, the scheduler allocates compute (token budget) and memory (KV cache) across requests without knowing:

- Input size
- Finish time
- Chunk arrival pattern

Challenges with Context Streaming

Each iteration, the scheduler allocates compute (token budget) and memory (KV cache) across requests without knowing:

- Input size
- Finish time
- Chunk arrival pattern

Prior work:

- Single-request streaming: e.g., PipeRAG, AquaPipe
- Serving with static inputs: e.g., vLLM, Sarathi-serve

Challenges with Context Streaming

Each iteration, the scheduler allocates compute (token budget) and memory (KV cache) across requests without knowing:

- Input size
- Finish time
- Chunk arrival pattern

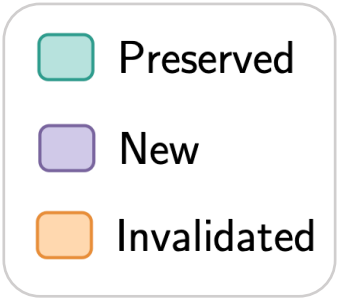
Prior work:

- Single-request streaming: e.g., PipeRAG, AquaPipe
- Serving with static inputs: e.g., vLLM, Sarathi-serve

Gap: multi-tenant, streaming inference.

Characterize Retrieval Workloads

Two distinct behavior modes:



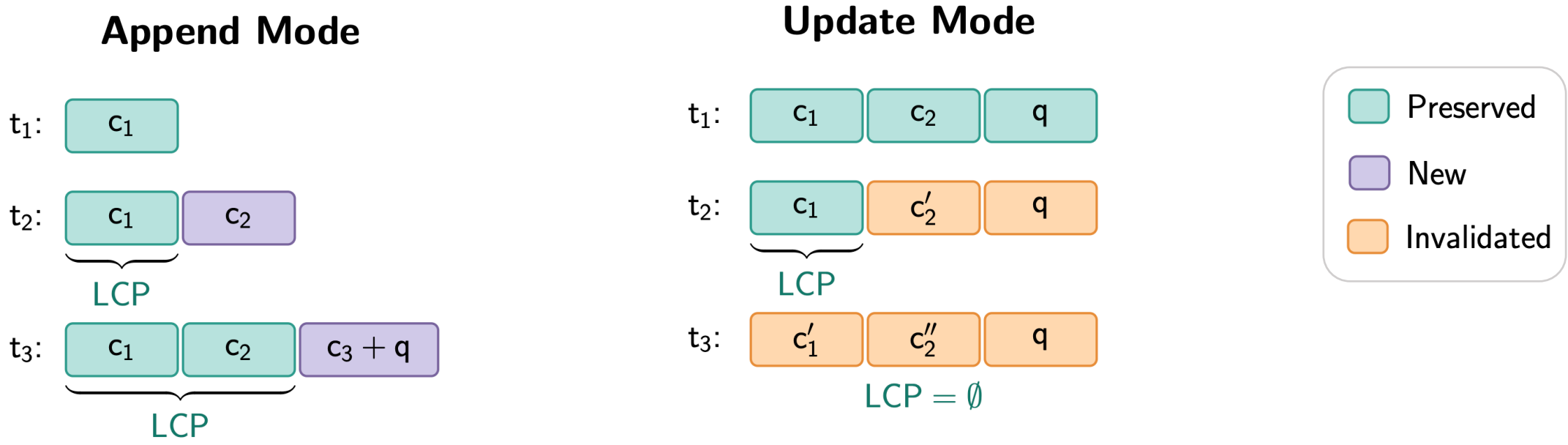
Characterize Retrieval Workloads

Two distinct behavior modes:

- Progressive accumulation (e.g., web crawler)
- Cache grows incrementally

Characterize Retrieval Workloads

Two distinct behavior modes:



- Progressive accumulation (e.g., web crawler)
- Cache grows incrementally

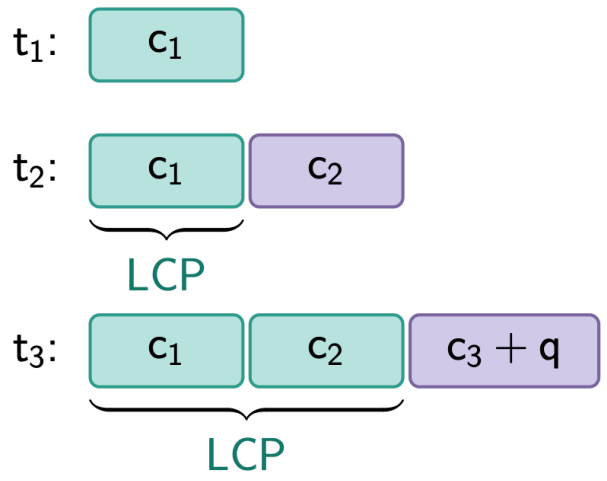
- Iterative refinement (e.g., vector search)
- Update could invalidate cache

Characterize Retrieval Workloads

Two distinct behavior modes:

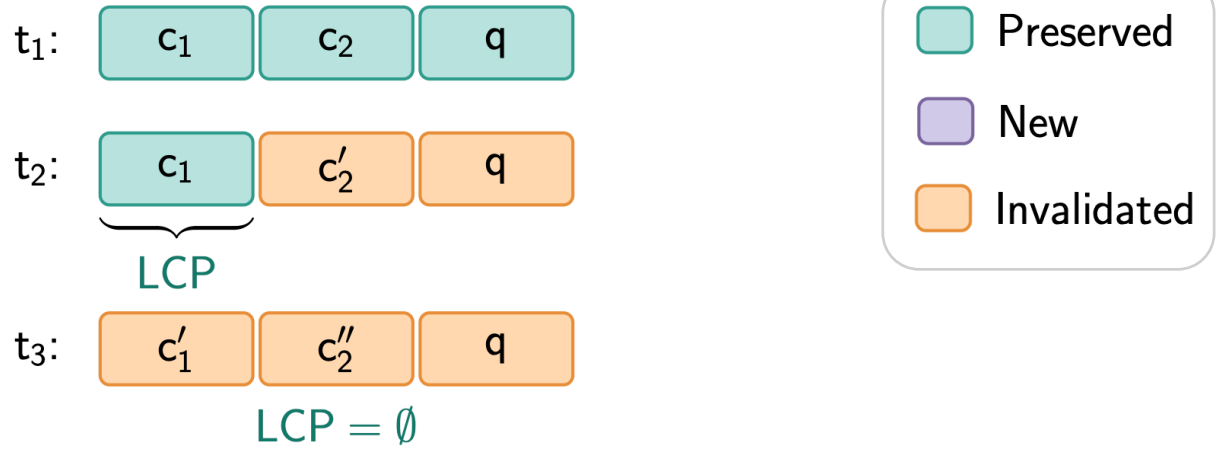
Require different scheduling and cache management mechanisms

Append Mode



- Progressive accumulation (e.g., web crawler)
- Cache grows incrementally

Update Mode

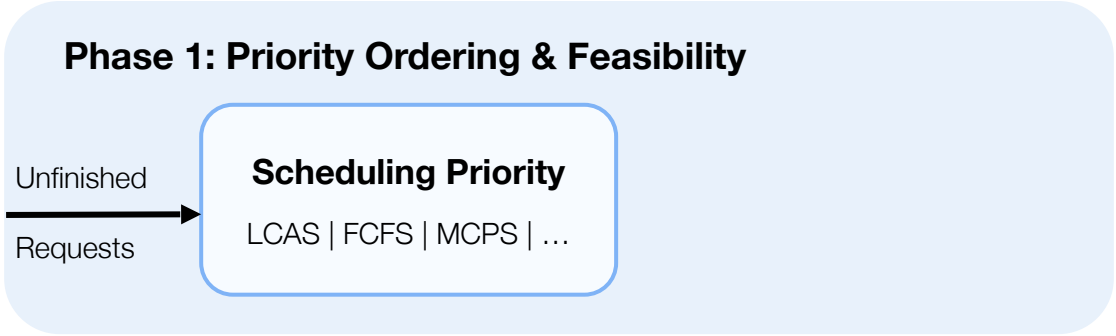


- Iterative refinement (e.g., vector search)
- Update could invalidate cache

Two-Phase Scheduling

Phase 1: What to run

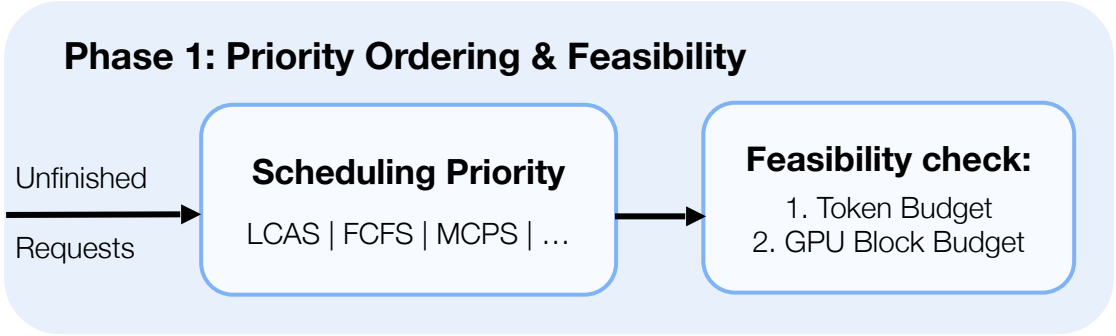
Select requests based on priority; check feasibility



Two-Phase Scheduling

Phase 1: What to run

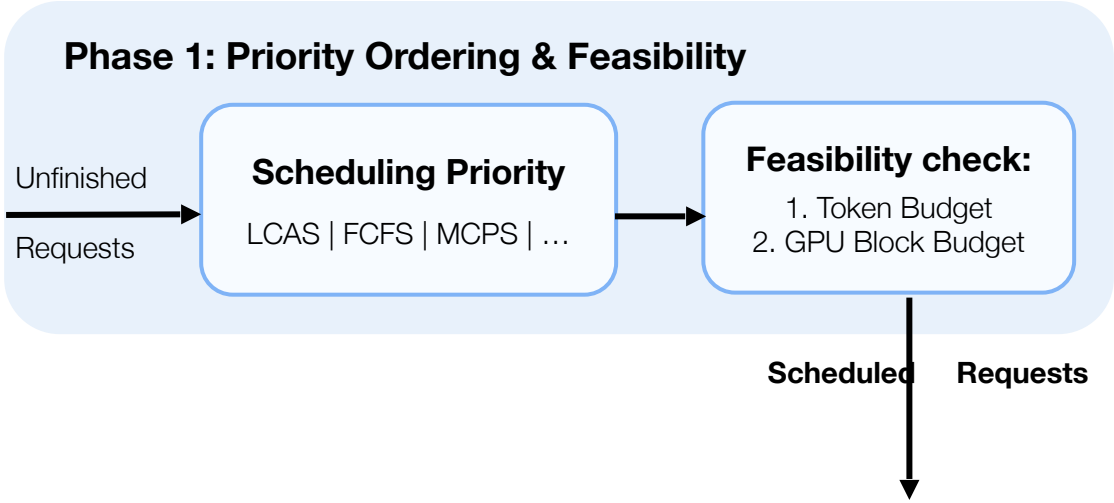
Select requests based on priority; check feasibility



Two-Phase Scheduling

Phase 1: What to run

Select requests based on priority; check feasibility



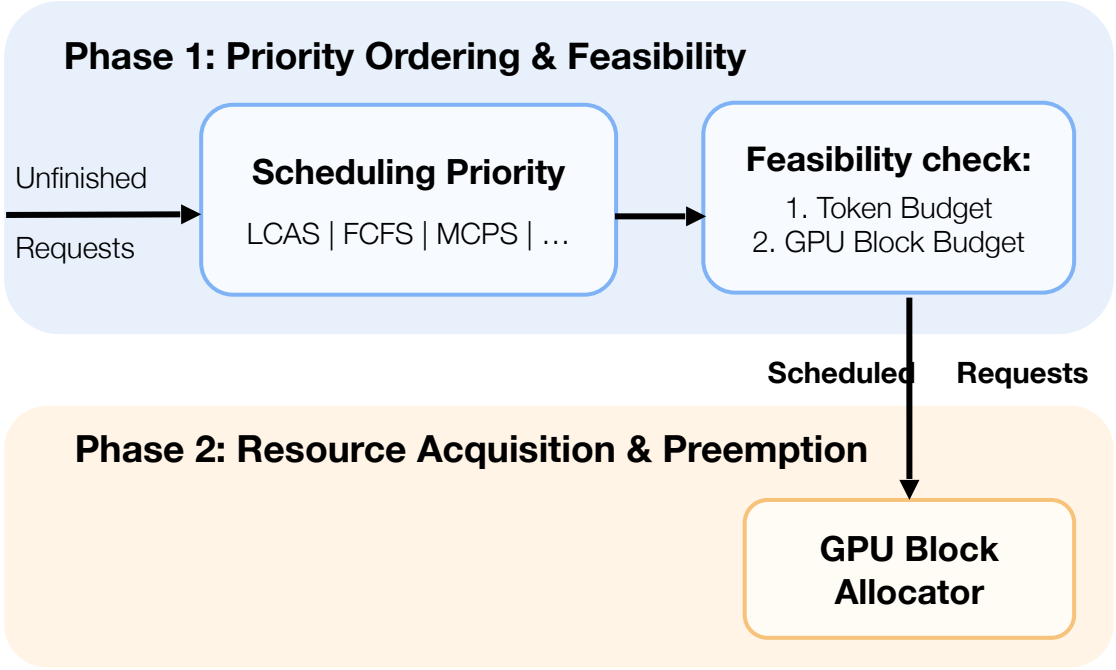
Two-Phase Scheduling

Phase 1: What to run

Select requests based on priority; check feasibility

Phase 2: How to allocate

If allocation fails: preempt lowest-priority running request;



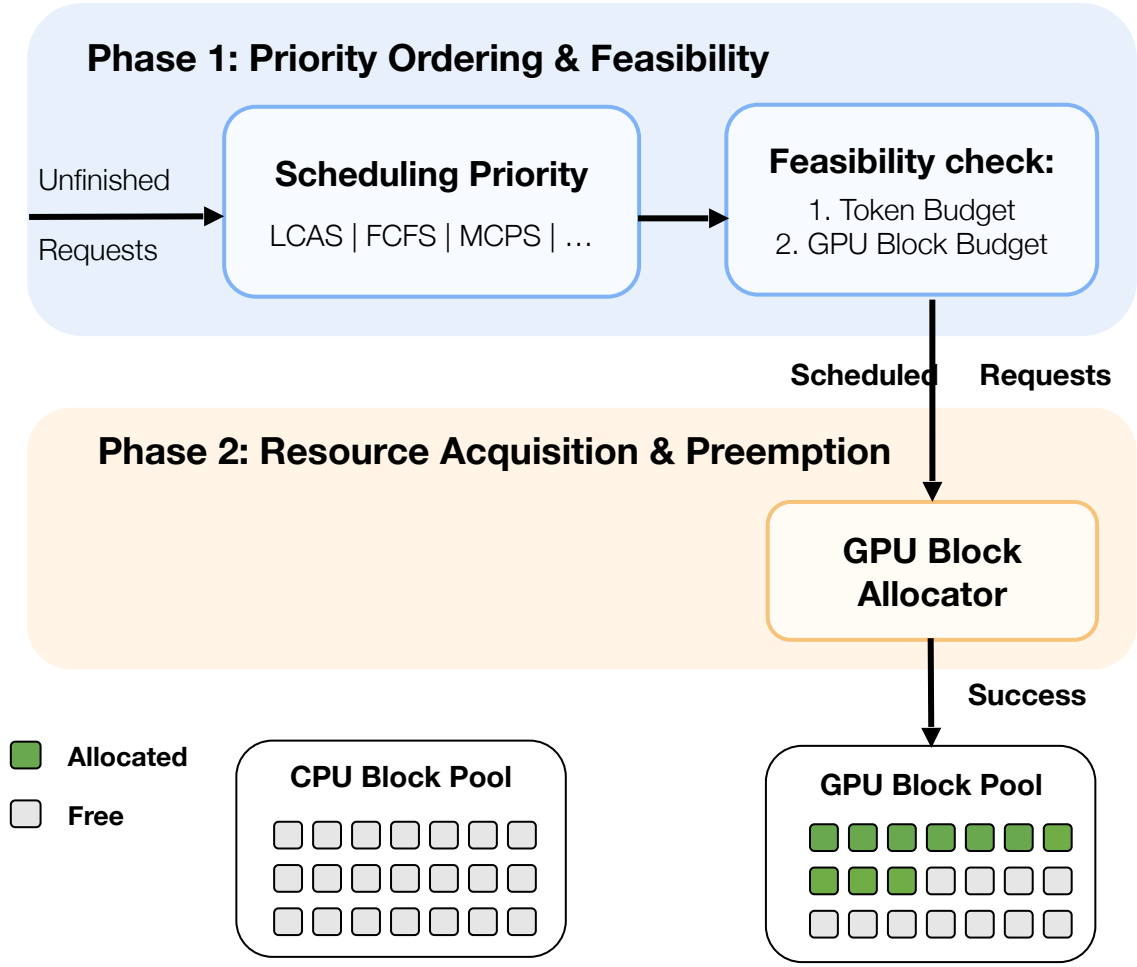
Two-Phase Scheduling

Phase 1: What to run

Select requests based on priority; check feasibility

Phase 2: How to allocate

If allocation fails: preempt lowest-priority running request;



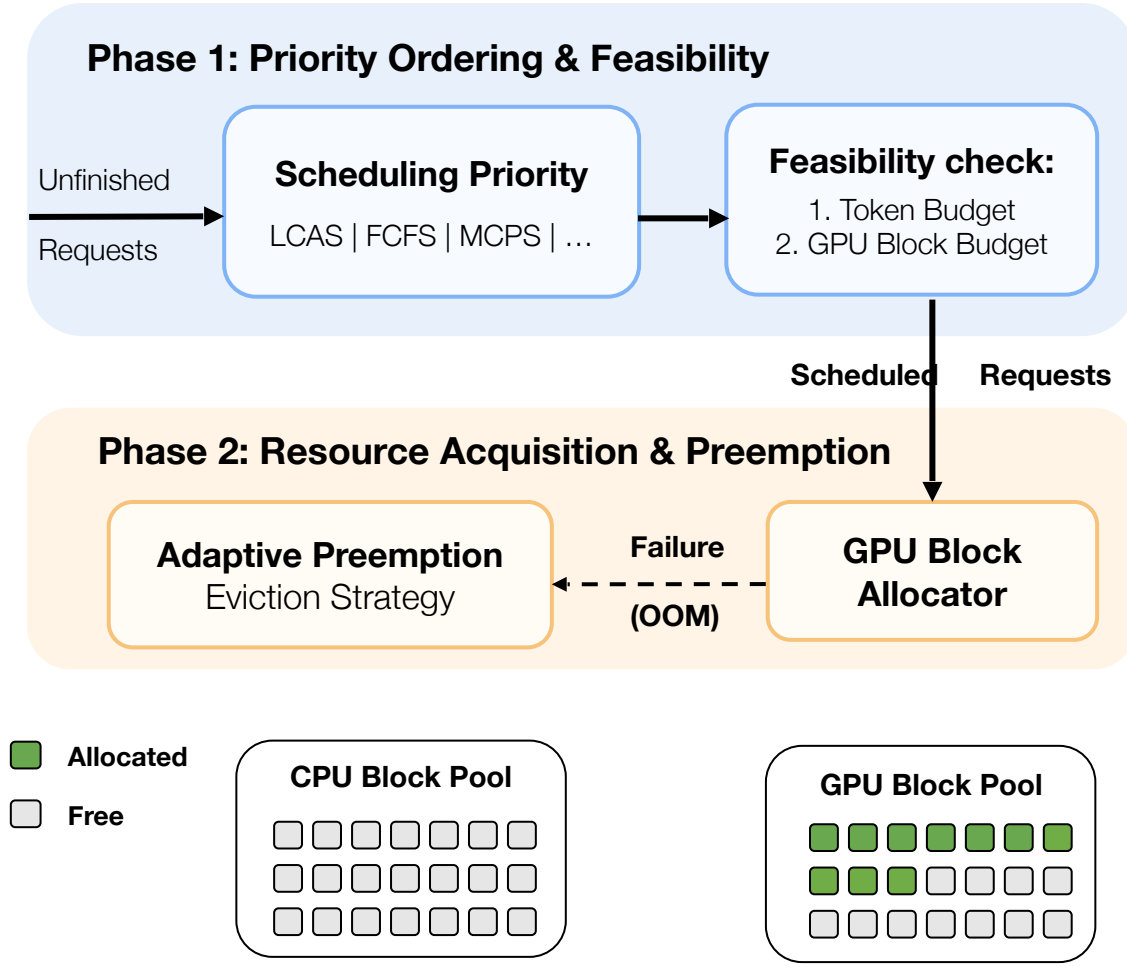
Two-Phase Scheduling

Phase 1: What to run

Select requests based on priority; check feasibility

Phase 2: How to allocate

If allocation fails: preempt lowest-priority running request;



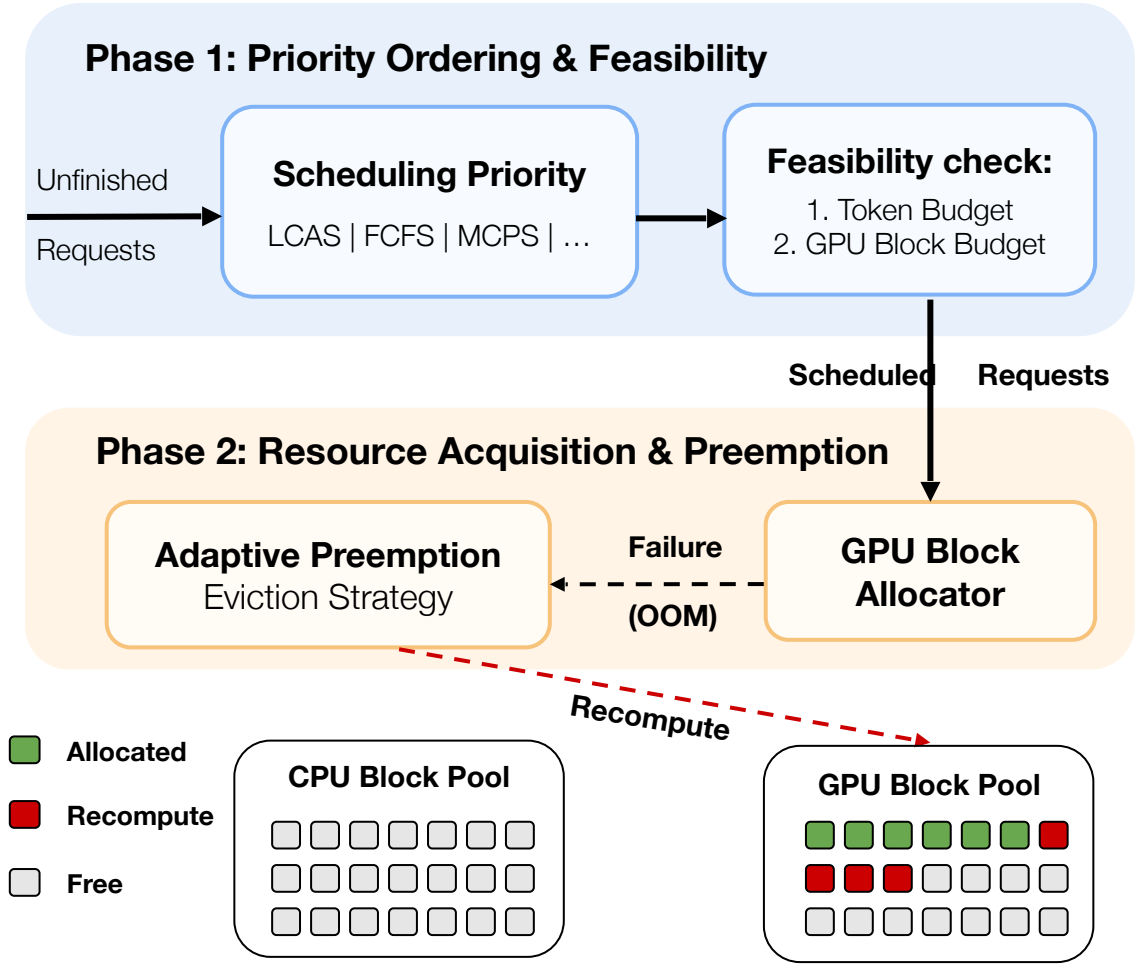
Two Phase Scheduling

Phase 1- What to run:

Select requests based on priority; check feasibility

Phase 2 - How to allocate:

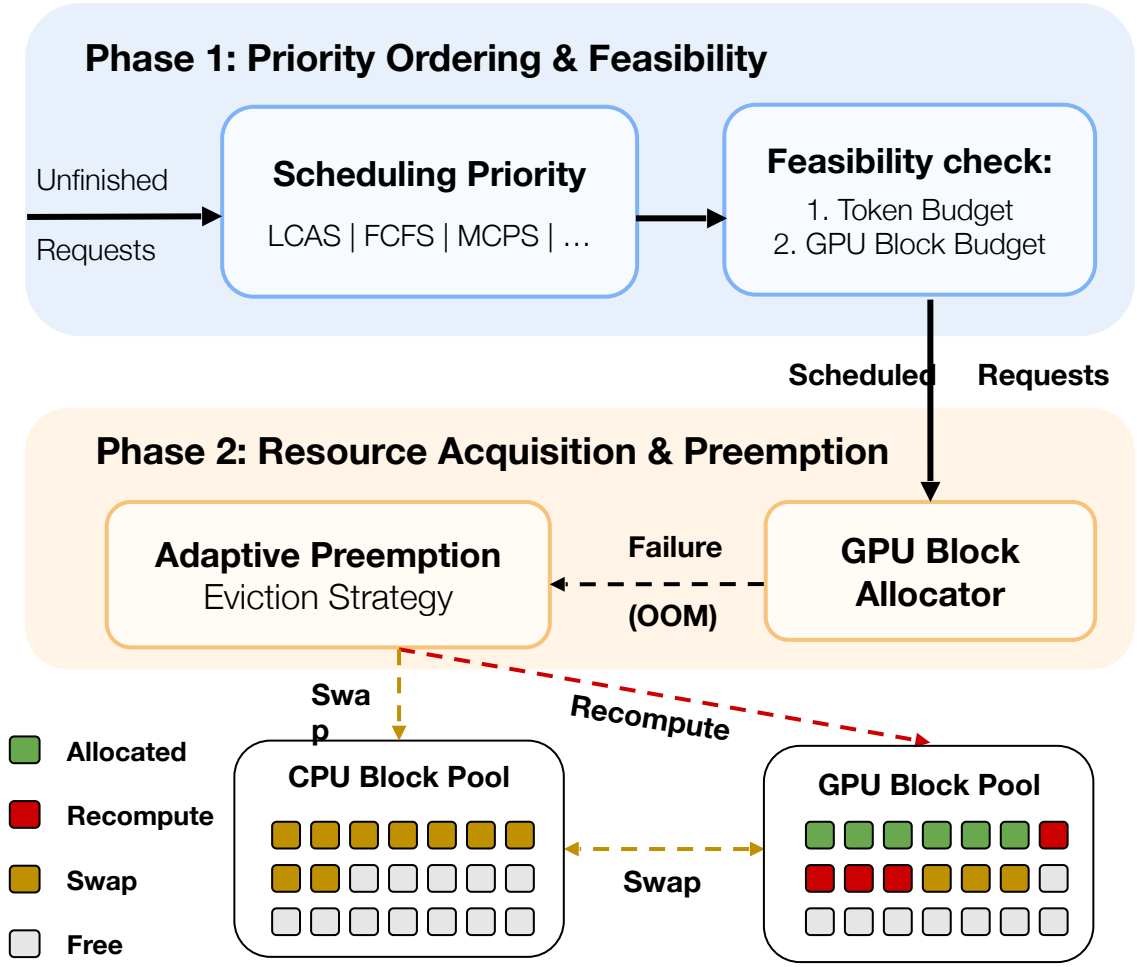
If allocation fails: preempt lowest-priority running request; choose recompute or swap (based on cost model)



Two Phase Scheduling

Phase 1- What to run:
Select requests based on priority; compute scheduling feasibility

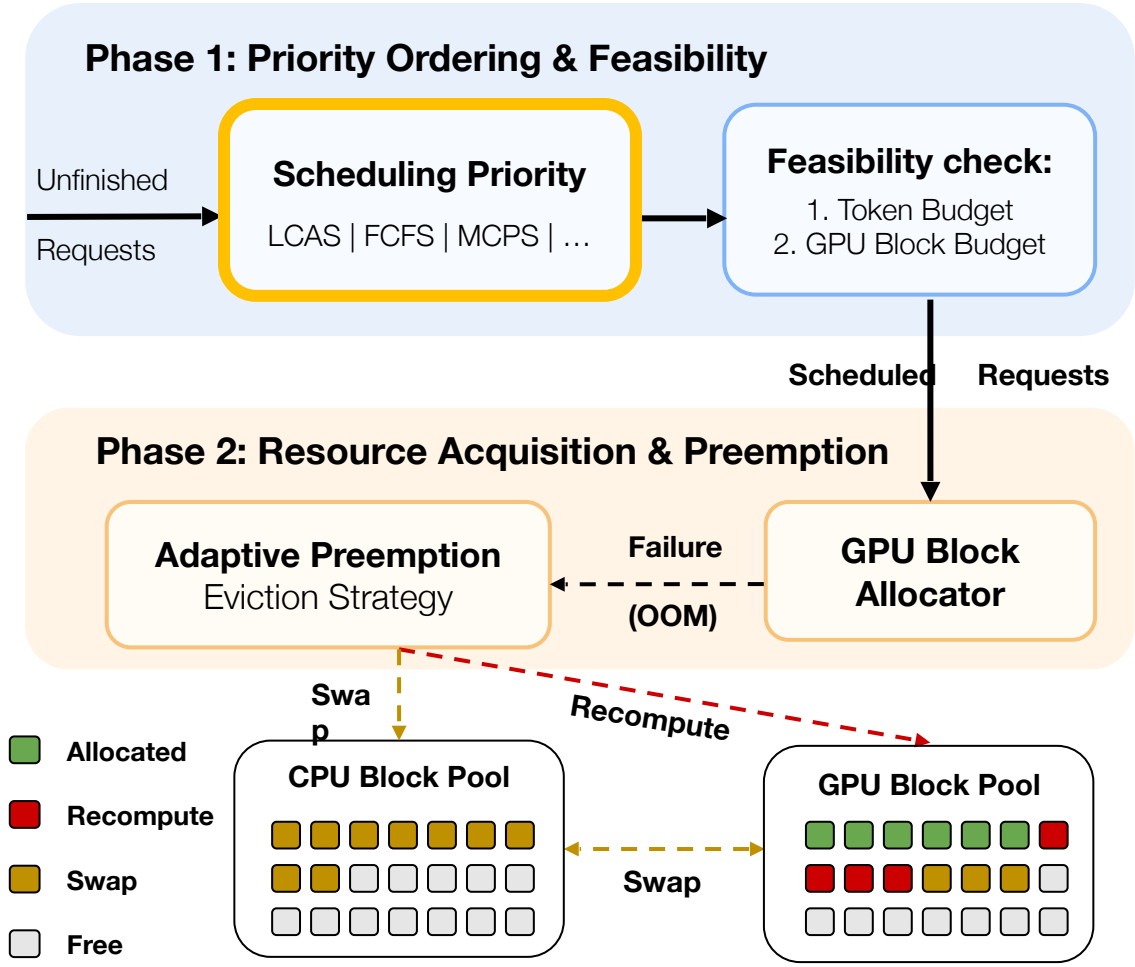
Phase 2 - How to allocate:
If allocation fails: preempt lowest-priority running request; choose recompute or swap (based on cost model)



Two Phase Scheduling

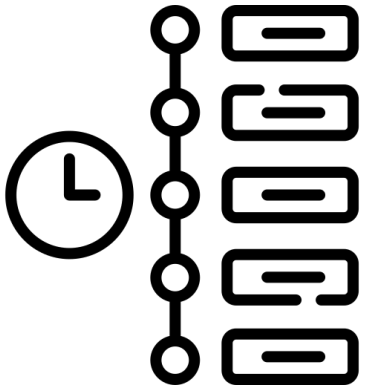
Phase 1- What to run:
Select requests based on priority; compute scheduling feasibility

Phase 2 - How to allocate:
If allocation fails: preempt lowest-priority running request; choose recompute or swap (based on cost model)



Scheduling Policies

**Eviction in reverse scheduling priority*

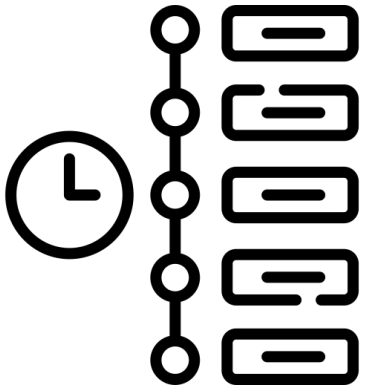


Default (FIFO)

- Based on request arrival order

Scheduling Policies

**Eviction in reverse scheduling priority*



Default (FIFO)

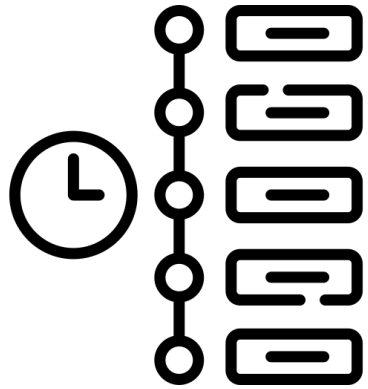
- Based on request arrival order

Most processed tokens (MPCS)

- Sort all requests based on progress (aka #tokens processed)

Scheduling Policies

**Eviction in reverse scheduling priority*



Default (FIFO)

- Based on request arrival order

Most processed tokens (MPCS)

- Sort all requests based on progress (aka #tokens processed)

First-Come-First-Served (FCFS)

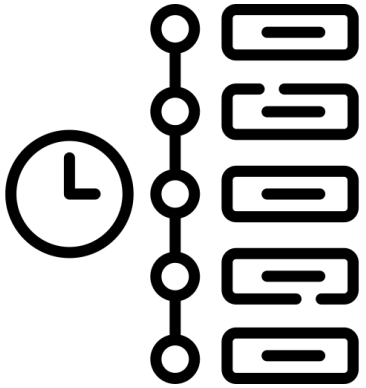
- Sort request in each tier using request arrival order

Last chunk arrival time (LCAS)

- Sort request in each tier using last chunk arrival time

Scheduling Policies

**Eviction in reverse scheduling priority*



Default (FIFO)

- Based on request arrival order

Most processed tokens (MPCS)

- Sort all requests based on progress (aka #tokens processed)

First-Come-First-Served (FCFS)

- Sort request in each tier using request arrival order

Last chunk arrival time (LCAS)

- Sort request in each tier using last chunk arrival time

Two-tier approach:

- *Intuition: Complete requests have higher priority over Partial*

Evaluation Setup

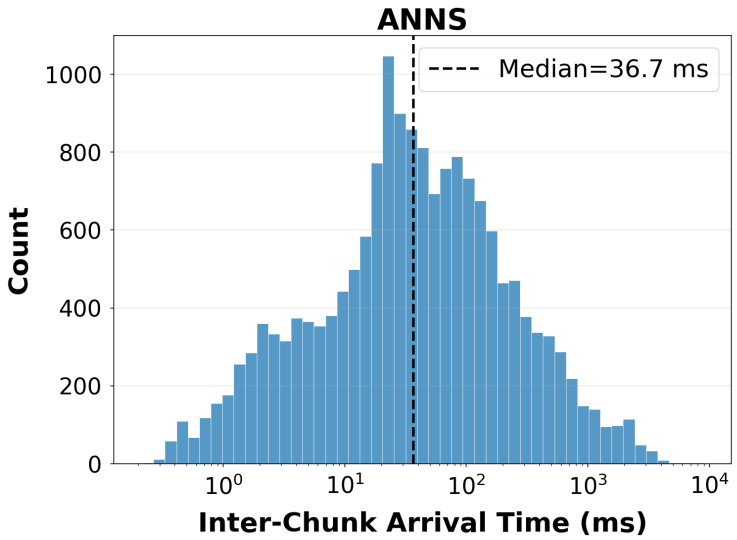
Model: Llama-3.1-8B-Instruct

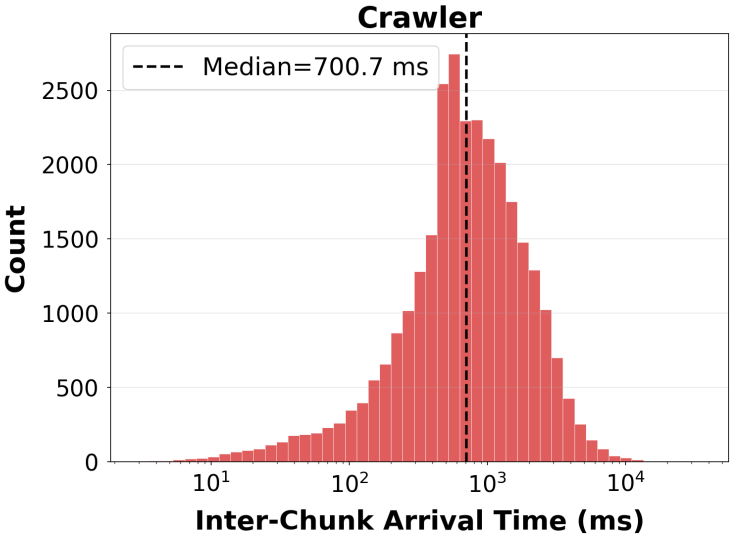
Hardware: H200 (TP=2)

Baselines: vLLM-NS vLLM-S

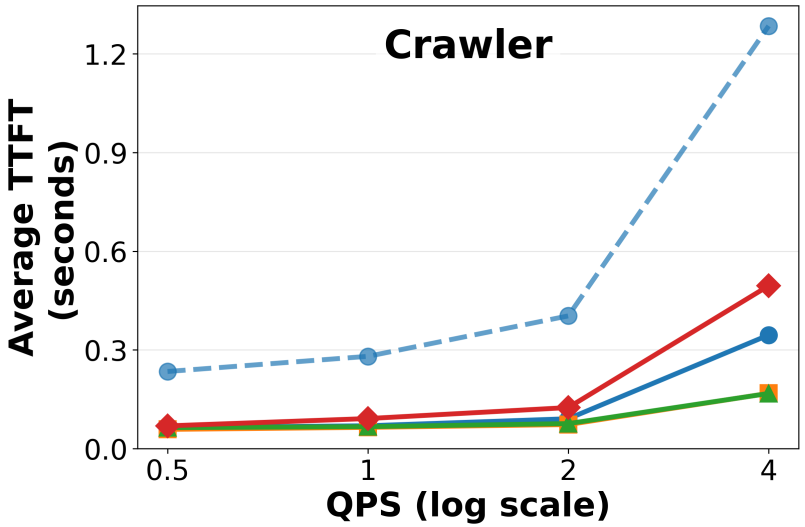
Retrieval Workloads:

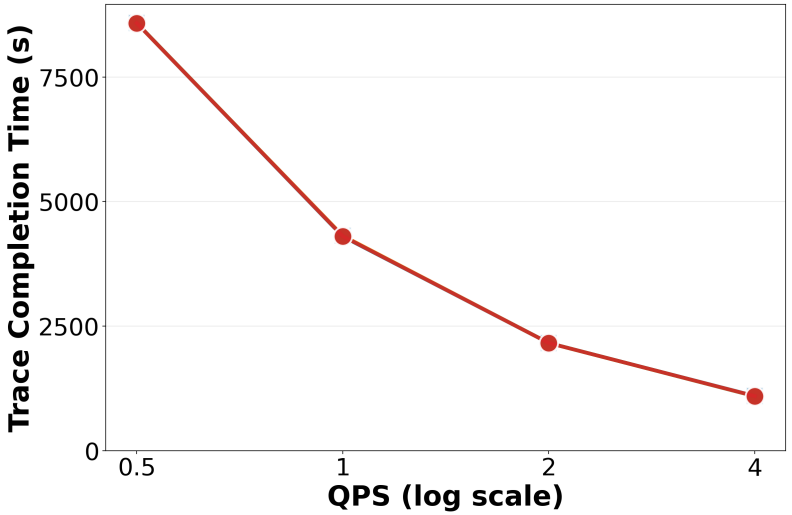
- Append – web crawler
  - Latency: median: 3.9s, P95: 8.5s
  - #Tokens/Query: median: 10K, P95: 31K
- Update – diskANN (AquaPipe [SIGMOD'25])
  - Retrieval Latency: median: 9.3s, P95: 16.7s
  - #Tokens/Query: median: 5.8K, P95: 28.9K





Evaluation (Append): TTFT vs QPS

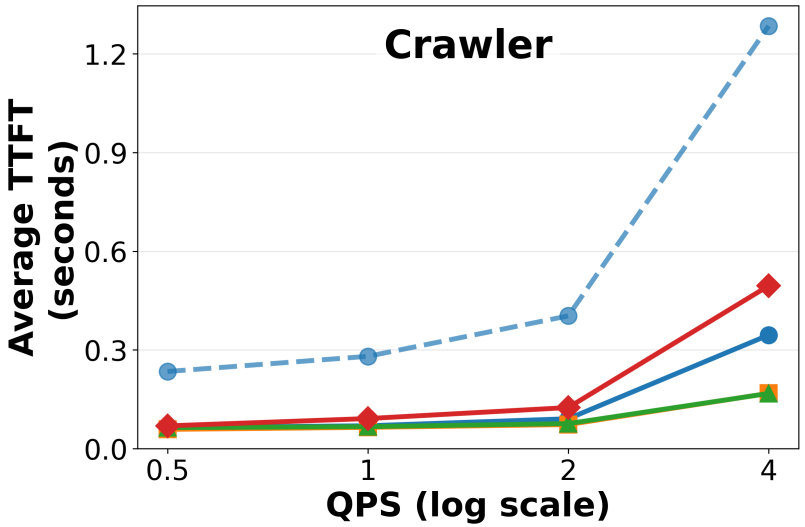


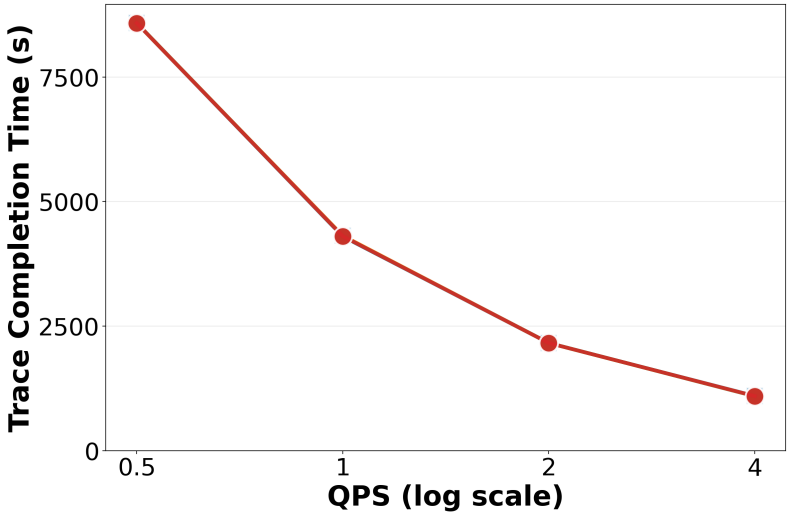


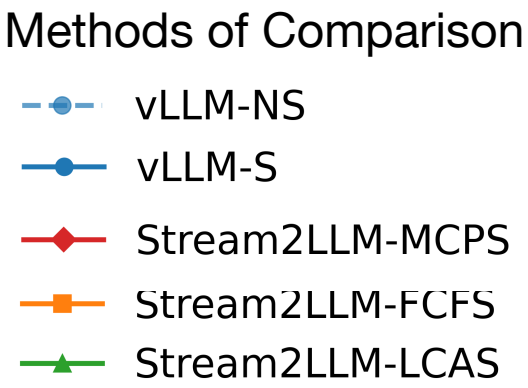
Methods of Comparison

- vLLM-NS
- vLLM-S
- Stream2LLM-MCPS
- Stream2LLM-FCFS
- Stream2LLM-LCAS

Evaluation (Append): TTFT vs QPS

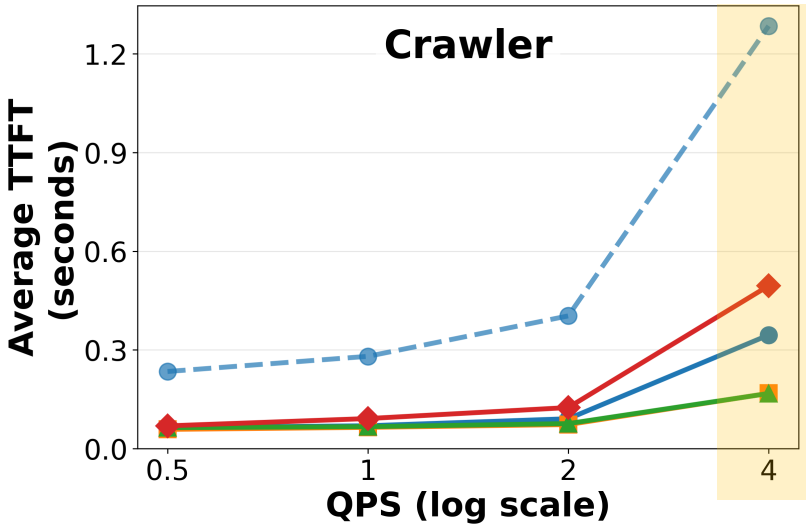


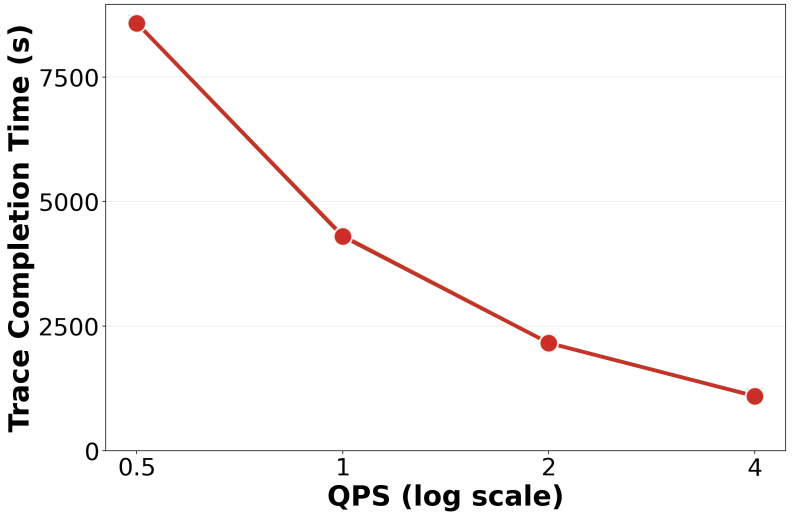


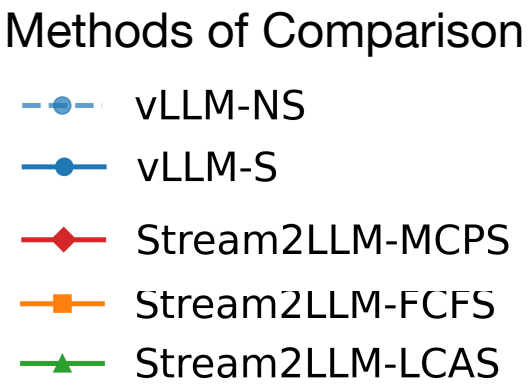


- Throughput is not reduced

Evaluation (Append): TTFT vs QPS

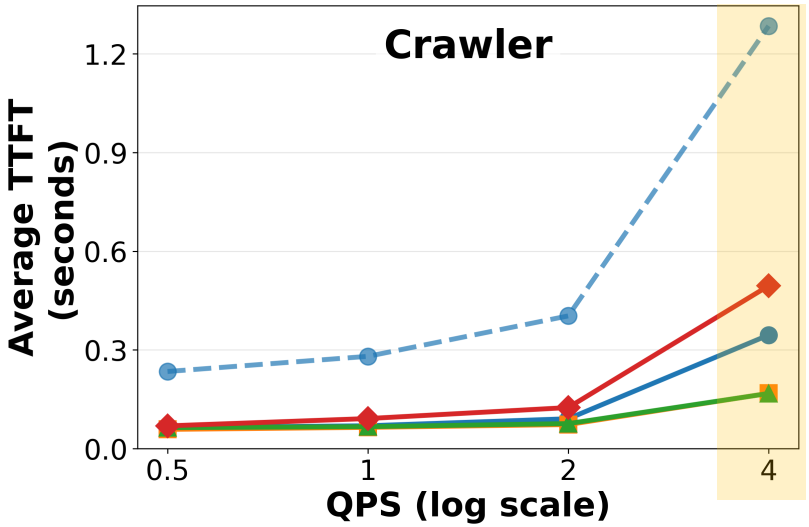


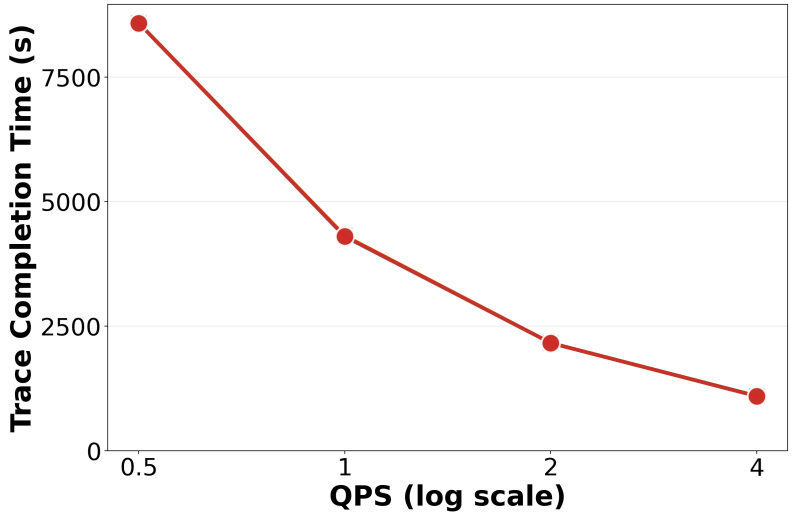




- Throughput is not reduced
- 11× speed up in median latency than non-streaming (QPS=4)

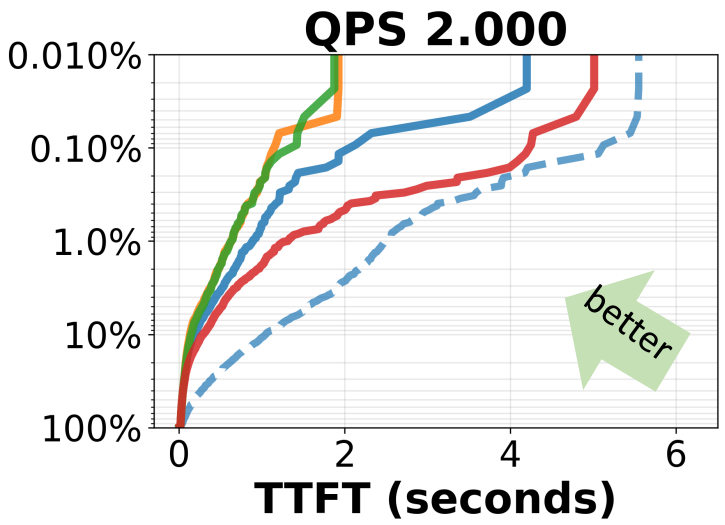
Evaluation (Append): TTFT vs QPS

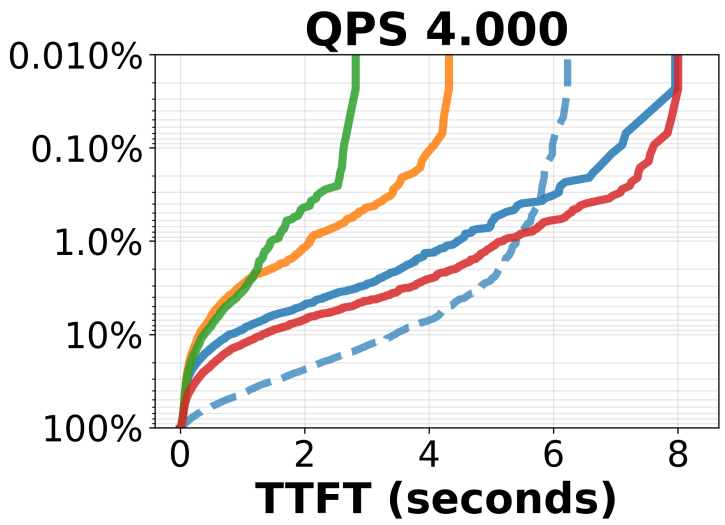




Methods of Comparison

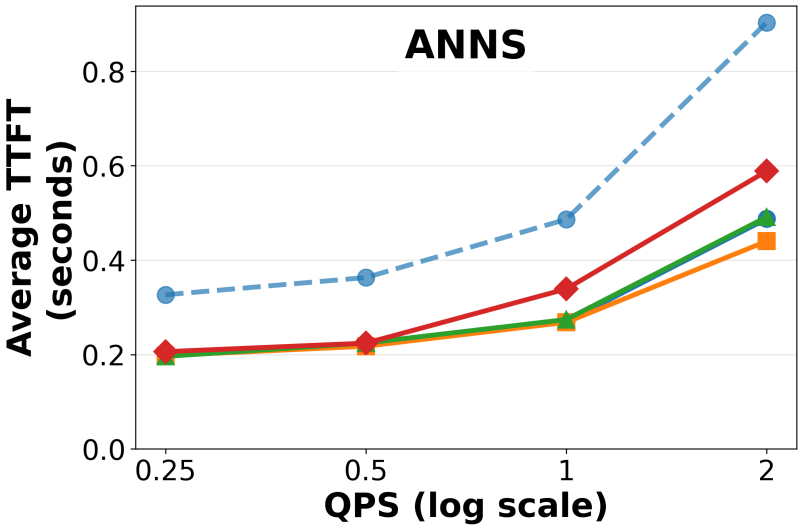
- vLLM-NS
- vLLM-S
- Stream2LLM-MCPS
- Stream2LLM-FCFS
- Stream2LLM-LCAS

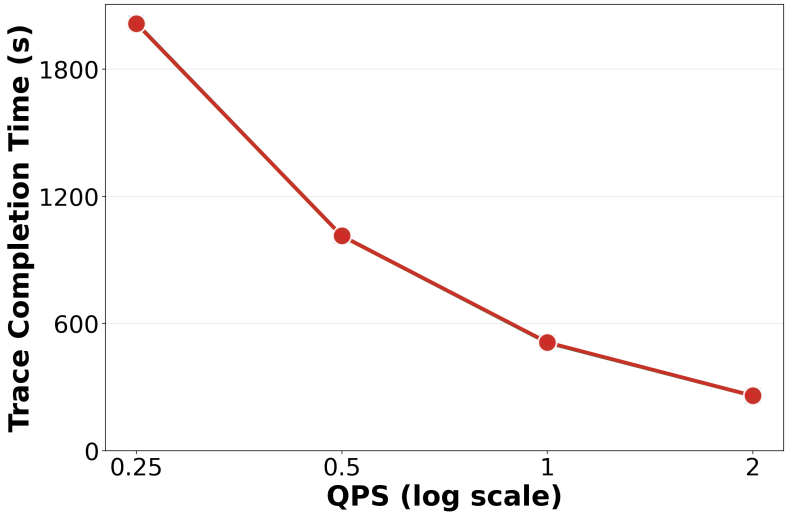




- Throughput is not reduced
- 11× speed up in median latency than non-streaming (QPS=4)

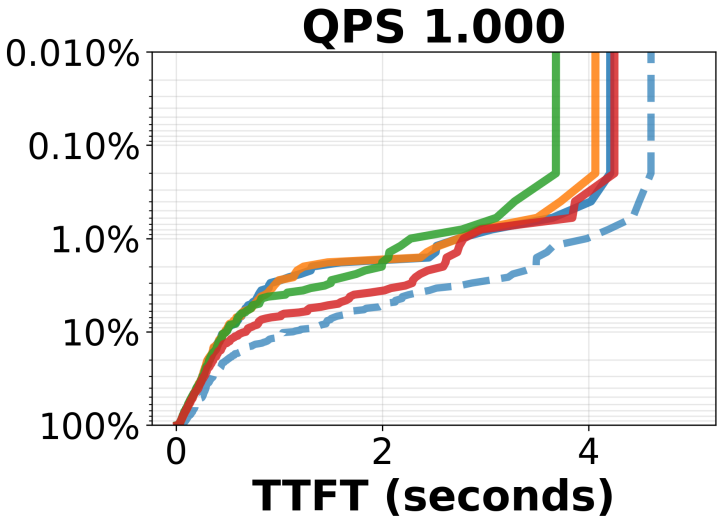
Evaluation (Update): TTFT vs QPS

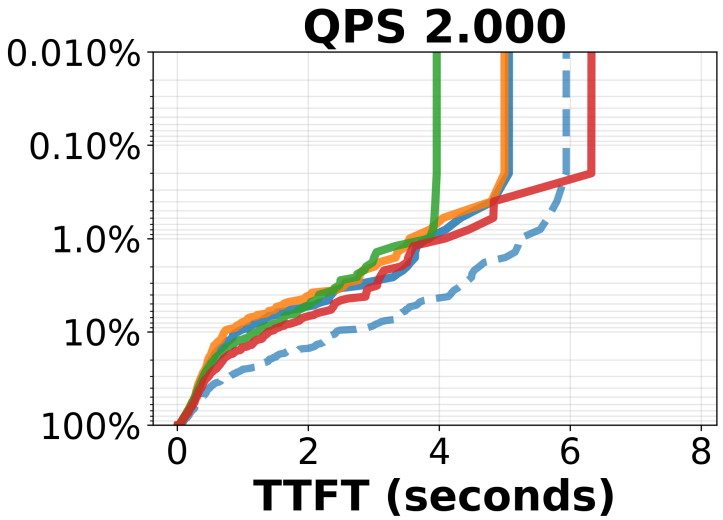




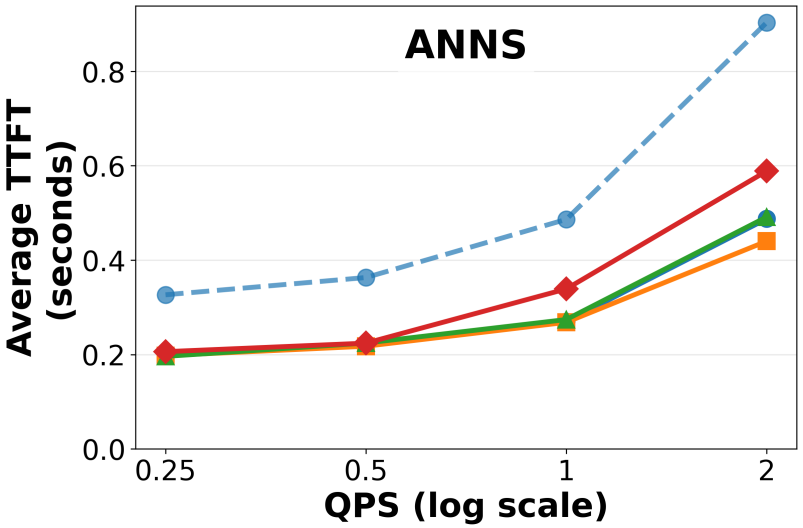
Methods of Comparison

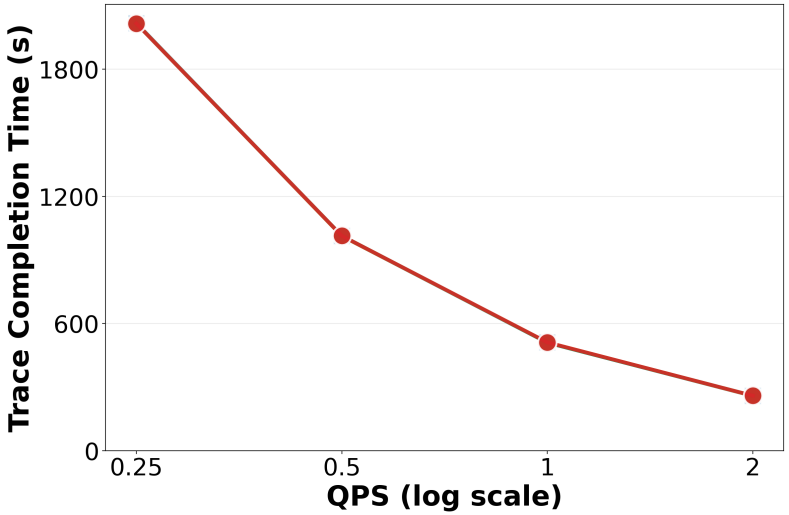
- vLLM-NS
- vLLM-S
- Stream2LLM-MCPS
- Stream2LLM-FCFS
- Stream2LLM-LCAS

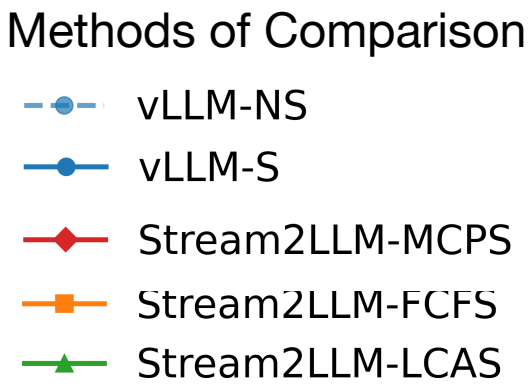


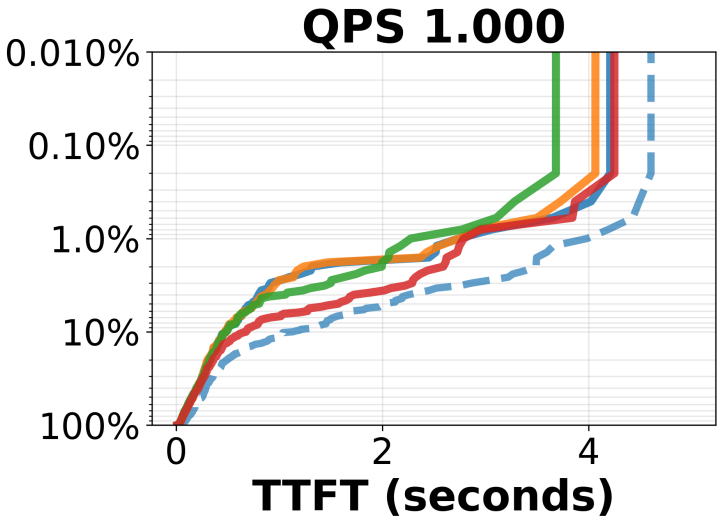


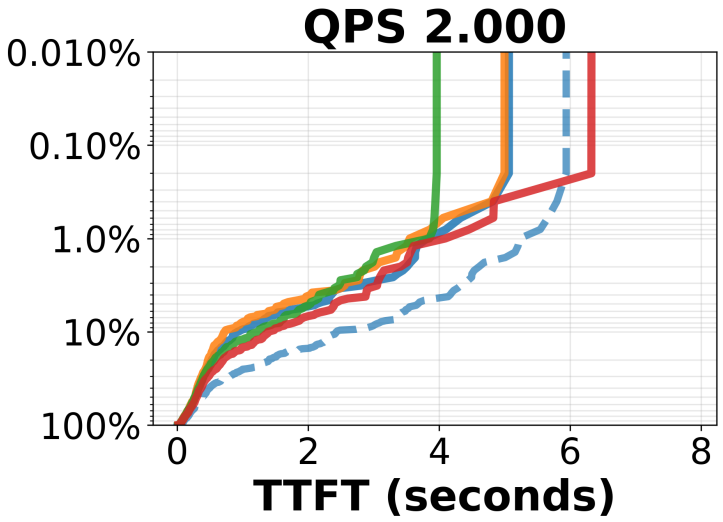
Evaluation (Update): TTFT vs QPS





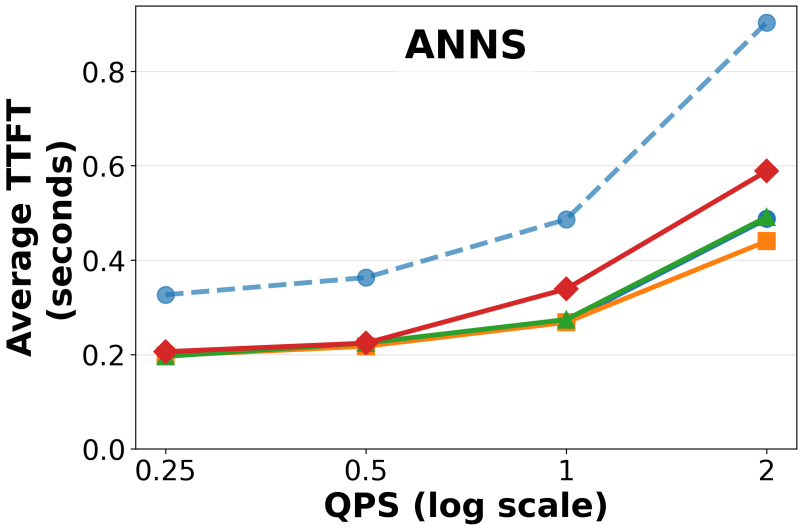


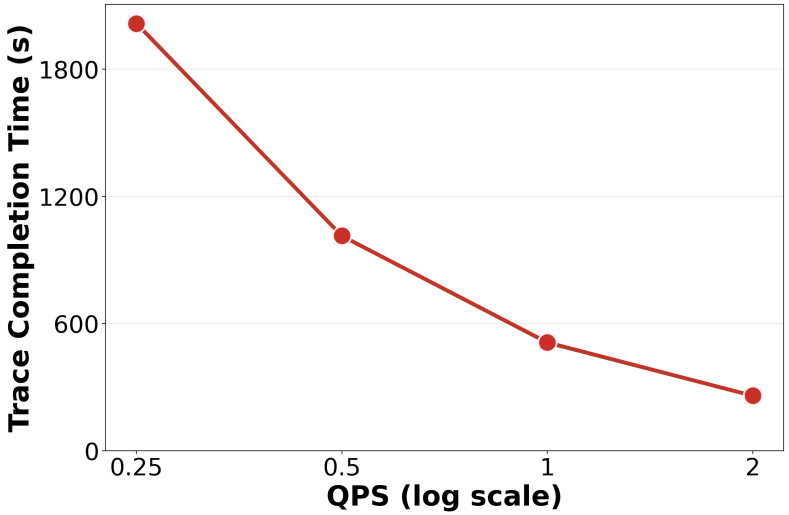


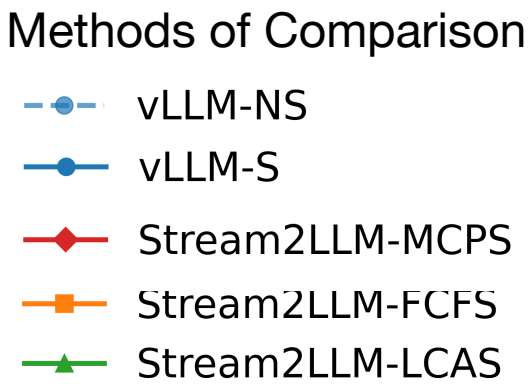


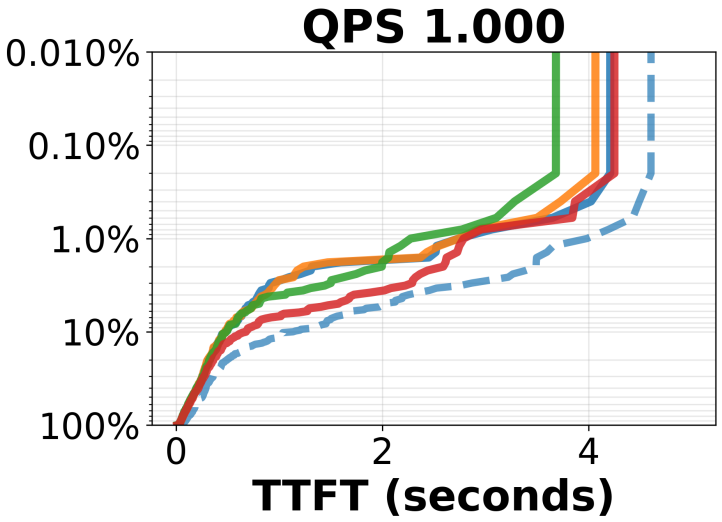
- Smaller gains compared to append mode: ~2× faster than non-streaming

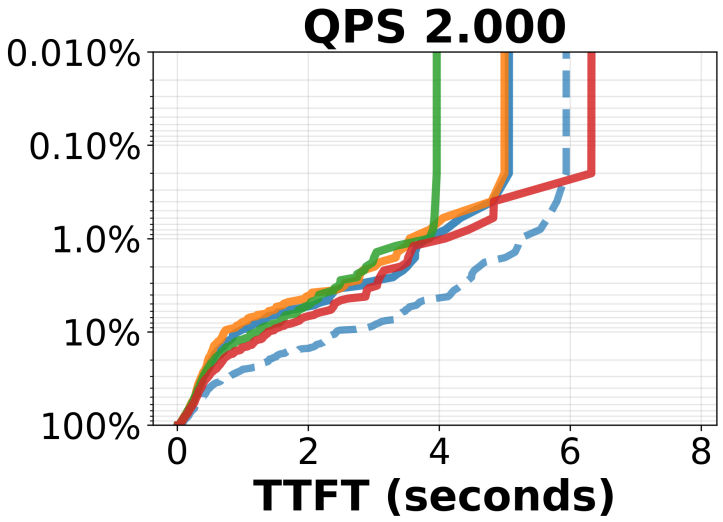
Evaluation (Update): TTFT vs QPS











- Smaller gains compared to append mode: ~2× faster than non-streaming
- >10% of requests invalidate over 10K tokens

Evaluation: Memory Pressure

Stress test: chunk inter-arrival delays to saturate GPU block pool

Scheduler	**Crawler** (4 QPS, 10× delay) P50	P99	**ANNS** (2 QPS, 30× delay) P50	P99
vLLM-NS (baseline)	1.00×	1.00×	1.00×	1.00×
vLLM-S (default stream)	8.25×	0.71×	2.63×	0.19×
Stream2LLM – FCFS	**8.30**×	8.62×	**2.70**×	**2.04**×
Stream2LLM – LCAS	7.82×	**9.14**×	2.44×	1.79×
Stream2LLM – MCPS	4.96×	0.77×	2.23×	0.19×

Naïve streaming algorithms could collapse under memory pressure!

Stream2LLM Summary

Scan for Code!

Overlapping retrieval with prefill to reduce TTFT

- Append vs update mode
- Streaming-aware scheduling design