Sizing LLM Inference Systems

Notebook 1: Understanding Batching Strategies

Metrics

LLM Inference Measurement – Time to First Token

The time it takes to process the input and generate the first token

Time to First Token (TTFT)

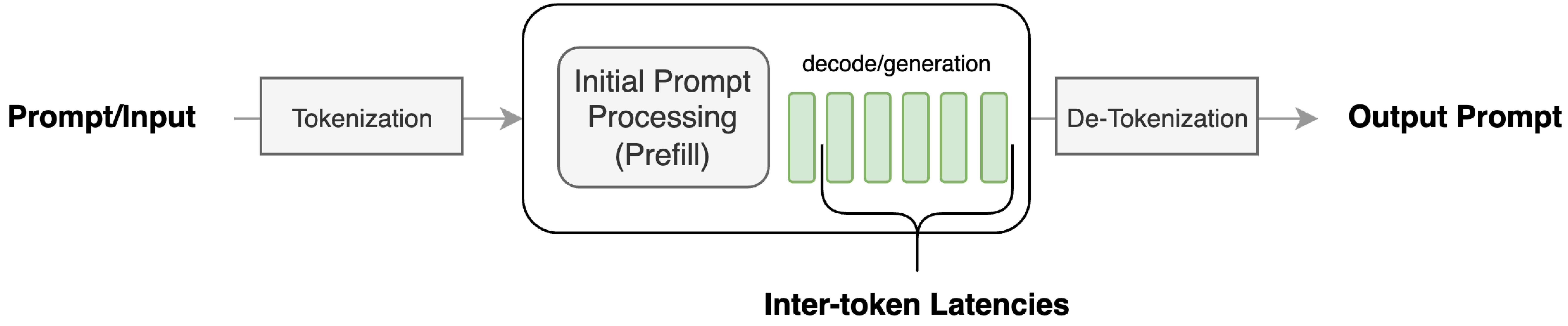
Signifies time it takes to process the prompt and generate the first token

Synonyms:
- First Token Latency (FTL)

https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/genai-perf/README.html#metrics

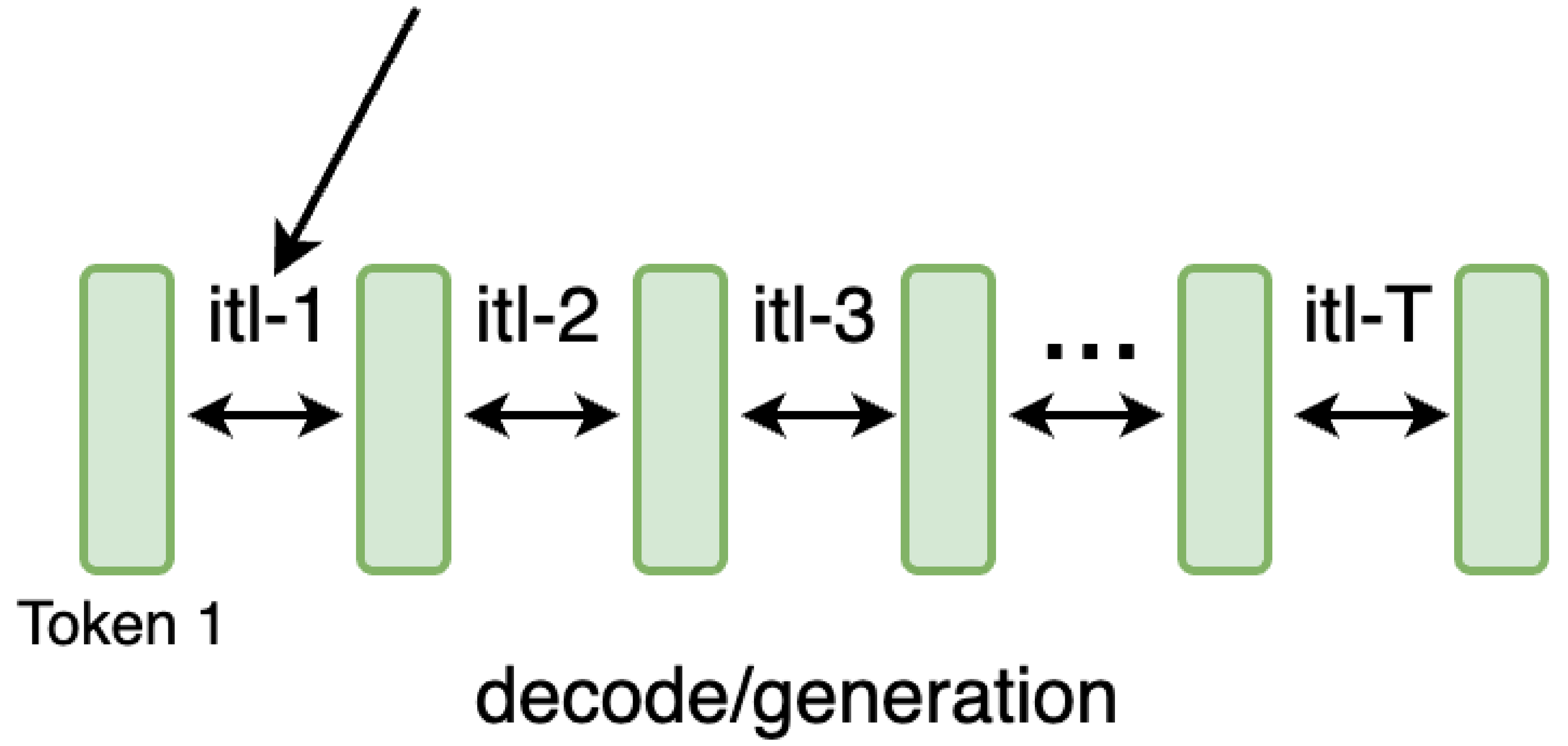
LLM Inference Measurement – Inter-token Latency

Measuring generation performance when the system is under load



Inter-token Latencies - Should be near constant over all token generations.

Near constant time implies efficient generation



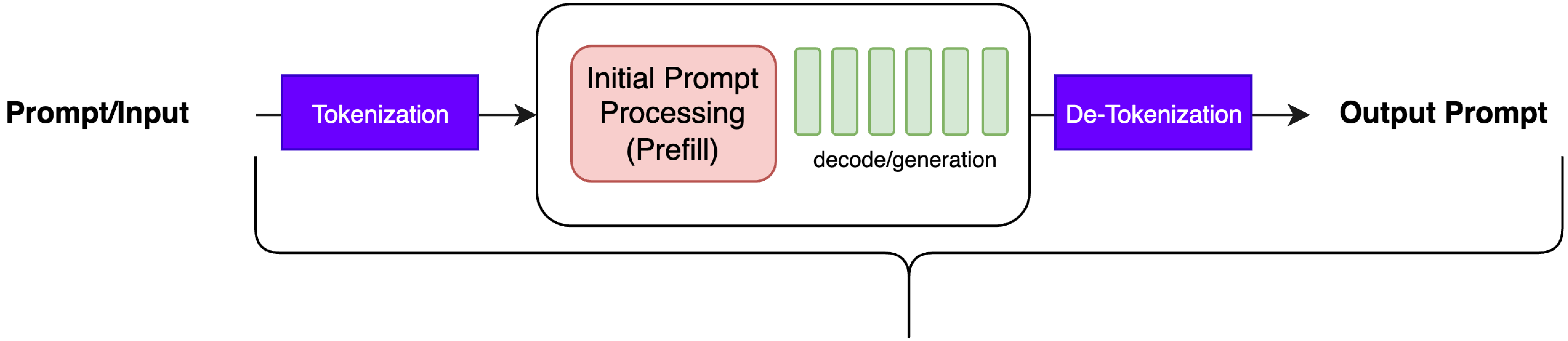
Synonyms:

- Time Per Output Token (TPOT)

https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/genai-perf/README.html#metrics

LLM Inference Measurement – Total Time to Generation

The time it takes to process a prompt and generate all the tokens



Total Time to Generation

Measures to total time to process input and generate all tokens

Synonyms:

- End-to-End Latency (E2E latency)
- Time to last Token

https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

Throughput Metrics

- Always specify
  - Model
  - Precision
  - Input Length
  - Output Length
  - Concurrency
  - TP
- The most unambiguous metric
  - to measure is **requests/second/instance**
  - to use in sizing **requests/second/GPU**

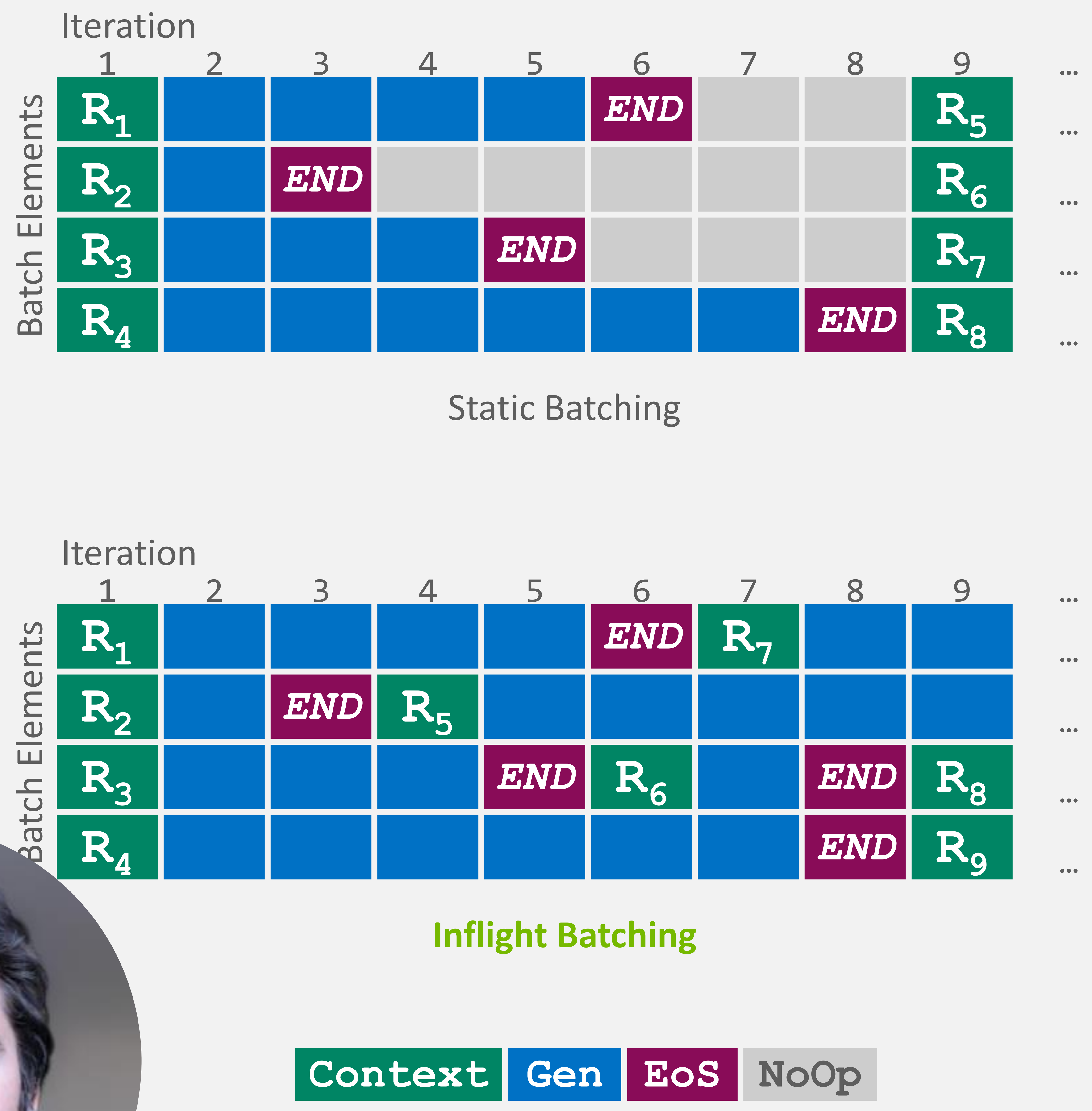
Inference Optimizations

Inflight Batching

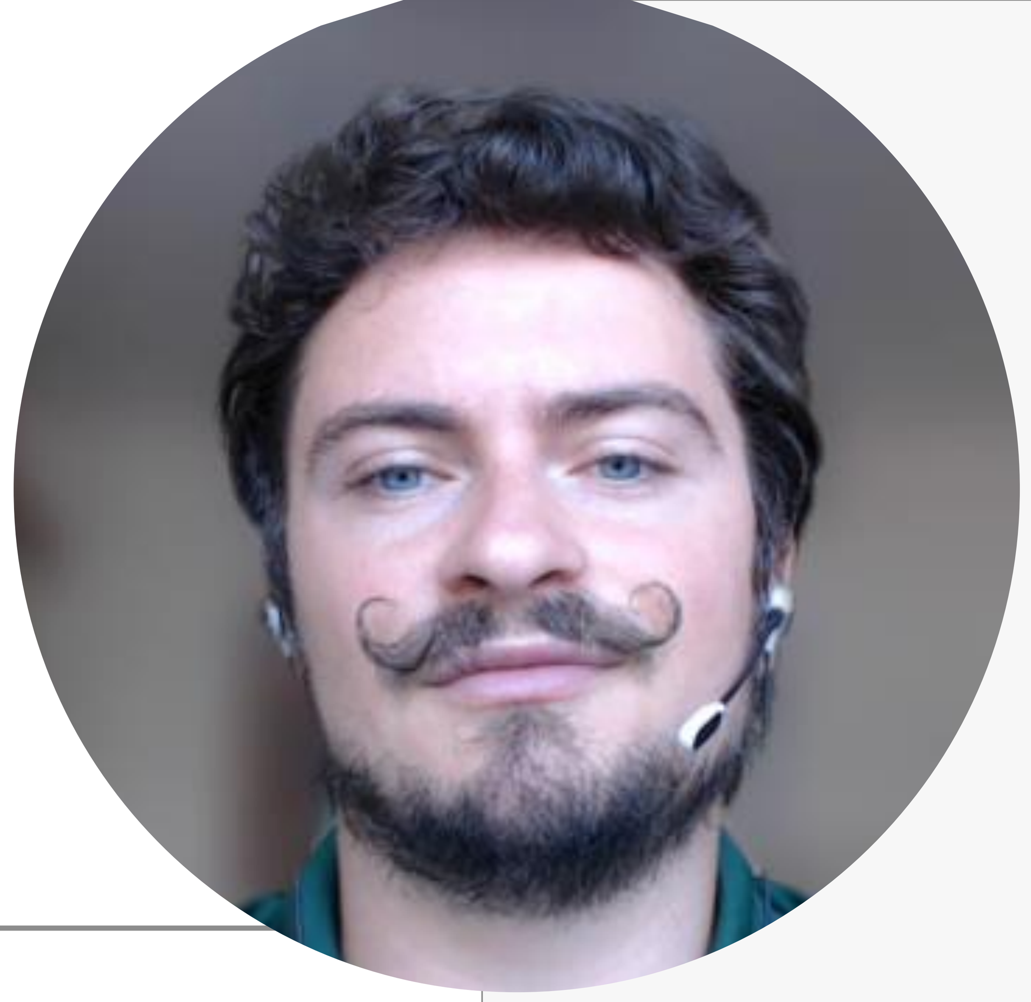
Maximing GPU Utilization during LLM Serving

TensorRT-LLM provides custom Inflight Batching to optimize GPU utilization during LLM Serving

- Replaces completed requests in the batch
  - Evicts requests after EoS & inserts a new request
- Improves throughput, time to first token, & GPU utilizaiton
- Integrated directly into the TensorRT-LLM Triton backend
- Accessible though the TensorRT-LLM Batch Manager



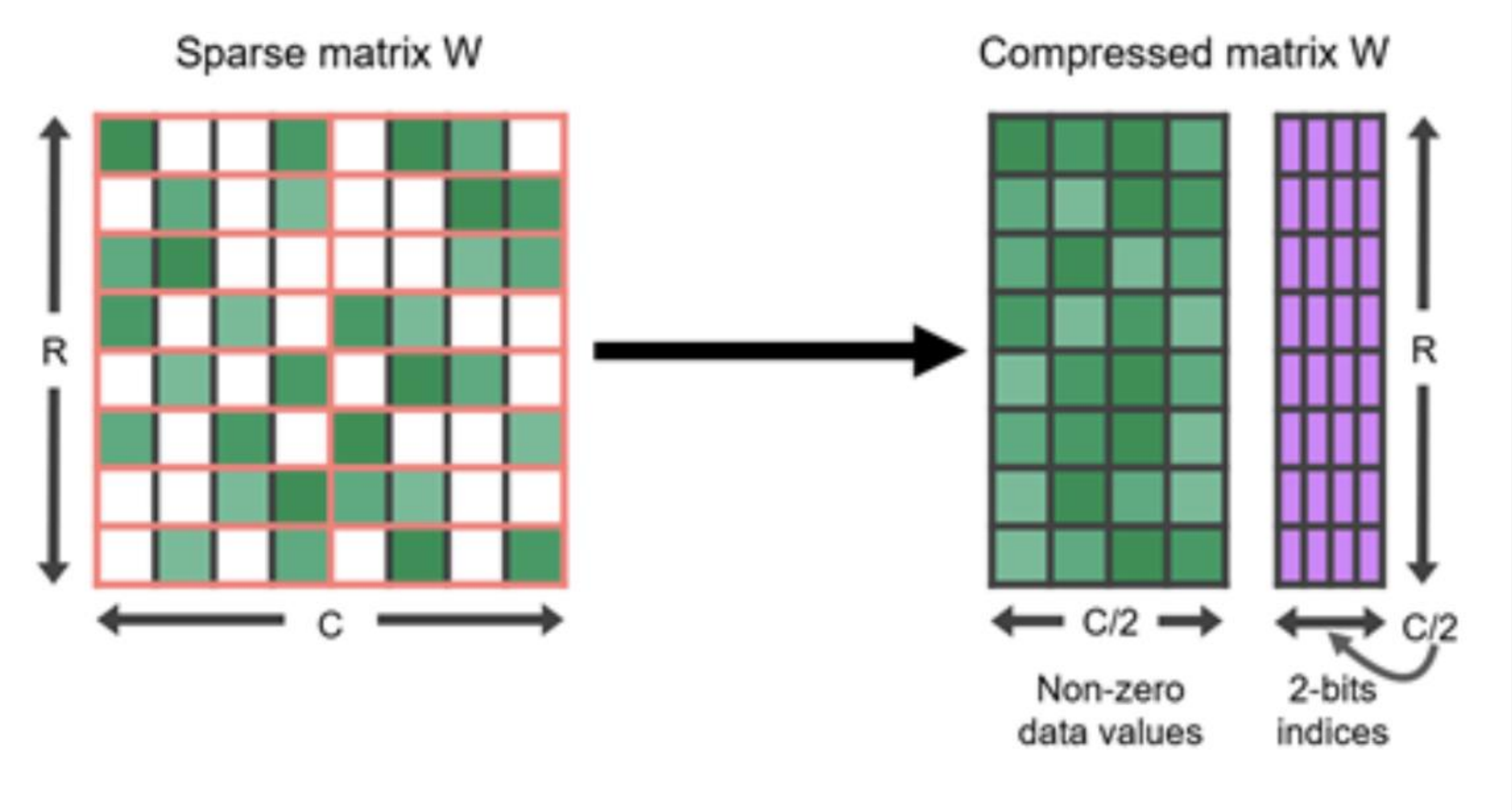
Model Optimizer - Sparsity



Model Optimizer offers Sparsity (with fine-tuning or post-training sparsity) to reduce the memory footprint and accelerate inference.

It supports NVIDIA 2:4 Sparsity sparsity pattern and various sparsification methods, such as (NVIDIA ASP) and (SparseGPT).

- **MLPerf-Inference v4.0 Results** (blog)
  - Uses 2:4 sparsity on a Llama2-70B.
  - Post-training sparsification (PTS) with SparseGPT yields no accuracy drop.
  - PTS reduces model size by 37%, facilitating deployment on a single H100 SXM with TP=1, PP=1, and achieving a 1.3x speedup.



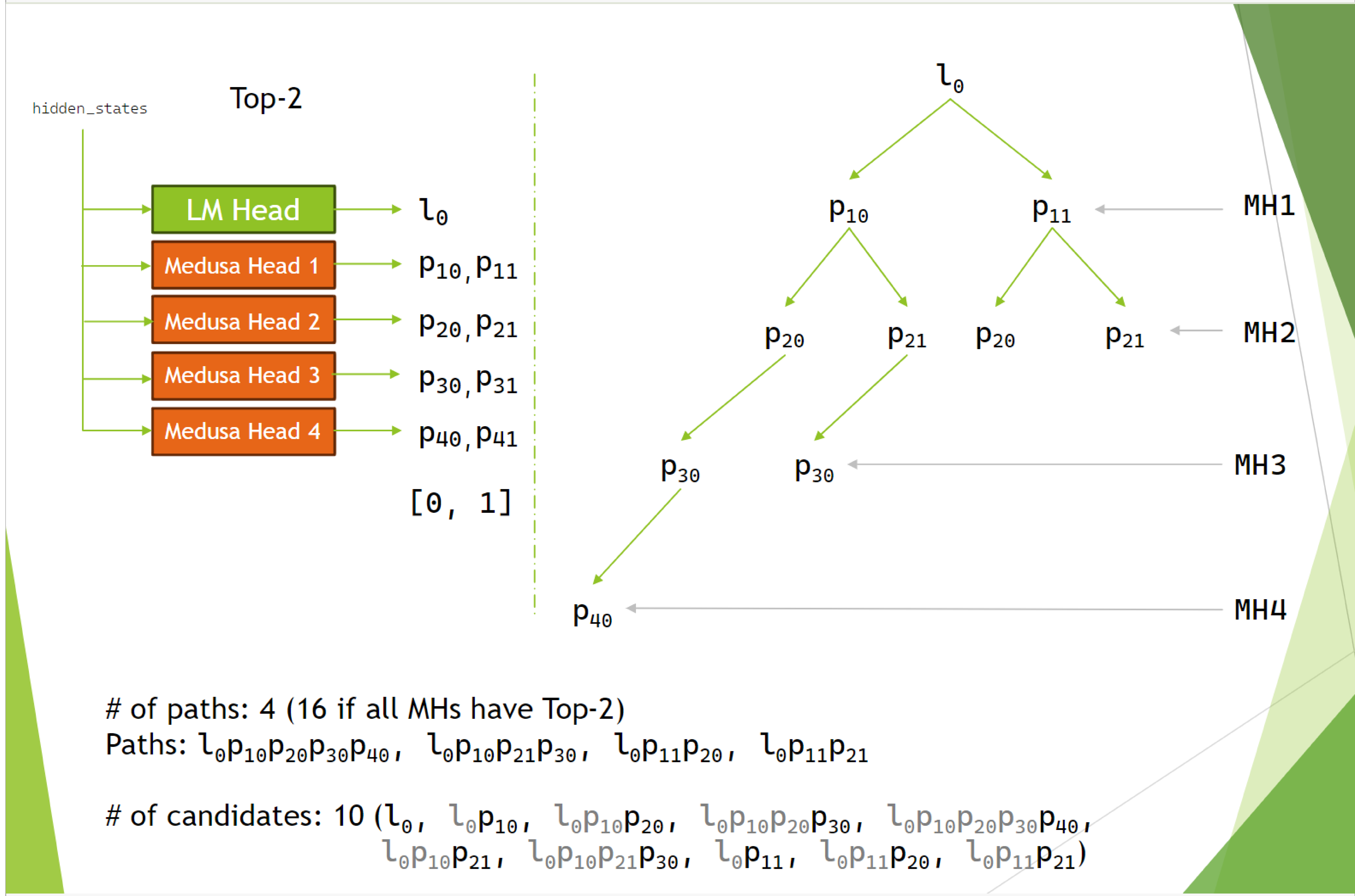
A 2:4 structured sparse matrix W, and its compressed representation

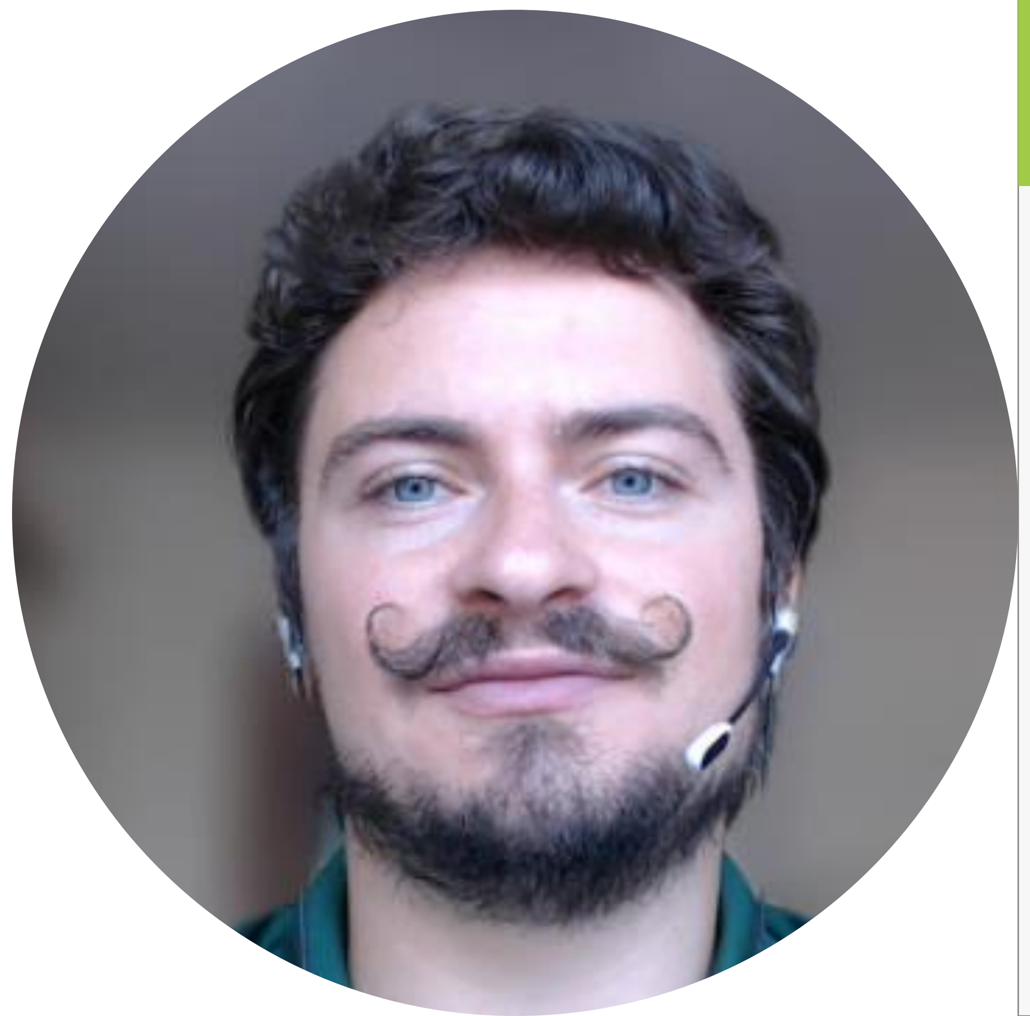
Model	Batch_size	Inference Speedup (Compared to the FP8 dense model with the same batch size)
Sparsified Llama 2-70B	32	1.62x
	64	1.52x
	128	1.35x
	896 (MLPerf)	1.30x

Performance improvement with sparsity

Speculative Decoding

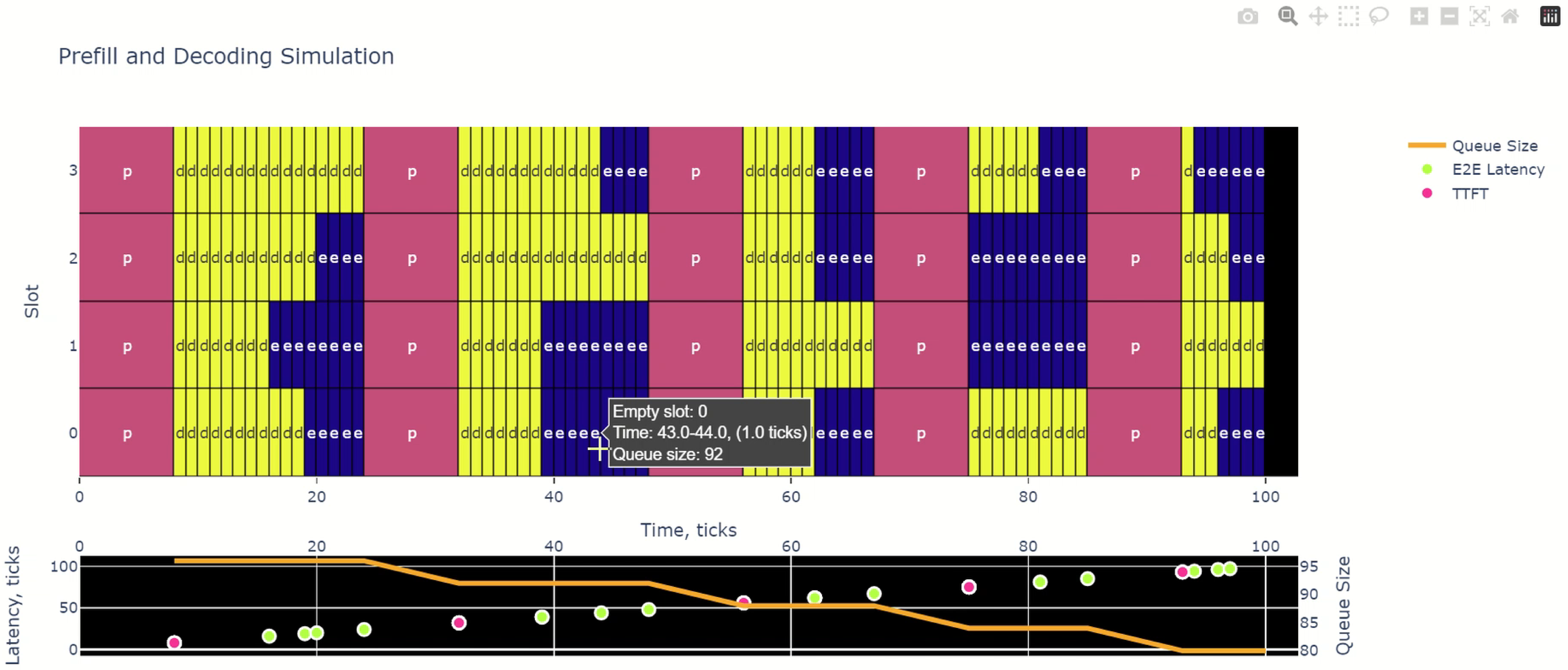
- Generate of more than one token per forward pass iteration
- Helpful if only if the GPU is underutilized due to small batch sizes.
- Perform more computations, keeping memory access almost the same
- Supports: separate draft model, Medusa, ReDrafter





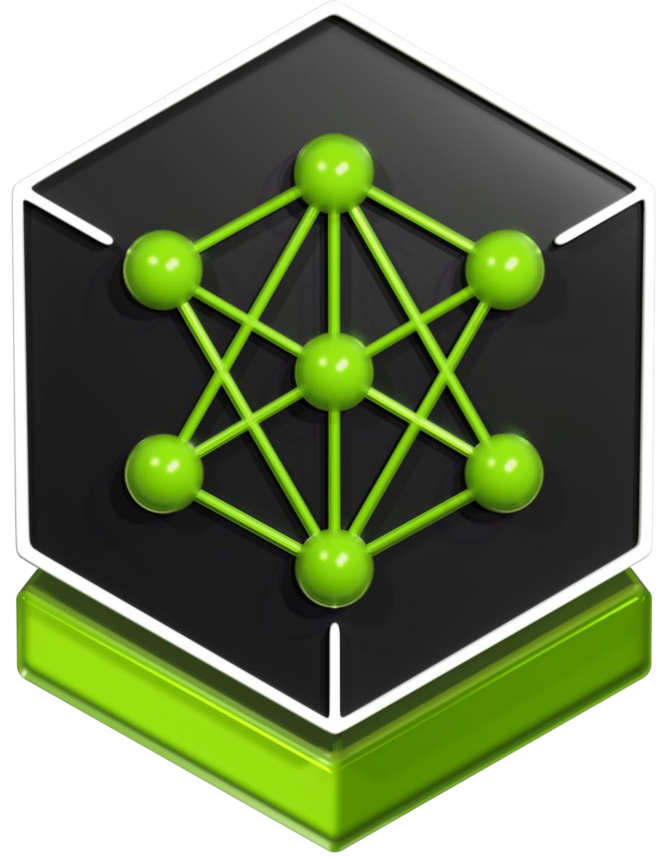
The Simulator

The Simulator



Objectives of this notebook

1. Understand and measure time to first token (TTFT), end-to-end latency (E2E Latency), and inter-token latency (ITL).
2. Analyze throughput metrics and simulate their dependencies on various factors.
3. Explore the impact of batching and inflight batching on GPU utilization and performance.
4. Investigate the effects concurrency settings on latency and throughput.



NVIDIA