Optimizing AI-Powered NPCs Cost-Efficiency Using TRT-LLM Without Sacrificing Quality

20th March 2024 | GTC

Igor Poletaev, VP of AI, Inworld
Sagar Singh, Data Scientist, NVIDIA

Leading AI Platform for Games

Inworld is a vertically integrated AI platform optimized for AAA Game Studios.

We are primarily known for leading the market in tools for building and integrating **next-generation AI NPCs.**

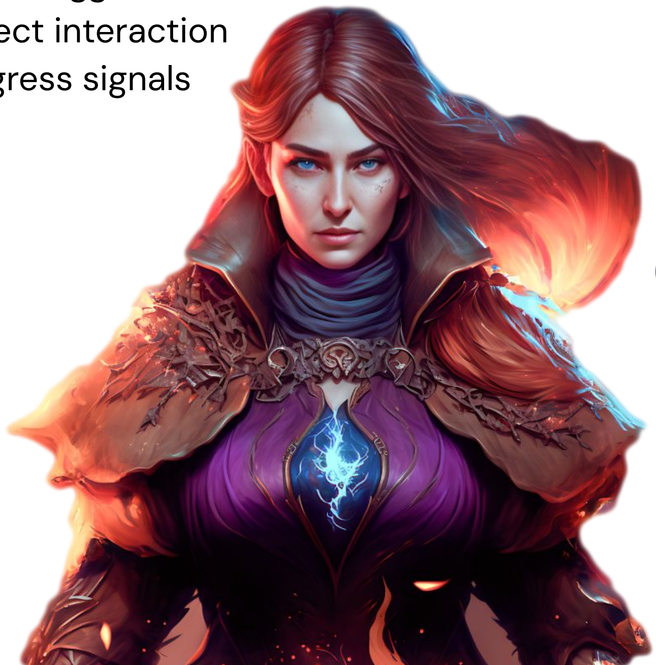
Our tools span from designtime, augmenting existing game development, to core infrastructure to enable in-house ML training and serving.

Inworld Engine

Real-time optimized engine is powered by **dozens of ML models orchestrated** together to mimic social and expressive nature of human interactions

Perception

- Audio
- Visual
- Event triggers
- Object interaction
- Progress signals



Cognition

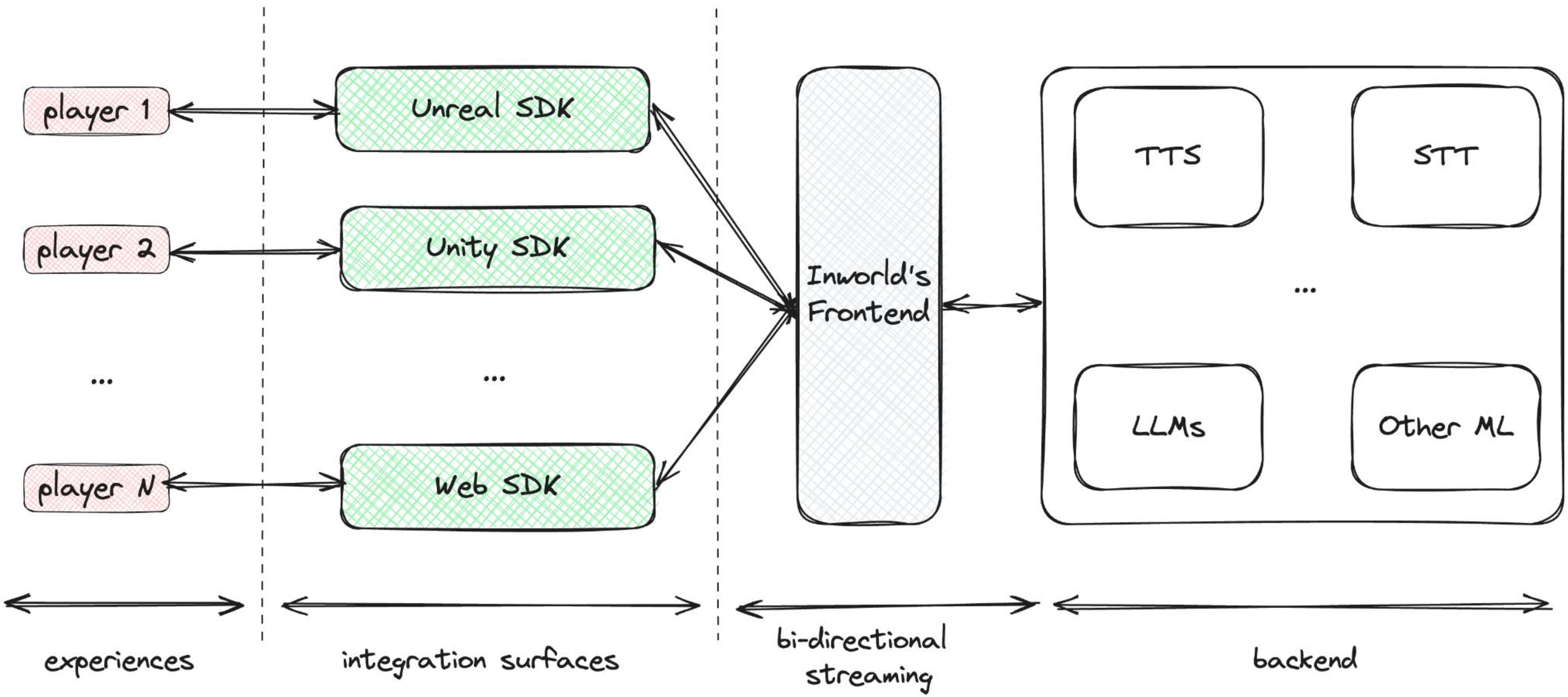
- Personality
- Background
- Goals
- Memory
- Emotions

Behavior

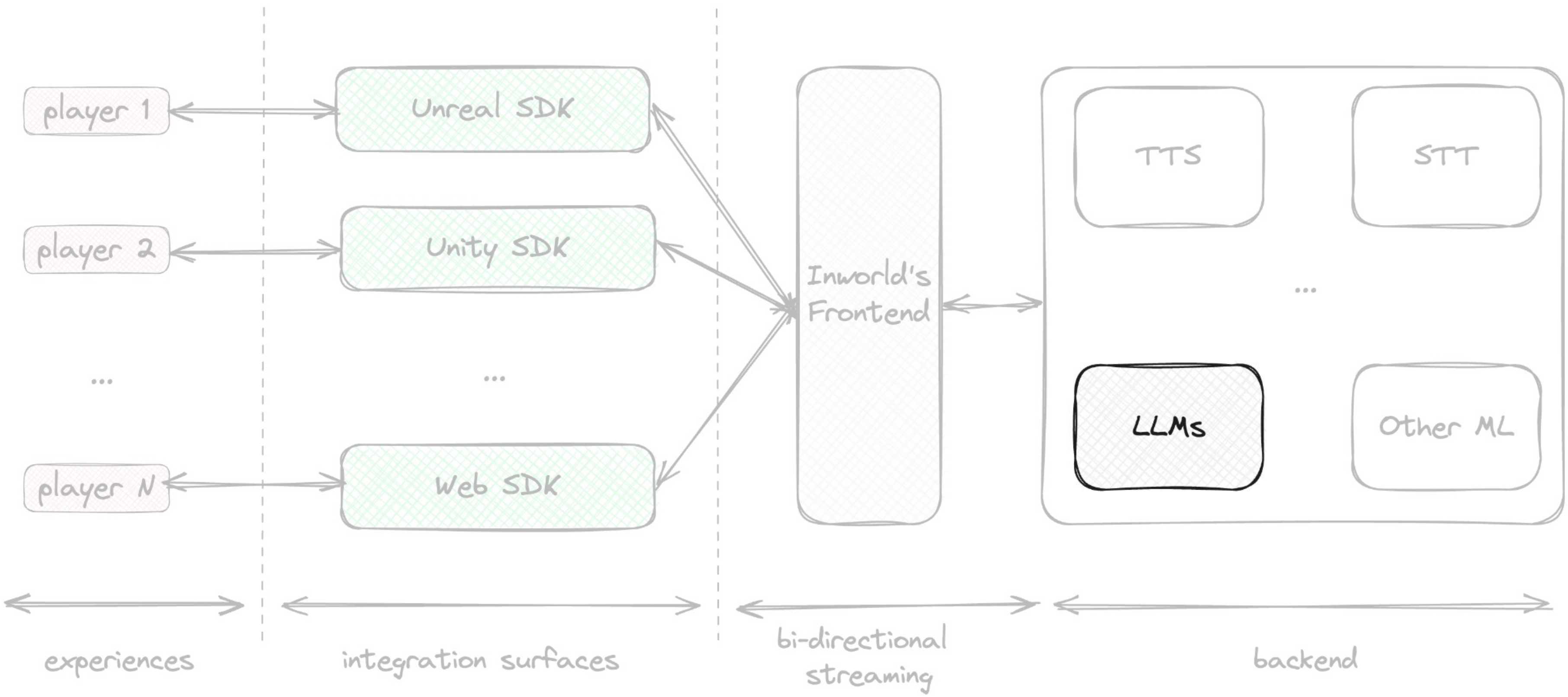
- Speech
- Gestures
- Body language
- Movement
- Event triggers

Runtime

Inworld



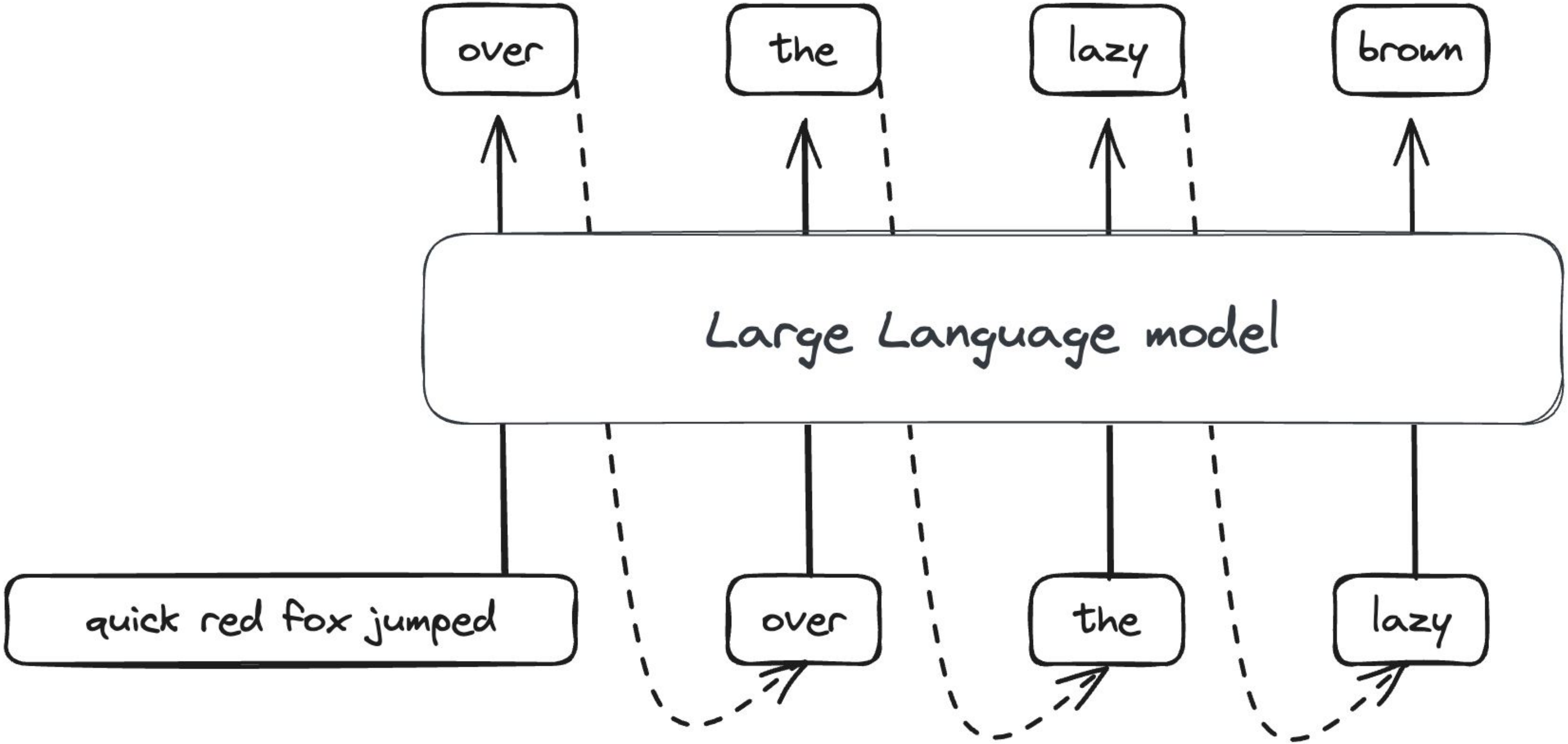
Runtime



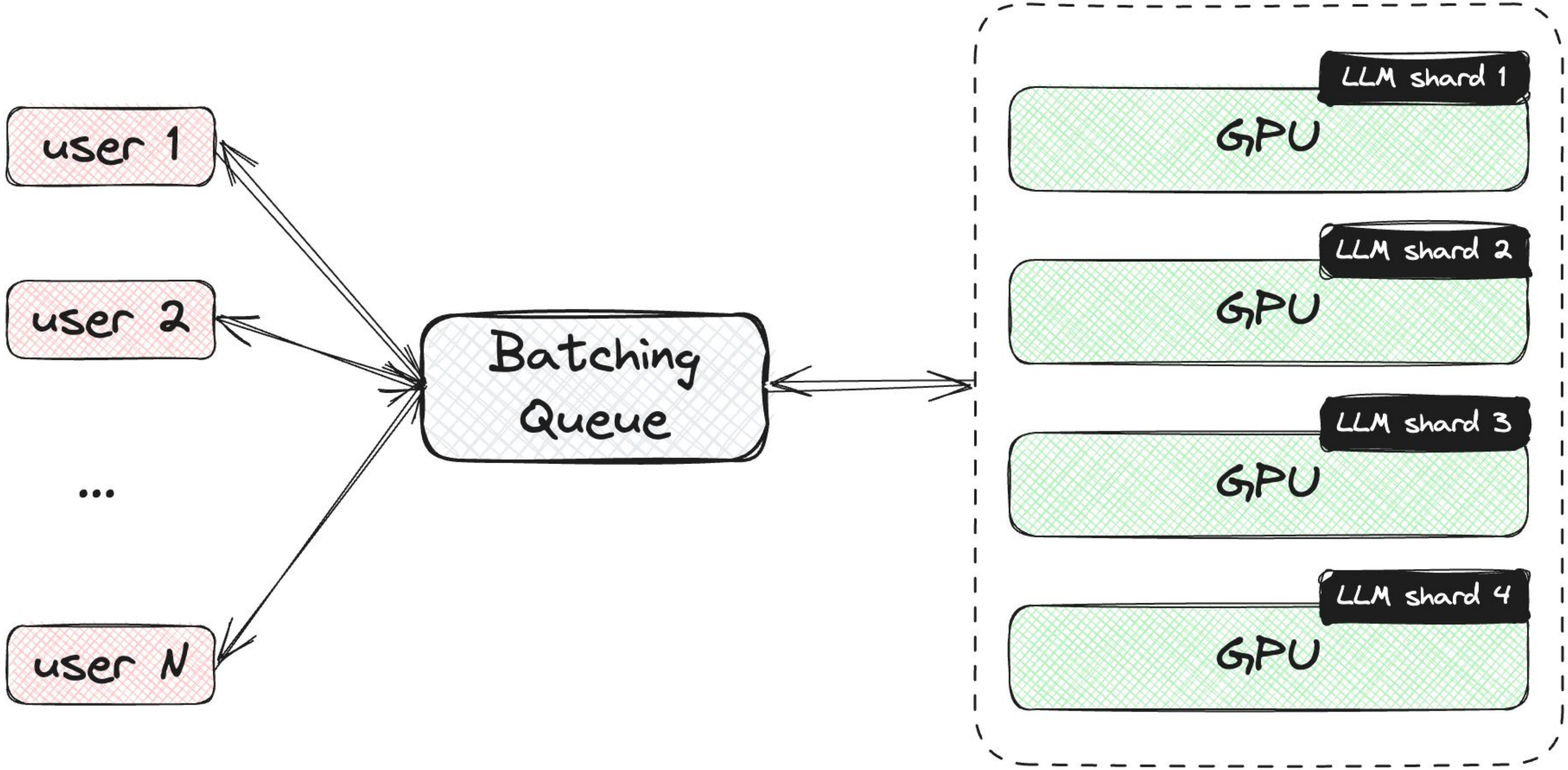
player 1
player 2
...
player N
Unreal SDK
Unity SDK
...
Web SDK
Inworld's Frontend
TTS
STT
...
LLMs
Other ML
experiences
integration surfaces
bi-directional streaming
backend

LLM serving

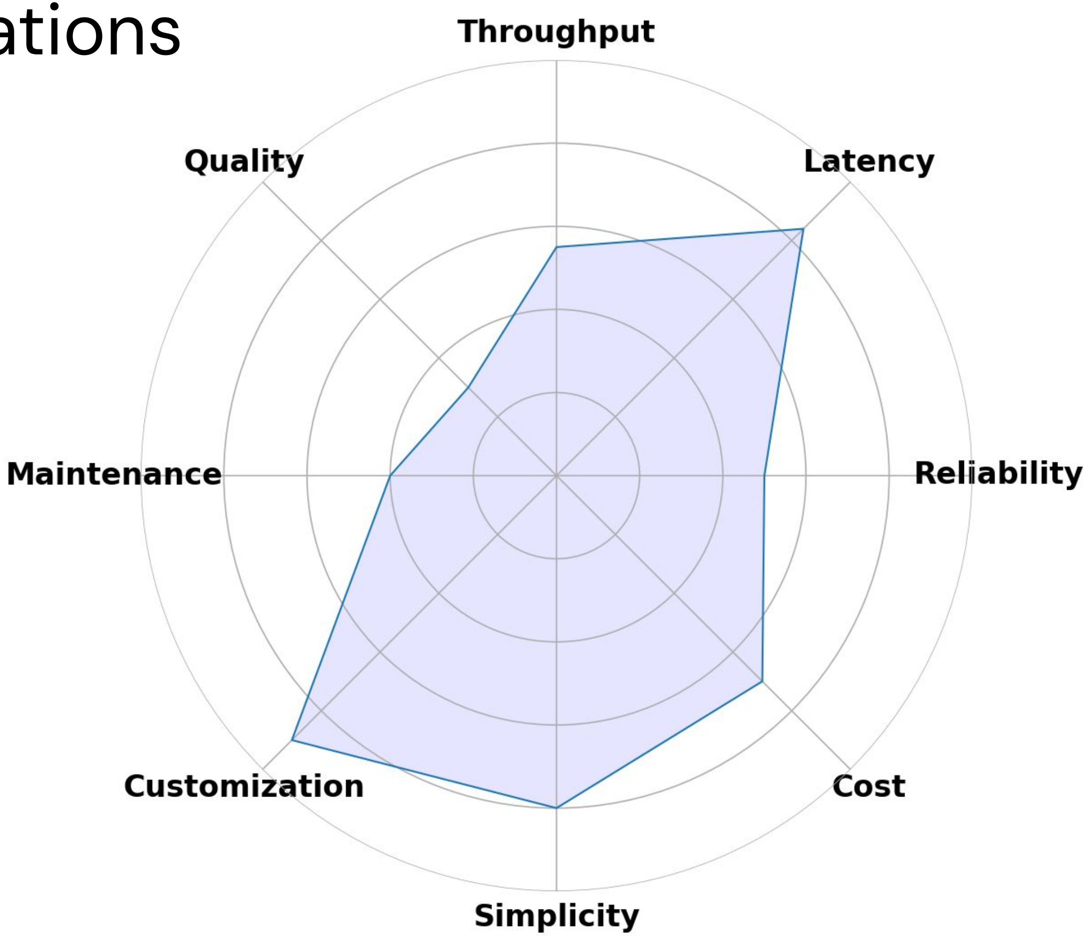
GPTs


over
the
lazy
brown
Large Language model
quick red fox jumped
over
the
lazy

Serving Setup



Considerations



Trade-Offs

Inworld

model parallelism

Lower costs — Higher costs

Higher latency — Lower latency

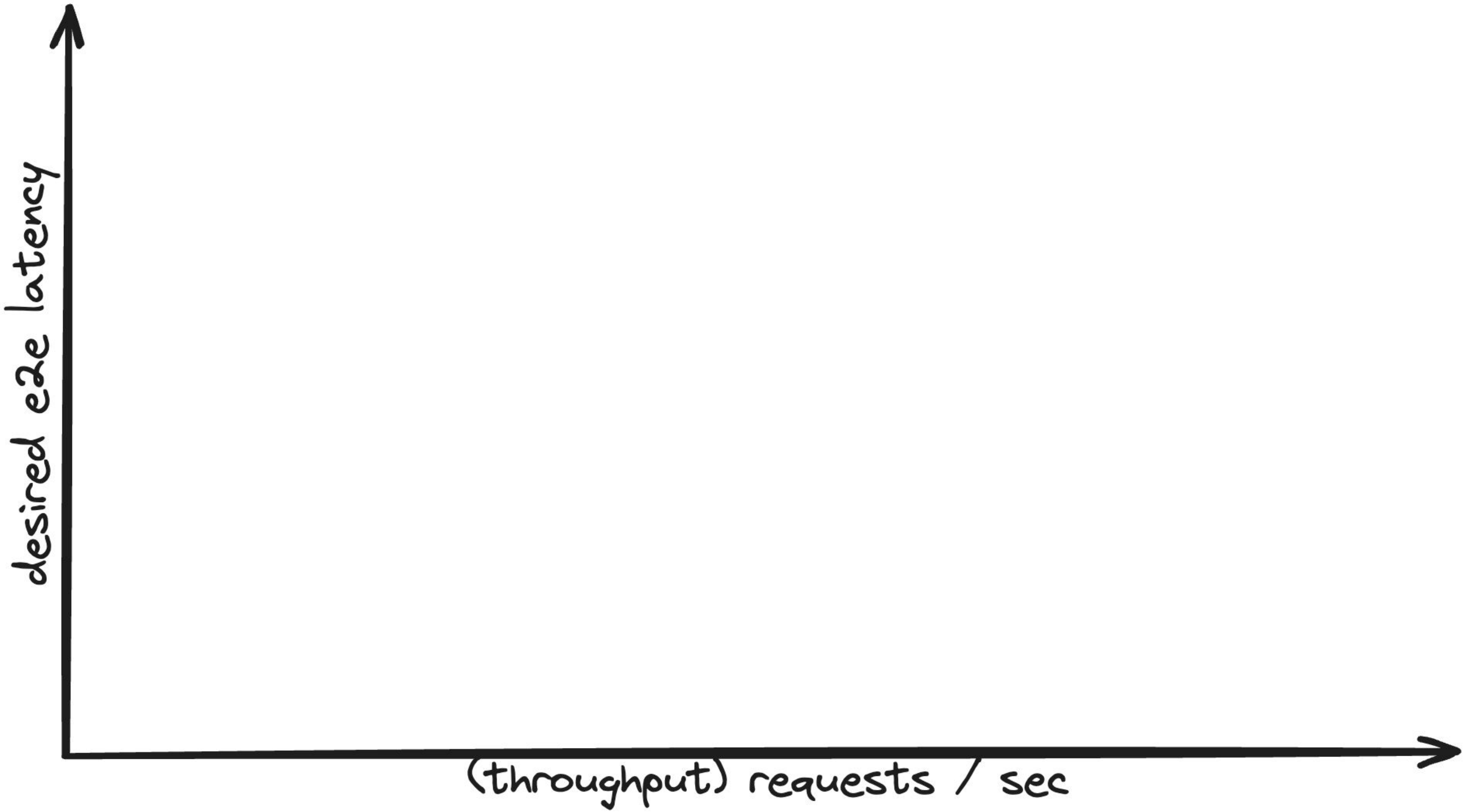
precision

Lower quality* — Higher quality

Lower latency — Higher latency

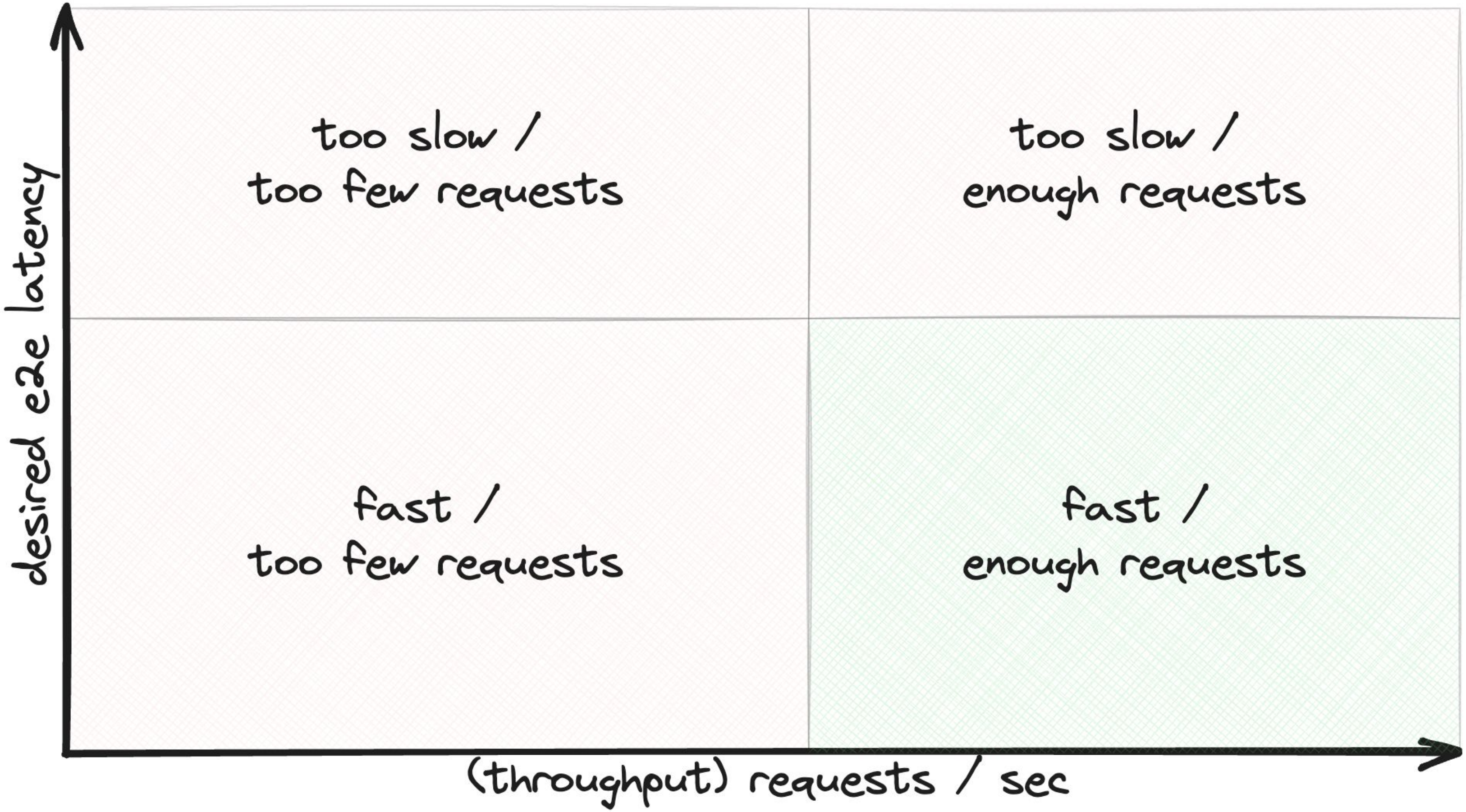
*it's tricky

Practical Approach

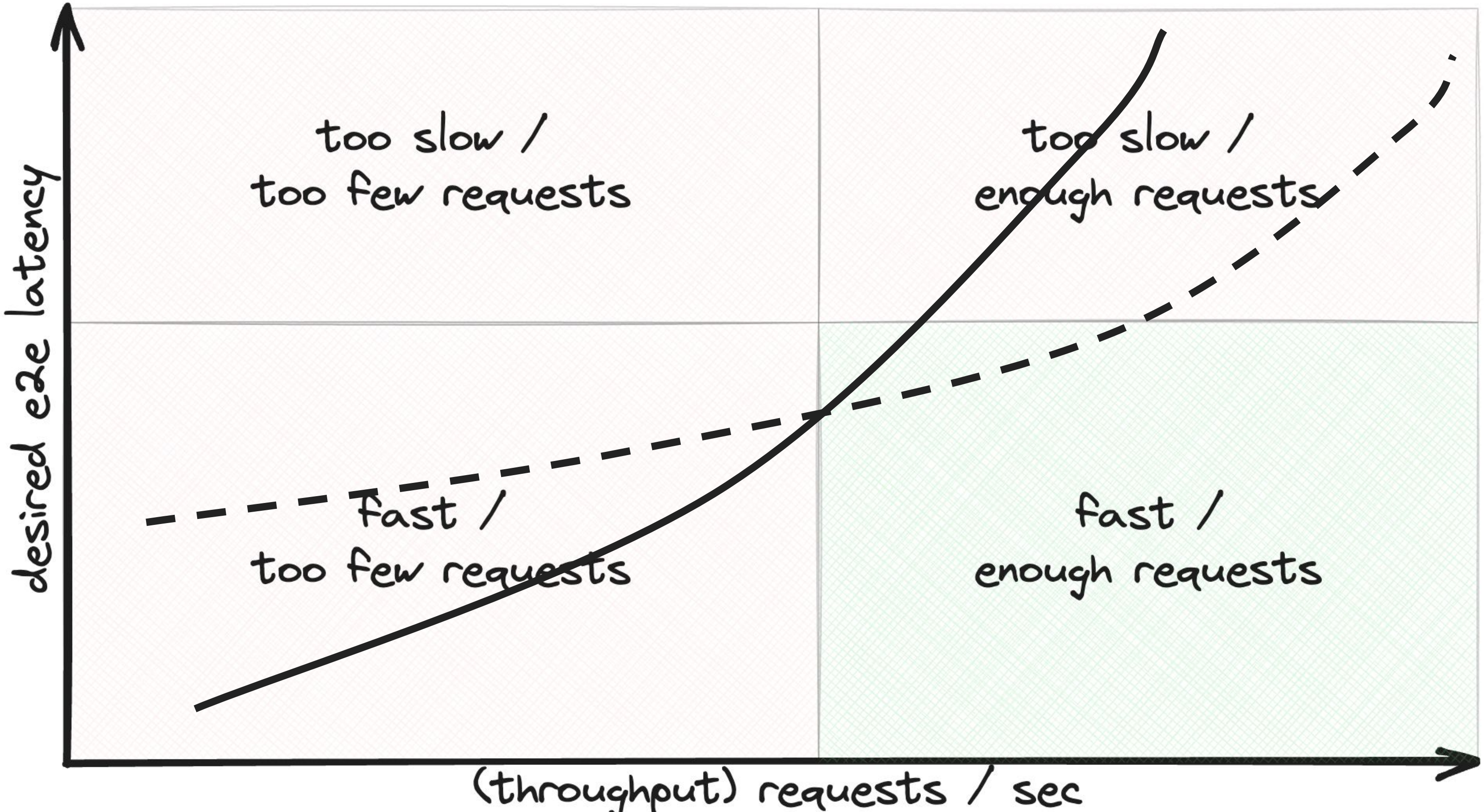


Practical Approach

Inworld



Practical Approach



Practical Approach

In order to ***roughly*** estimate the ***optimistically-biased*** single query cost:

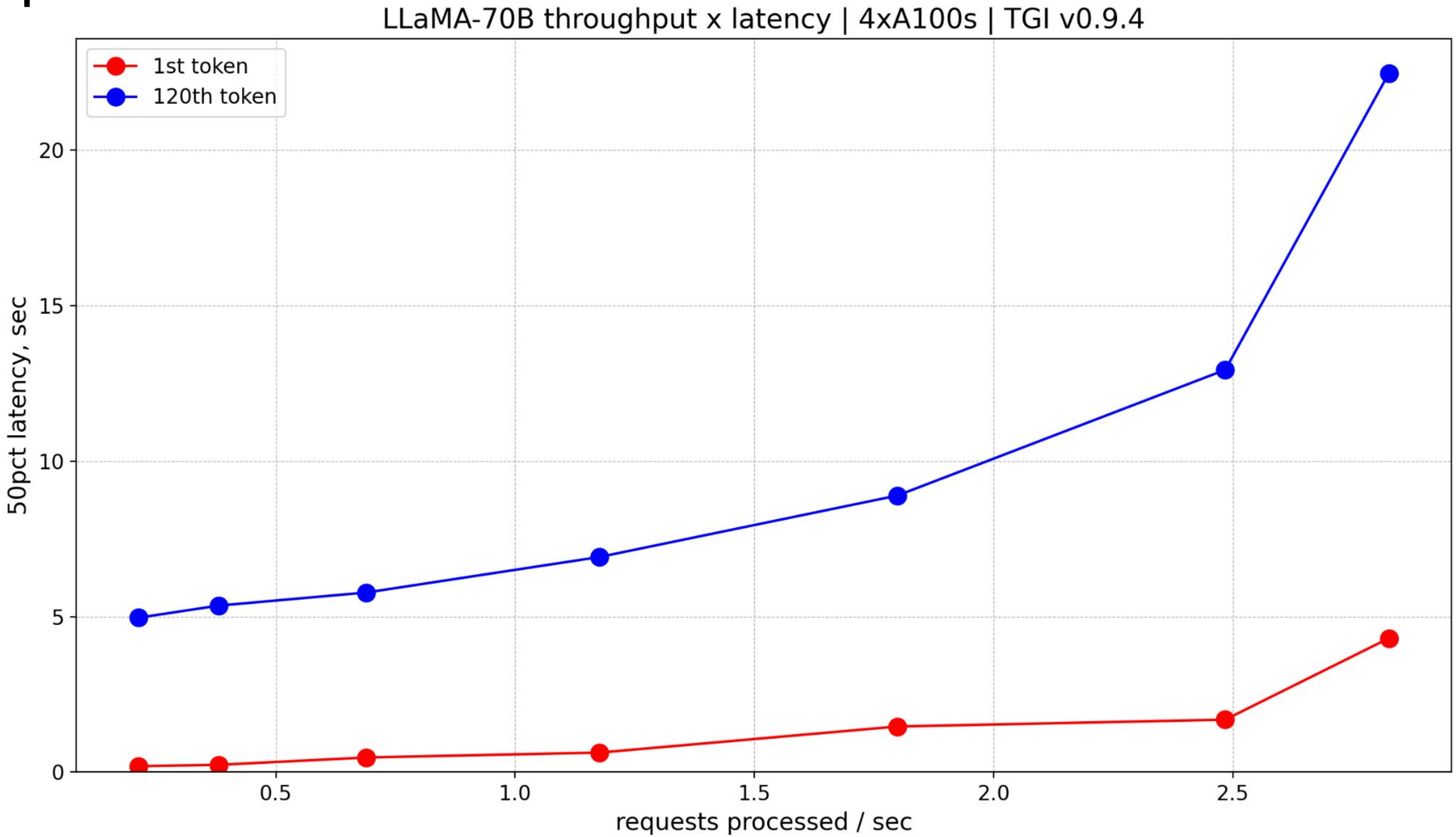
1. Figure out the ***number of input and output tokens*** for your typical prompt and response to be generated

2. Plot the ***throughput/latency curves*** for multiple serving configs (sharding on 1, 2, 4 GPUs, vary precision, etc)

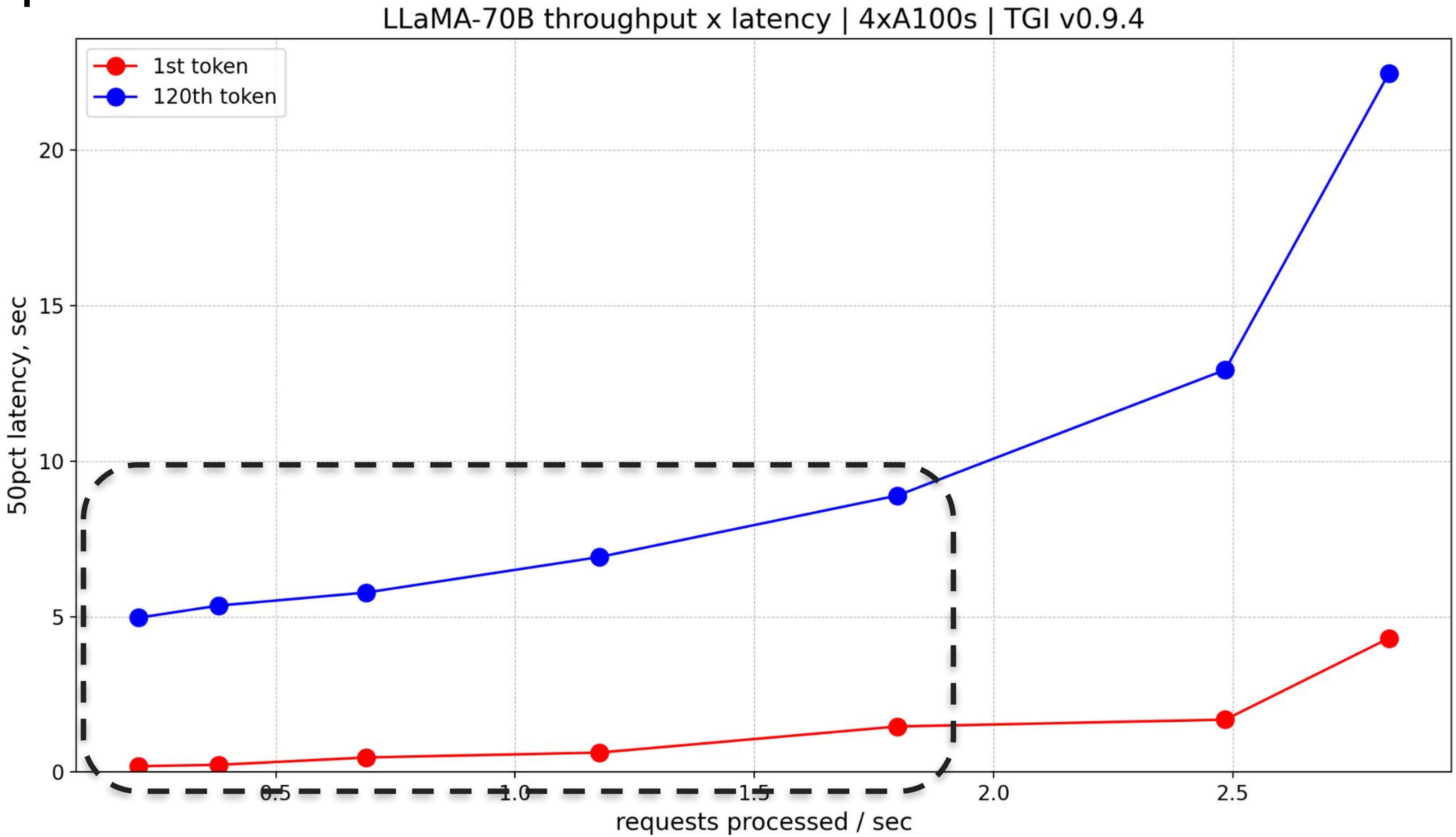
3. For the ***acceptable 1st token and/or e2e latency*** find out how many queries can be processed per second

4. Considering 100% utilization divide combined server's hourly price by total number of queries can be handled per hour → ***$/query***

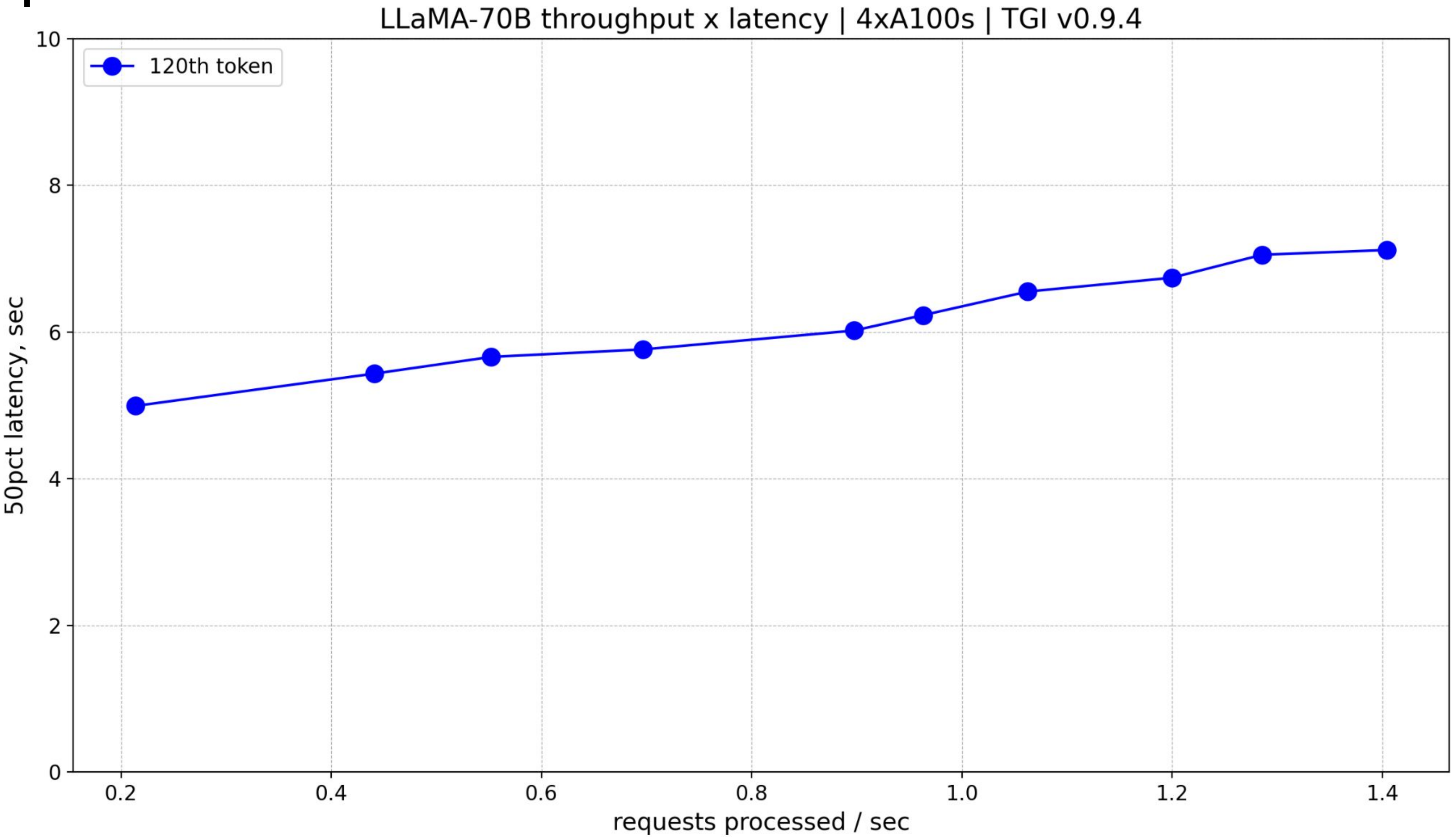
Experiment


LLaMA-70B throughput x latency | 4xA100s | TGI v0.9.4
1st token
120th token
50pct latency, sec
0
5
10
15
20
0.5
1.0
1.5
2.0
2.5
requests processed / sec

Experiment


LLaMA-70B throughput x latency | 4xA100s | TGI v0.9.4
1st token
120th token
50pct latency, sec
0
5
10
15
20
0.5
1.0
1.5
2.0
2.5
requests processed / sec

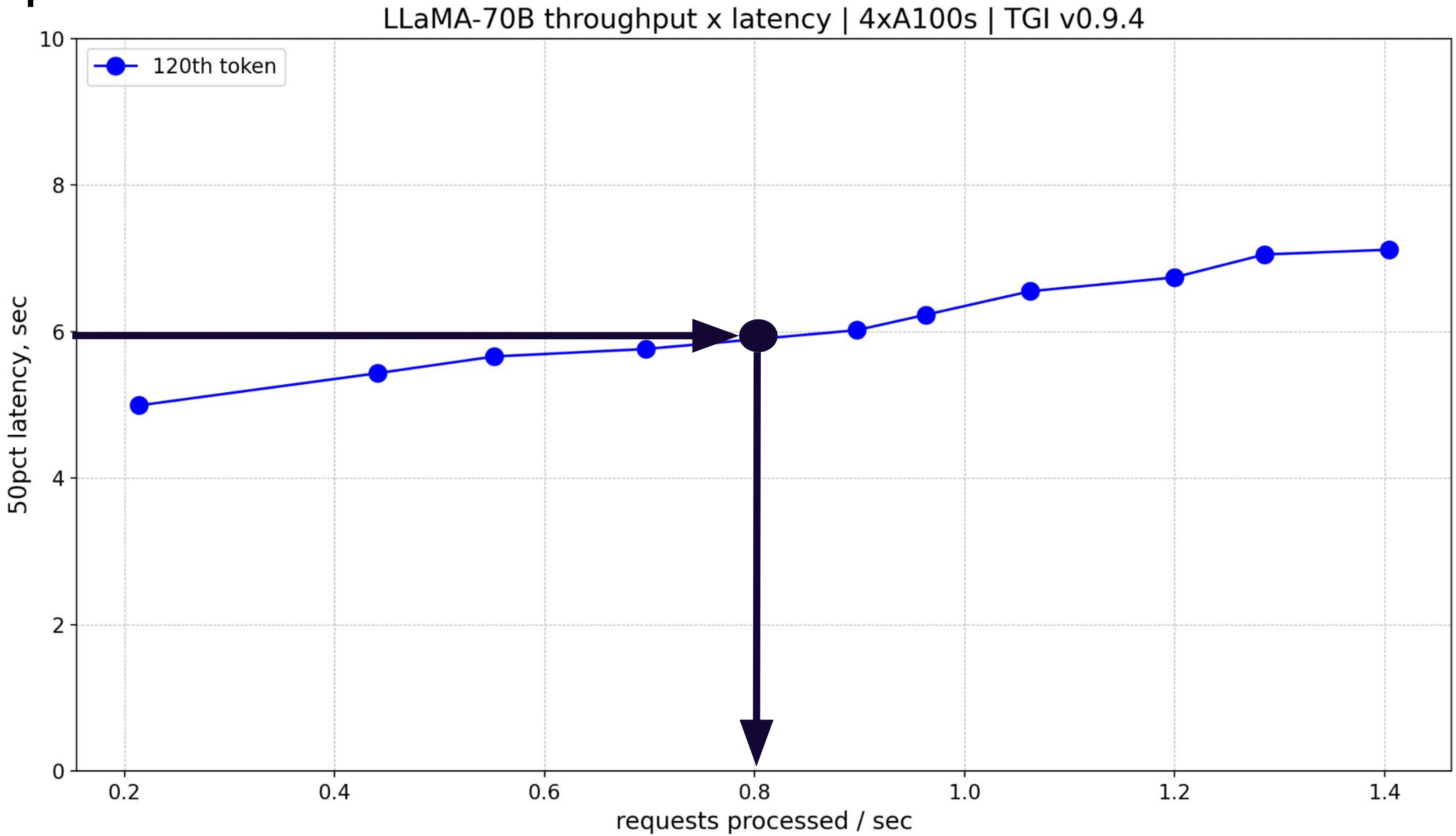
Experiment



LLaMA-70B throughput x latency | 4xA100s | TGI v0.9.4
120th token
50pct latency, sec
requests processed / sec
0
2
4
6
8
10
0.2
0.4
0.6
0.8
1.0
1.2
1.4

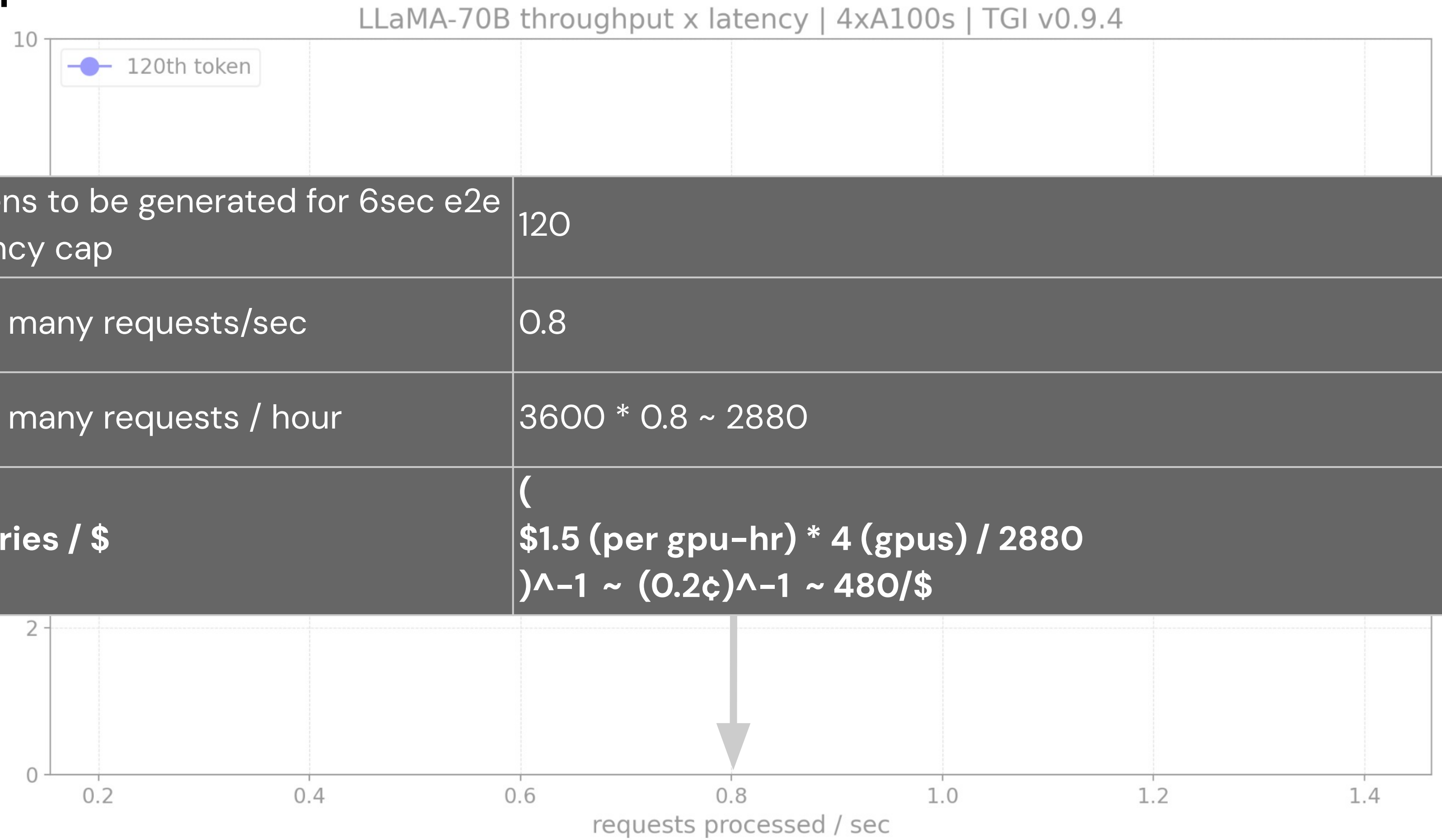
Experiment

Inworld



Experiment

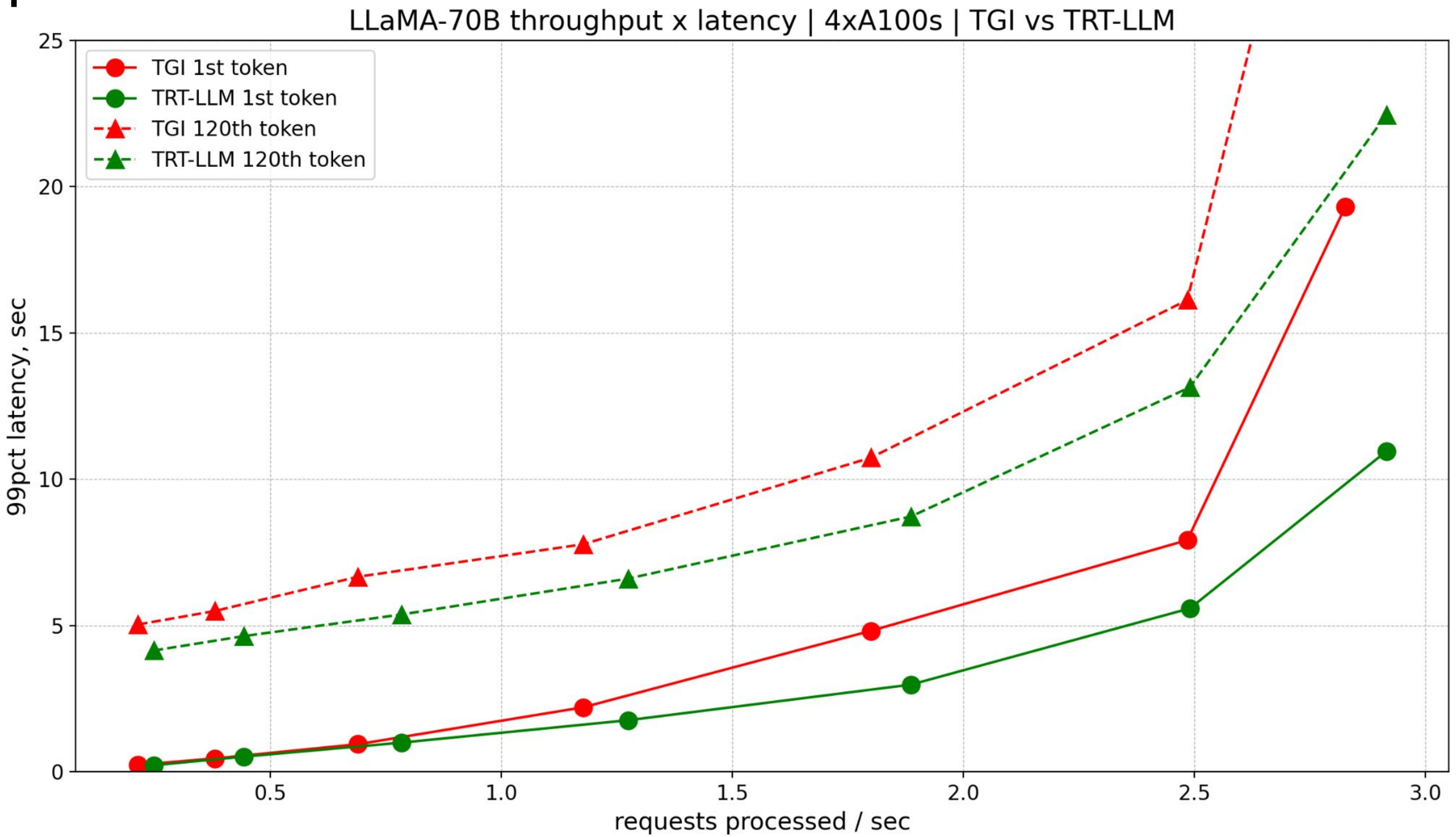
Inworld



Tokens to be generated for 6sec e2e latency cap	120
How many requests/sec	0.8
How many requests / hour	3600 * 0.8 ~ 2880
Queries / $	**($1.5 (per gpu-hr) * 4 (gpus) / 2880)^-1 ~ (0.2¢)^-1 ~ 480/$**

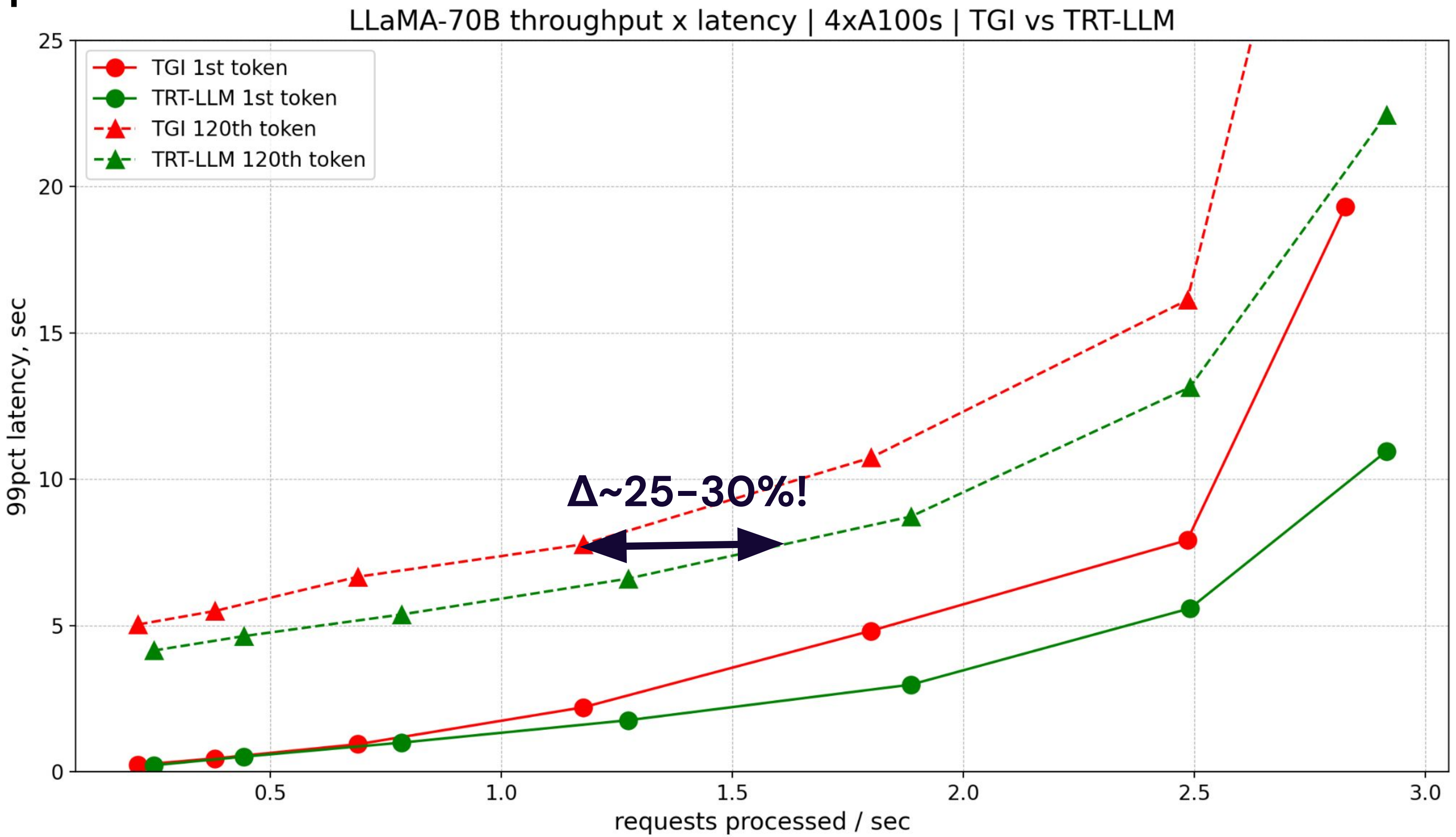
Experiment

Inworld



Experiment

Inworld



NVIDIA TRT-LLM

TensorRT-LLM Optimizing LLM Inference

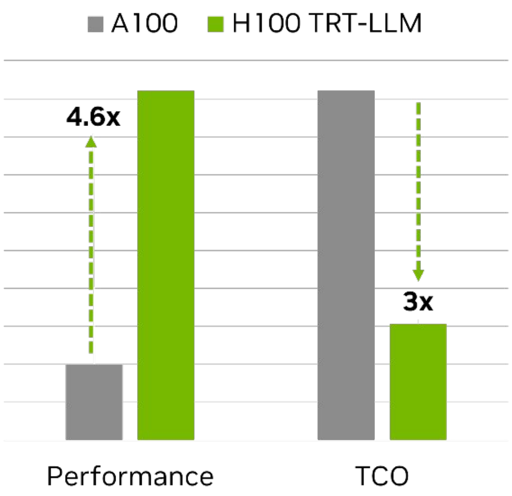
SoTA Performance for Large Language Models for Production Deployments

Challenges: LLM performance is crucial for real-time, cost-effective, production deployments. Rapid evolution in the LLM ecosystem, with new models & techniques released regularly, requires a performant, flexible solution to optimize models.

TensorRT-LLM is an **open-source** library to **optimize inference performance** on the latest **Large Language Models** for NVIDIA GPUs. It is built on FasterTransformer and TensorRT with a simple Python API for defining, optimizing, & executing LLMs for inference in production.

SoTA Performance

Leverage TensorRT compilation & kernels from FasterTransformers, CUTLASS, OAI Triton, ++



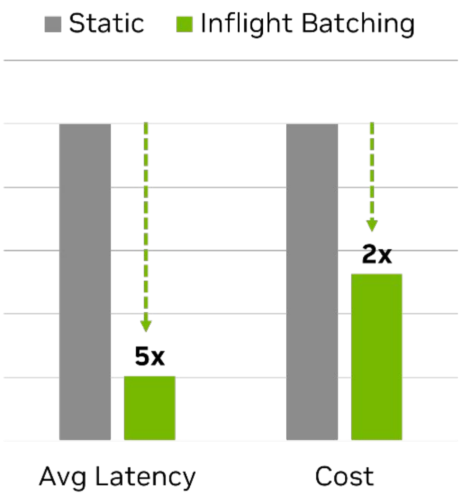
Ease Extension

Add new operators or models in Python to quickly support new LLMs with optimized performance

```
# define a new activation
def silu(input: Tensor) -> Tensor:
    return input * sigmoid(input)

#implement models like in DL FWs
class LlamaModel(Module)
  def __init__(…)
    self.layers = ModuleList([…])

  def forward (…)
    hidden = self.embedding(…)

    for layer in self.layers:
      hidden_states = layer(hidden)

    return hidden
```

LLM Batching with Triton

Maximize throughput and GPU utilization through new scheduling techniques for LLMs



Numbers are preliminary based on internal evaluation on Llama 7B on H100

Now Available!
TensorRT-LLM v0.8

- New models
  - Mixtral, BART, mBART, FairSeq NMT, Whisper, Mamba, Nougat, Qwen-VL, RoBERTa, Skywork

- New Features
  - Speculative Decoding
  - StreamingLLM support for LLaMA
  - Python bindings for GPTManager & GPTSession
  - Chunked context support

- New *X*QA kernel
  - Up to 2x improvement on models with GQA & MQA

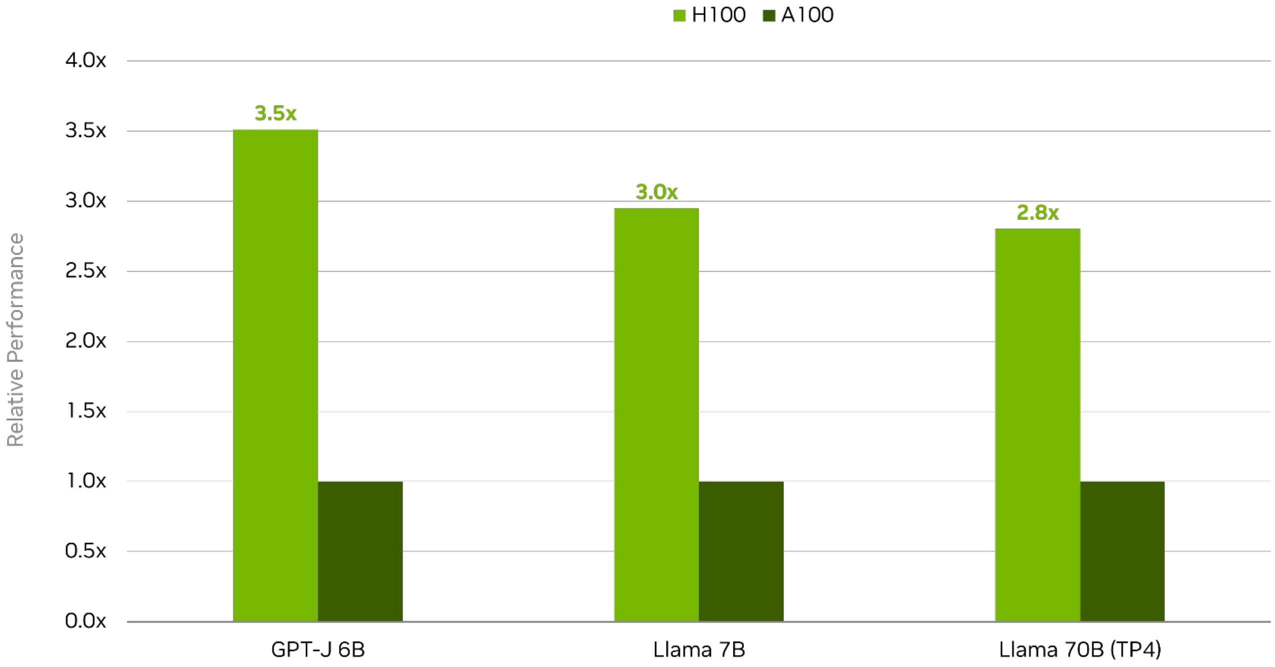
For the full list of improvements, please see the release notes


Versions 0.8.0

- Model Support
  - Phi-1.5/2.0
  - Mamba support (see examples/mamba/README.md)
    - The support is limited to beam width = 1 and single-node single-GPU
  - Nougat support (see examples/multimodal/README.md#nougat)
  - Qwen-VL support (see examples/qwenvl/README.md)
  - RoBERTa support, thanks to the contribution from @erenup
  - Skywork model support
  - Add example for multimodal models (BLIP with OPT or T5, LlaVA)
- Features
  - Chunked context support (see docs/source/gpt_attention.md#chunked-context)
  - LoRA support for C++ runtime (see docs/source/lora.md)
  - Medusa decoding support (see examples/medusa/README.md)
  - StreamingLLM support for LLaMA (see docs/source/gpt_attention.md#streamingllm)
  - Support for batch manager to return logits from context and/or generation phases
    - Include support in the Triton backend
  - Support AWQ and GPTQ for QWEN
  - Support ReduceScatter plugin
  - Support for combining `repetition_penalty` and `presence_penalty` #274
  - Support for `frequency_penalty` #275
  - OOTB functionality support:
    - Baichuan
    - InternLM
    - Qwen
    - BART
  - LLaMA
    - Support enabling INT4-AWQ along with FP8 KV Cache
    - Support BF16 for weight-only plugin
  - Baichuan
    - P-tuning support

TensorRT-LLM Performance Across Architectures

End-to-End Performance Using Inflight Batching & Triton



TensorRT-LLM v0.5.0 internal build. Triton with TensorRT-LLM inflight batching backend
DGX H100 FP8 & DGX A100 FP16
CNN Daily Mail dataset. Varying max concurrency. SOL serving scenario
TPN = Tensor Parallel across N devices.

Implementing New Operators

Utilizing custom CUDA kernels

TensorRT-LLM can use custom kernels via "Plugins"

- Allows for peak performance on key ops (ex. MHA)
- Quickly improve any performance bottlenecks
- Any kernels from CUTLASS, OpenAI Triton, CUDA, & more
- Insert as layers directly into the TensorRT-LLM model
- Execution & management handled entirely by TensorRT

0. GPU Kernel
Compiled GPU kernel from CUTLASS, OAIT, or other

1. Define kernel config
Metadata on the kernel for plugin generation

2. Generate the plugin!
TensorRT-LLM auto generates the plugins

TensorRT-LLM Plugin | `functional.py` layers

3. Embed!
Use the generated layers in the model definition

Q & A