
LLM Optimization Libraries: Inference & Training Phases

2110572: Natural Language Processing Systems

Assoc. Prof. Peerapon Vateekul, Ph.D.
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

Credit: TA.Pluem & TA.Knight



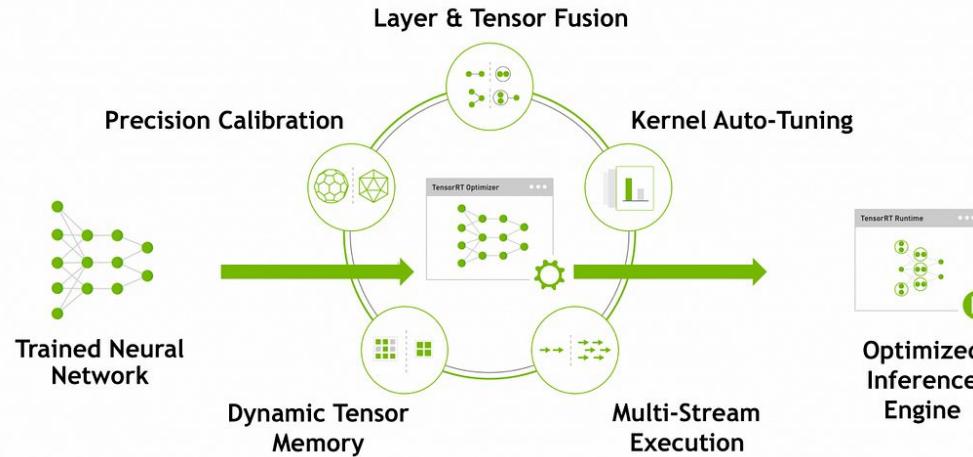
Outline

- Only inference
 - 1) TensorRT-LLM + Triton Inference Server [\[demo\]](#)
 - 2) NVIDIA Dynamo [\[resource\]](#)
- Supporting fine-tuning (training) & inference
 - 3) Unsloth (single GPU - free, but multiple GPUs - paid) [\[demo\]](#)
 - 4) DeepSpeed (multiple GPUs - all free, but more complicated than Unsloth) [\[resource\]](#)
- Conclusion

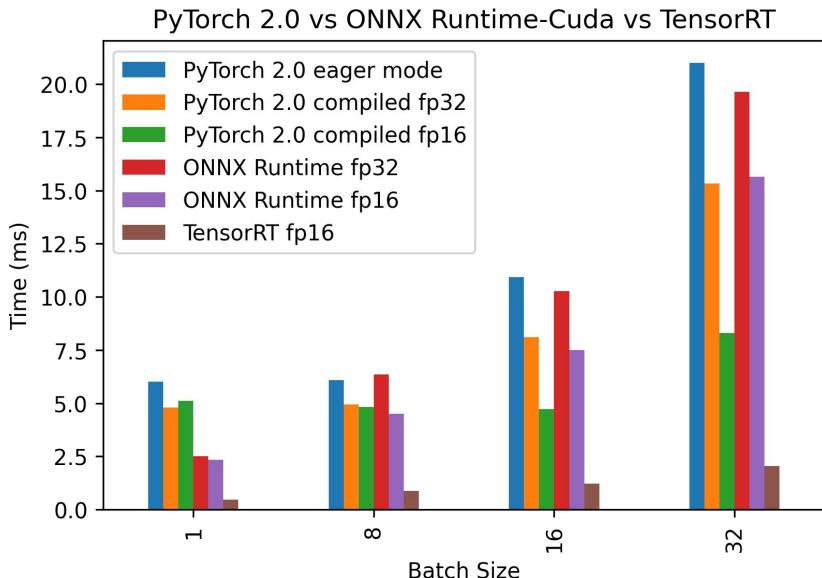
1) TensorRT-LLM + NVIDIA Triton Server [inference]

TensorRT: What is it?

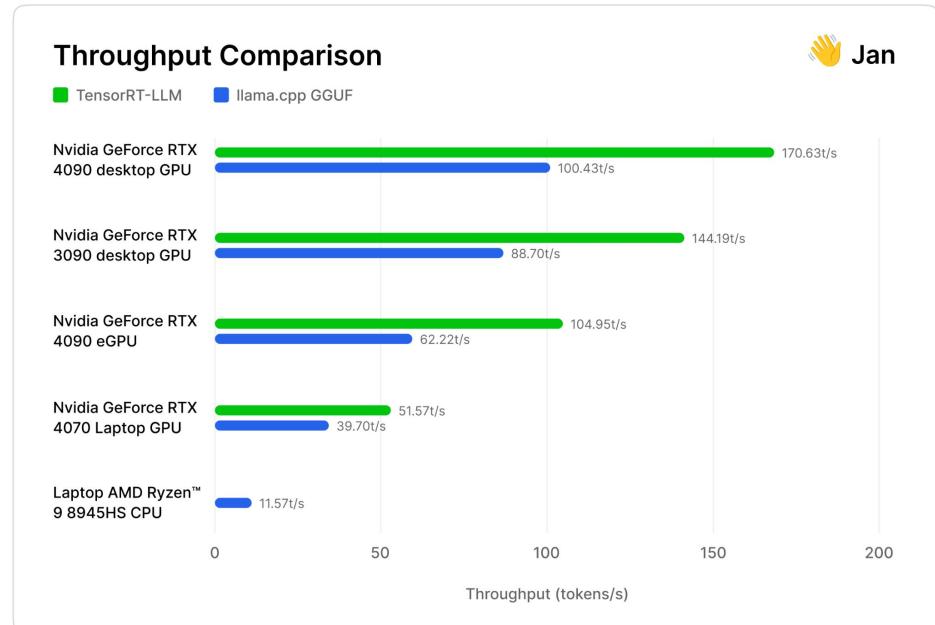
TensorRT is a library developed by NVIDIA [for faster inference on NVIDIA GPUs](#). It is built on CUDA, NVIDIA's parallel programming model.



TensorRT: Its performance



a 3090Ti with a ResNet



Mistral 7b



TensorRT-LLM: Features

1. Quantization
2. In-flight Batching
3. Chunked Context
4. KV cache reuse
5. Speculative Sampling

BF16 stands for bfloat16 (short for Brain Floating Point 16-bit), a 16-bit floating point format primarily designed to speed up AI/ML training while maintaining accuracy close to 32-bit floating point (FP32).

1. Quantization

| |
|----------------------------------|
| 12345 = 1.2345 × 10 ⁴ |
| • Mantissa = 1.2345 |
| • Exponent = 4 |
| • Base = 10 (in decimal) |

| Common Floating-Point Formats | | | |
|-------------------------------|------------|---------------|--|
| Format | Total Bits | Size in Bytes | Description |
| FP32 | 32 bits | 4 bytes | Standard for training, high precision |
| FP16 | 16 bits | 2 bytes | Half-precision, used in mixed-precision training |
| BF16 | 16 bits | 2 bytes | Similar dynamic range as FP32, lower precision |
| FP64 | 64 bits | 8 bytes | Double-precision, not commonly used in DL |

TensorRT-LLM supports running LLMs on various numerical precision modes.

- FP32, FP16 and BF16
- INT8 SmoothQuant (W8A8) [1]: SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [2024]
- INT4 and INT8 Weight-Only (W4A16 and W8A16)
- GPTQ and AWQ (W4A16) [2, 3]: (1) GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers [2022], (2) AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [2023]
- FP8 (Hopper GPUs only)
- NVFP4 (Blackwell GPUs only)

[1] <https://arxiv.org/abs/2211.10438>

[2] <https://arxiv.org/abs/2210.17323>

[3] <https://arxiv.org/abs/2306.00978>

| Term | Explanation |
|-------|---|
| W | Weights bit precision (number of bits used to represent model weights) |
| A | Activations bit precision (number of bits used to represent inputs/outputs of layers) |
| W4A16 | Weights are 4-bit, Activations are 16-bit |
| W8A16 | Weights are 8-bit, Activations are 16-bit |

2. In-flight Batching (IB): assume batch size = 4

w/ IB

| T_1 | T_2 | T_3 | T_4 | T_5 | T_6 | T_7 | T_8 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| S_1 | S_1 | S_1 | S_1 | | | | |
| S_2 | S_2 | S_2 | | | | | |
| S_3 | S_3 | S_3 | S_3 | | | | |
| S_4 | S_4 | S_4 | S_4 | S_4 | | | |

| T_1 | T_2 | T_3 | T_4 | T_5 | T_6 | T_7 | T_8 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| S_1 | S_1 | S_1 | S_1 | S_1 | END | | |
| S_2 | END |
| S_3 | S_3 | S_3 | S_3 | S_3 | END | | |
| S_4 | END |

w/o IB

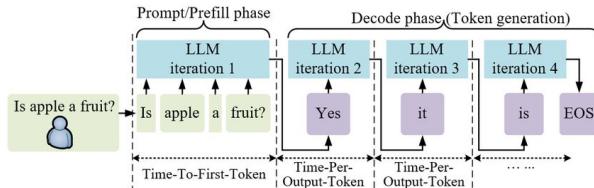
| T_1 | T_2 | T_3 | T_4 | T_5 | T_6 | T_7 | T_8 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| S_1 | S_1 | S_1 | S_1 | | | | |
| S_2 | S_2 | S_2 | | | | | |
| S_3 | S_3 | S_3 | S_3 | | | | |
| S_4 | S_4 | S_4 | S_4 | S_4 | | | |

| T_1 | T_2 | T_3 | T_4 | T_5 | T_6 | T_7 | T_8 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| S_1 | S_1 | S_1 | S_1 | S_1 | END | S_6 | S_6 |
| S_2 | END |
| S_3 | S_3 | S_3 | S_3 | S_3 | END | S_5 | S_5 |
| S_4 | S_7 |

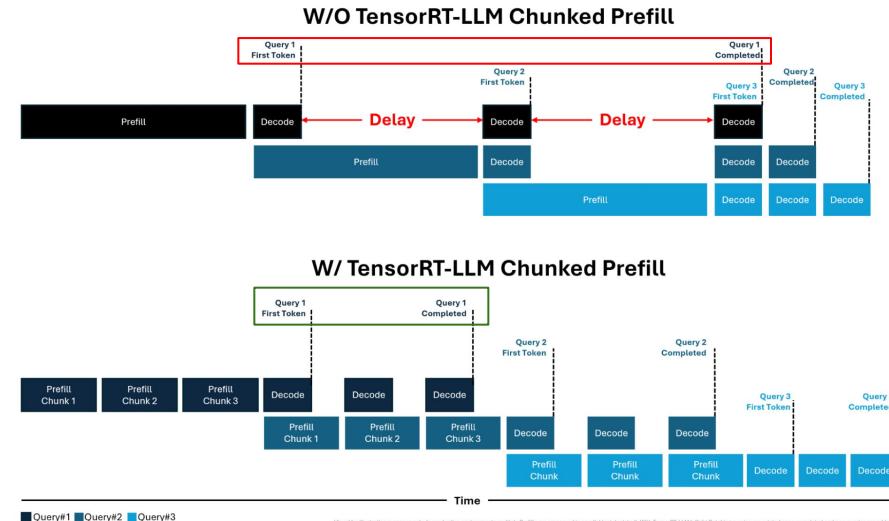
3. Chunked Context

The **chunked context** feature splits the context into several manageable chunks rather than processing all context tokens at once. **Chunked prefill** optimizes the initial embedding phase by batching prefill computations across requests, boosting GPU efficiency—especially with in-flight batching and variable-length inputs.

- Chunked context ensures long prompts won't cause Out-Of-Memory (OOM).
- Chunked prefill ensures many prompts can be processed efficiently.



| Stage | Description | Characteristics |
|---------|---|--|
| Prefill | Processes input tokens and computes KV cache. | Highly parallelized, efficient GPU utilization. |
| Decode | Generates output tokens one at a time based on previous tokens. | Sequential processing, potentially slower, memory-bound. |

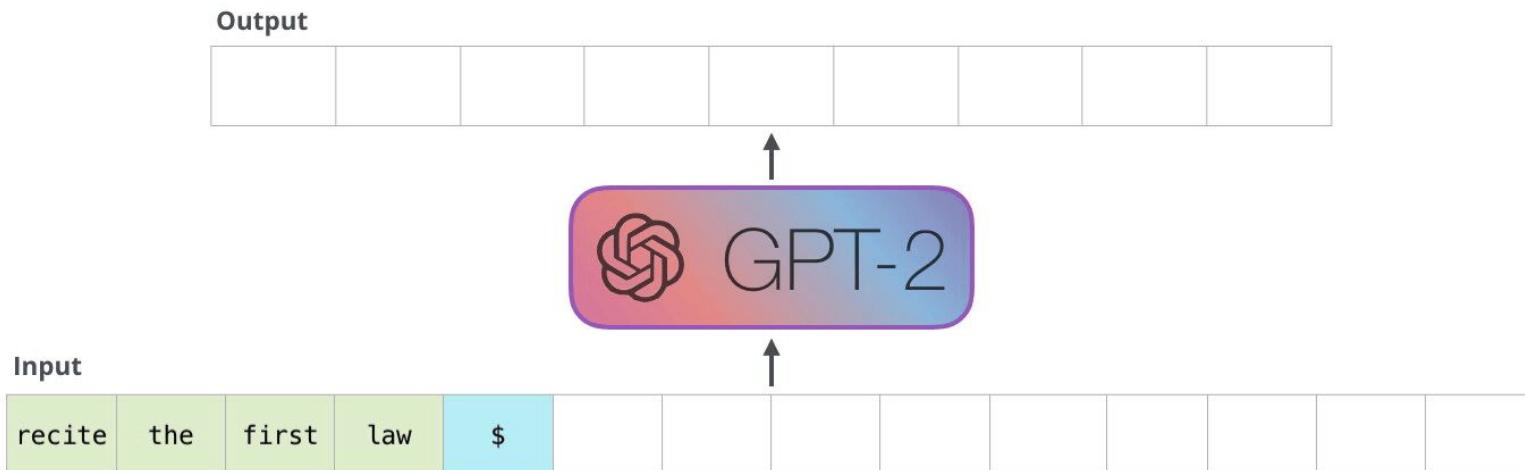


4. KV cache reuse

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

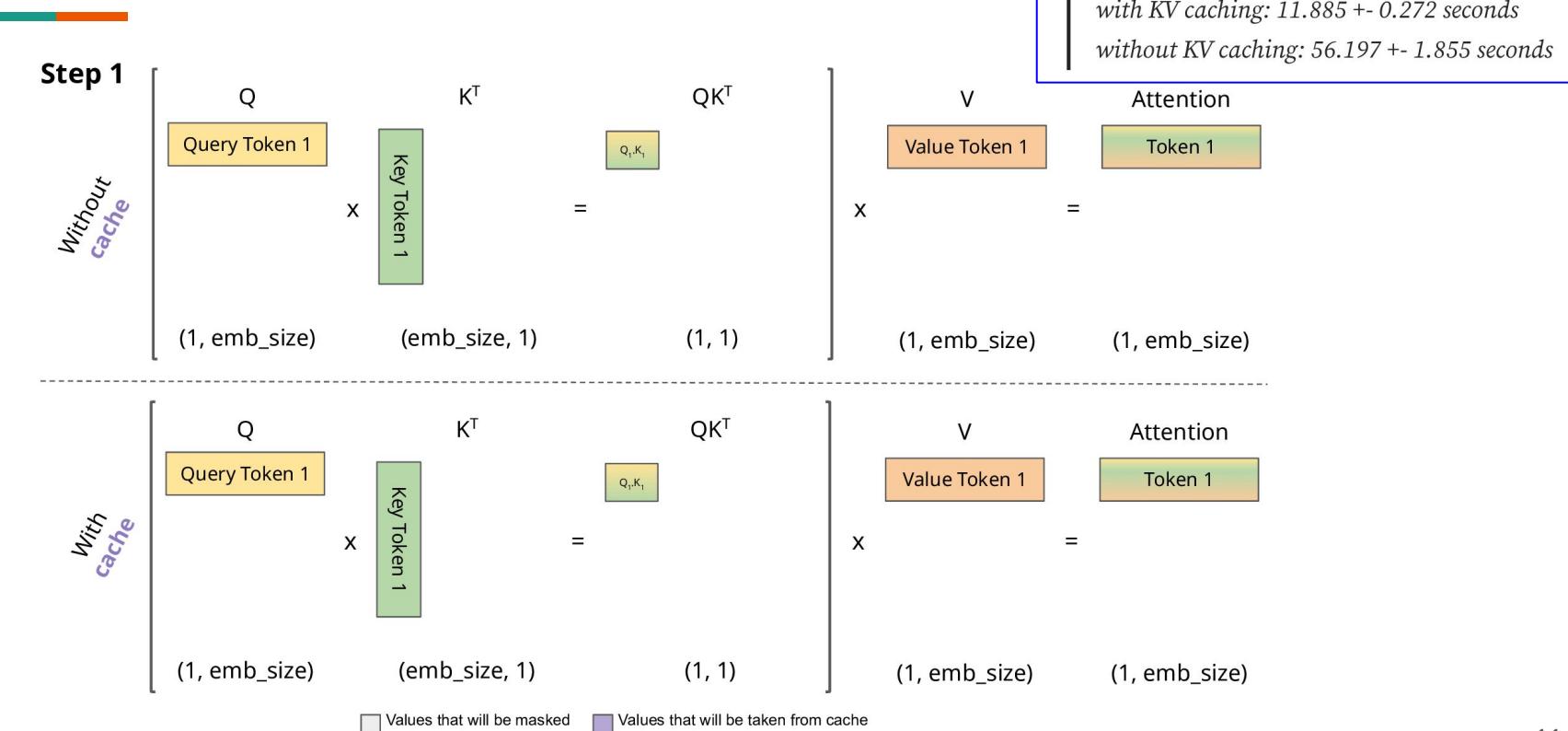
Reduces time-to-first-token by reusing KV cache from multiple requests with the same system prompts or multi-turn requests.

KV caching occurs during multiple token generation steps and only happens in the decoder (i.e., in decoder-only models like GPT, or in the decoder part of encoder-decoder models like T5). Models like BERT are not generative and therefore do not have KV caching.



4. KV cache reuse (cont.)

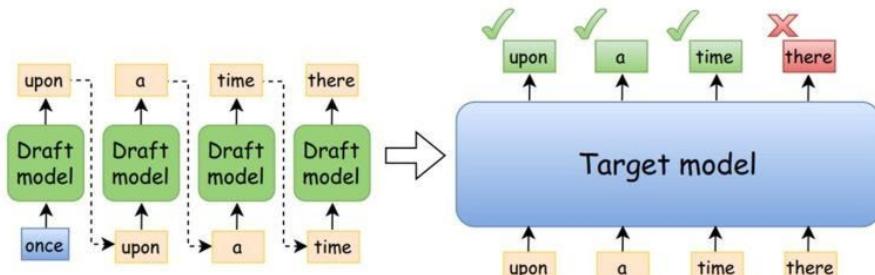
Reduces time-to-first-token by reusing KV cache from multiple requests with the same system prompts or multi-turn requests.



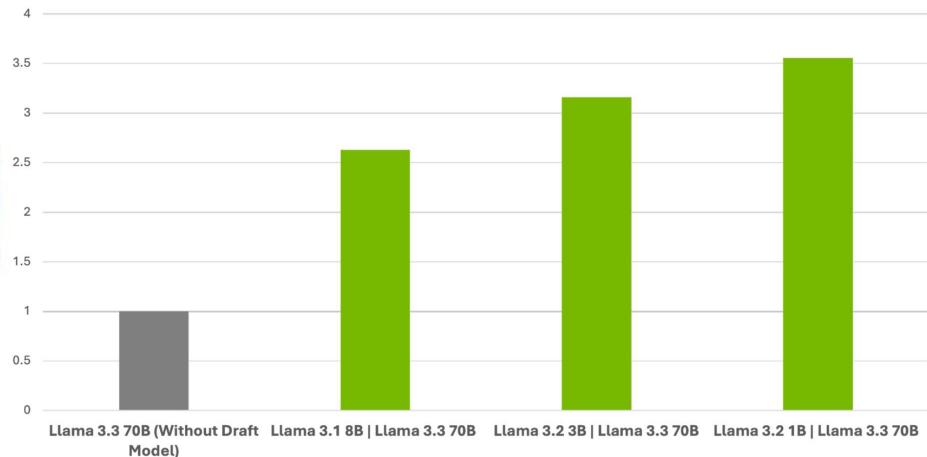
5. Speculative Sampling

Speculative Sampling (or Speculative decoding) uses a smaller LM to predict multiple future tokens (draft tokens). The predicted continuation is then verified by the larger LLM we are hosting. If it is accepted, use the draft tokens as the output and continue generating the next token.

- Draft model = generator
- Target model = verifier



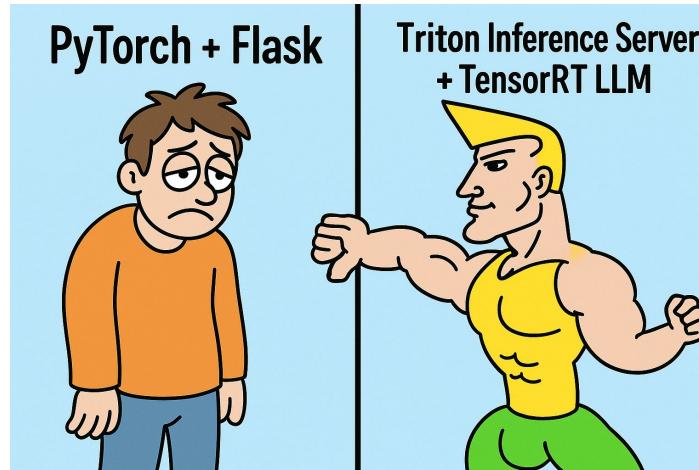
Up to 3.6x Increase in Llama 3.3 70B Token Generation



Example (demo from lab's server)

Objective: Deploy Qwen2.5 7B model using Triton Inference Server with TensorRT-LLM backend.

Required GPU with 24GB VRAM (at least 3090 or 4090)



Pull a Triton Inference Server from Nvidia NGC. This demo uses 25.03-pyt-python-py3.

Step 0: Load TensorRT-LLM docker

The screenshot shows the Nvidia NGC Container Catalog interface. The top navigation bar includes 'Containers > Triton Inference Server' and a 'Get Container' button. The main content area is titled 'Triton Inference Server' and displays the following information:

- Overview:** Shows the NVIDIA logo and the text 'TRITON INference SERVER'.
- Tags:** The 'Tags' tab is selected. A note says 'A public key is required to validate the signed images below.' and links to 'View all public keys maintained by NVIDIA'.
 - 25.03-trtllm-python-py3**: SIGNED. Published on 04/05/2025 at 6:41 AM, size 17.21 GB, 2 architectures. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-trtll...
 - 25.03-pyt-python-py3**: SIGNED. Published on 04/05/2025 at 6:41 AM, size 7.73 GB, 2 architectures. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-pyt-py3...
 - 25.03-py3-sdk**: SIGNED. Published on 04/05/2025 at 6:41 AM, size 7.86 GB, 2 architectures. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-py3-sd...
 - 25.03-vllm-python-py3**: SIGNED. Published on 04/05/2025 at 6:40 AM, size 11.69 GB, 1 architecture. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-vllm-py3...
 - 25.03-py3**: SIGNED. Published on 04/05/2025 at 6:40 AM, size 9.92 GB, 2 architectures. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-py3...
 - 25.03-py3-min**: SIGNED. Published on 04/05/2025 at 6:39 AM, size 5.43 GB, 2 architectures. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-py3-mi...
 - 25.03-py3-igpu-sdk**: SIGNED. Published on 04/05/2025 at 6:36 AM, size 4.62 GB, 1 architecture. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-py3-ig...
 - 25.03-py3-igpu-min**: SIGNED. Published on 04/05/2025 at 6:35 AM, size 2.48 GB, 1 architecture. Docker hub link: nvcr.io/nvidia/tritonserver:25.03-py3-ig...
- Layers:** Shows the layers of the container.
- Security Scanning:** Shows the security scan results.
- Related Collections:** Shows related collections like Signed Images, Automatic Speech Recognition, Automotive / Transportation, DL, Inference, Infrastructure Software, and NVIDIA AI.

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tritonserver/tags>

Step 1: Load model Qwen 7B

Download model weights

```
root@a156c29a2adf:/app/examples/qwen# git clone https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct ./tmp/Qwen2.5/0.5B
```



Step 2: Convert the model based on configuration

```
for model_size in 0.5 7; do
    python convert_checkpoint.py --model_dir ./tmp/Qwen2.5/${model_size}B/ --output_dir ./tllm_checkpoint_1gpu_fp16_${model_size}B --dtype float16
    trtllm-build --checkpoint_dir ./tllm_checkpoint_1gpu_fp16_${model_size}B --output_dir ./tmp/Qwen2.5/${model_size}B/trt_engines/fp16/1-gpu \
    --gemm_plugin float16 --max_seq_len 2048 --max_batch_size 8 --max_beam_width 1 --reduce_fusion enable --use_paged_context_fmha enable --multiple_profiles enable
done
```

```
# for SmoothQuant quantization
python3 convert_checkpoint.py --model_dir ./tmp/Qwen2.5/7B/ --output_dir ./tllm_checkpoint_1gpu_sq --dtype float16 --smoothquant 0.5
trtllm-build --checkpoint_dir ./tllm_checkpoint_1gpu_sq --output_dir ./engine_outputs --gemm_plugin int8
```



Step 3: Configure Triton server & Launch the server

```
cd triton_model_repo
cp -R /app/all_models/inflight_batcher_llm/* ${MODEL_FOLDER}
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/ensemble/config.pbtxt triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},logits_datatype:${LOGITS_DATATYPE}
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/preprocessing/config.pbtxt tokenizer_dir:${TOKENIZER_DIR},triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},preprocessing_instance_count:${INSTANCE_COUNT}
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/tensorrt_llm/config.pbtxt triton_backend:tensorrtllm,triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},decoupled_mode:${DECOPLED_MODE},max_queue_delay_microseconds:${MAX_QUEUE_DELAY_MS},max_queue_size:${MAX_QUEUE_SIZE}
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/postprocessing/config.pbtxt tokenizer_dir:${TOKENIZER_DIR},triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},postprocessing_instance_count:${INSTANCE_COUNT},max_queue_size:${MAX_QUEUE_SIZE}
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/tensorrt_llm_bls/config.pbtxt triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},decoupled_mode:${DECOPLED_MODE},bfs_instance_count:${INSTANCE_COUNT},logits_datatype:${LOGITS_DATATYPE}
python3 /app/scripts/launch_triton_server.py --world_size=1 --model_repo=triton_model_repo --http_port=14524
```

Launch server in the last line!

```
✓ triton_model_repo
  > ensemble
  > postprocessing
  > preprocessing
  > tensorrt_llm
  > tensorrt_llm_bls
$ run_triton_server.sh
```

These lines just fill the config files with your specified config.

Step 4: Start the demo

4.1) KV Caching (code): 9.45 sec → 8.86 sec

KV Caching

```
import time

start = time.time()
response = generate_text(api_url, model, prompts[0])
end = time.time()
print(f'{end - start:.2f} sec')

start = time.time()
response = generate_text(api_url, model, prompts[0])
end = time.time()
print(f'{end - start:.2f} sec')
```

9.45 sec
8.86 sec

4.2) In-flight Batching (code): w/o IB 50.45 sec → w/ IB 9.72 sec (async)

No inflight batching

```
import time

total_time = 0
for i, prompt in enumerate(prompts):
    start = time.time()
    response = generate_text(api_url, model, prompt)
    end = time.time()
    total_time += end - start
    print(f'Iteration {i}', f'{end - start:.2f} sec')

print()
print(f'Total time: {total_time:.2f} sec')
```

With Inflight batching

```
import asyncio
import aiohttp
import time

async def generate_text(api_url, model, prompt, max_tokens=500, speculative_sampling=False, num_draft_tokens=5):
    data = {
        "text_input": prompt,
        "max_tokens": max_tokens,
        "bad_words": "",
        "stop_words": ""
    }

    if speculative_sampling:
        data["num_draft_tokens"] = num_draft_tokens
        data["use_draft_logits"] = False
        data["stream"] = False

    async with aiohttp.ClientSession() as session:
        async with session.post(f'{api_url}/v2/models/{model}/generate', json=data) as response:
            if response.status == 200:
                return await response.json()
            else:
                return {"error": await response.text()}
```

4.3) Chunk context (via configuration) Doesn't improve

```
ENABLE_KV_CACHE_REUSE=true  
BATCHING_STRATEGY=inflight_fused_batching  
ENABLE_CHUNKED_CONTEXT=false
```

4.4) Speculative sampling (via configuration) Doesn't improve

Step 1: Build a new engine with speculative decoding mode specified.

```
# For Speculative Sampling  
trtllm-build --checkpoint_dir ./tllm_checkpoint_1gpu_fp16_7B --output_dir ./tmp/Qwen2.5/7B/trt_engines/fp16/1-gpu-target \  
--gemm_plugin float16 --max_seq_len 2048 --max_batch_size 8 --max_beam_width 1 --reduce_fusion enable --use_paged_context_fmha enable --multiple_profiles enable  
| --max_draft_len=10 --speculative_decoding_mode draft_tokens_external
```

4.4 Speculative sampling (cont.)

Step 2: Slightly modify Triton config: draft & target models

```
✓ triton_model_repo_speculative_decoding
  > ensemble
  > postprocessing
  > preprocessing
  > tensorrt_llm
  > tensorrt_llm_bls
  > tensorrt_llm_draft
$ run_triton_server.sh
```

```
ENGINE_DIR=/app/examples/qwen/tmp/Qwen2.5/7B/trt_engines/fp16/1-gpu-target
DRAFT_ENGINE_DIR=/app/examples/qwen/tmp/Qwen2.5/0.5B/trt_engines/fp16/1-gpu
TENSORRT_LLM_DRAFT_MODEL_NAME="tensorrt_llm_draft"
TENSORRT_LLM_MODEL_NAME="tensorrt_llm"
TOKENIZER_DIR=/app/examples/qwen/tmp/Qwen2.5/7B
MODEL_FOLDER=/triton_model_repo_speculative_decoding
TRITON_MAX_BATCH_SIZE=4
INSTANCES_COUNT=4
MAX_QUEUE_DELAY_MS=10000
MAX_QUEUE_SIZE=0
FILL_TEMPLATE_SCRIPT=/app/tools/fill_template.py
DECOPLED_MODE=false
LOGITS_DATATYPE=TYPE_FP32
KV_CACHE_FREE_GPU_MEM_FRACTION=0.4
ENABLE_KV_CACHE_REUSE=true
MAX_TOKEN_IN_PAGED_KV_CACHE=2048*8
DRAFT_GPU_DEVICE_IDS=0
ENABLE_CHUNKED_CONTEXT=false

cp -R /app/all_models/inflight_batcher_llm/* ${MODEL_FOLDER}
mkdir -p tensorrt_llm_draft
cp -R ${MODEL_FOLDER}/tensorrt_llm/* ${MODEL_FOLDER}/tensorrt_llm_draft
```

```
## Speculative sampling ##
sed -i 's/name: "tensorrt_llm"/name: "tensorrt_llm_draft"/g' ${MODEL_FOLDER}/tensorrt_llm_draft/config.pbtxt
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/tensorrt_llm_bls/config.pbtxt triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},decoupled_mode:${DECOPLED_MODE},bls_instances_count:${INSTANCES_COUNT}
python3 ${FILL_TEMPLATE_SCRIPT} -i ${MODEL_FOLDER}/tensorrt_llm_draft/config.pbtxt triton_backend:tensorrt_llm,triton_max_batch_size:${TRITON_MAX_BATCH_SIZE},decoupled_mode:${DECOPLED_MODE},max_tokens_per_response:100
python3 /app/scripts/launch_triton_server.py --world_size=1 --model_repo=/triton_model_repo_speculative_decoding --http_port=14524 --tensorrt_llm_model_name "tensorrt_llm,tensorrt_llm_draft"
```

Add this into the BLS config

* BLS = Business logic script

```
,tensorrt_llm_model_name:${TENSORRT_LLM_MODEL_NAME},tensorrt_llm_draft_model_name:${TENSORRT_LLM_DRAFT_MODEL_NAME}
```



Must use KV cache (cannot disable KV cache option)

```
terminate called after throwing an instance of 'tensorrt_llm::common::Tl1mException'
  what(): [TensorRT-LLM][ERROR] Assertion failed: KV-cache-less is only supported for context (/workspace/tensorrt_llm/cpp/tensorrt_llm/plugins/gptAttentionPlugin/gptAttentionPlugin.cpp:1102)
1       0x7f47148f0bdd /usr/local/lib/python3.12/dist-packages/tensorrt_llm/libs/libnvinfer_plugin_tensorrt_llm.so(+0x8dbdd) [0x7f47148f0bdd]
```

Inference speed did not improve when the chunked context was enabled.
(Mistral) #331

Open



matichon-vultureprime opened on Feb 7, 2024

...

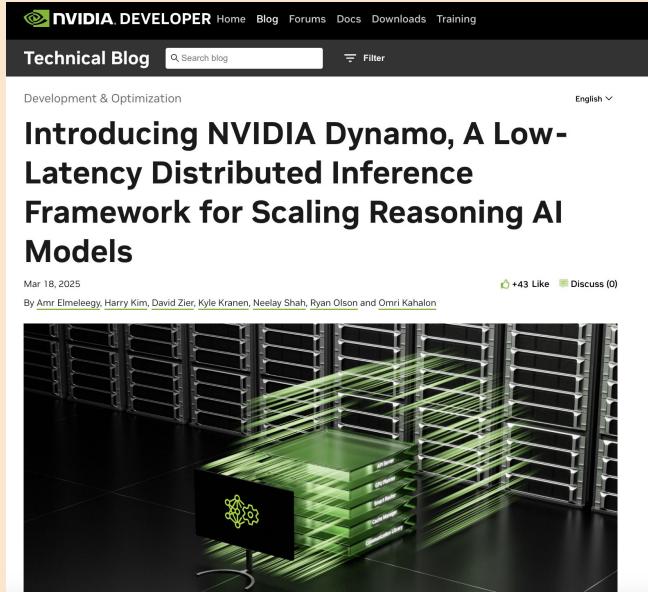
Assignees



Shixiaowei02

2) NVIDIA Dynamo [inference]

LLMs: NVIDIA Triton Server → NVIDIA Dynamo



<https://developer.nvidia.com/blog/introducing-nvidia-dynamo-a-low-latency-distributed-inference-framework-for-scaling-reasoning-ai-models/>

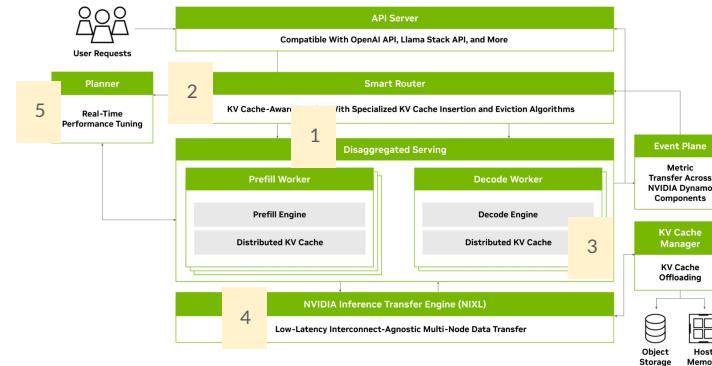
NVIDIA Dynamo

NVIDIA Dynamo is a high-throughput, low-latency inference framework designed for serving generative AI and reasoning models in **multi-node distributed environments**.

Dynamo is designed to be inference engine agnostic (**supports TRT-LLM, vLLM, SGLang or others**)

To enable large-scale distributed and disaggregated inference serving, Dynamo includes **five key features**.

1. Dynamo Disaggregated Serving
2. Dynamo Smart Router
3. Dynamo Distributed KV Cache Manager
4. NVIDIA Inference Transfer Library (NIXL)
5. Dynamo Planner



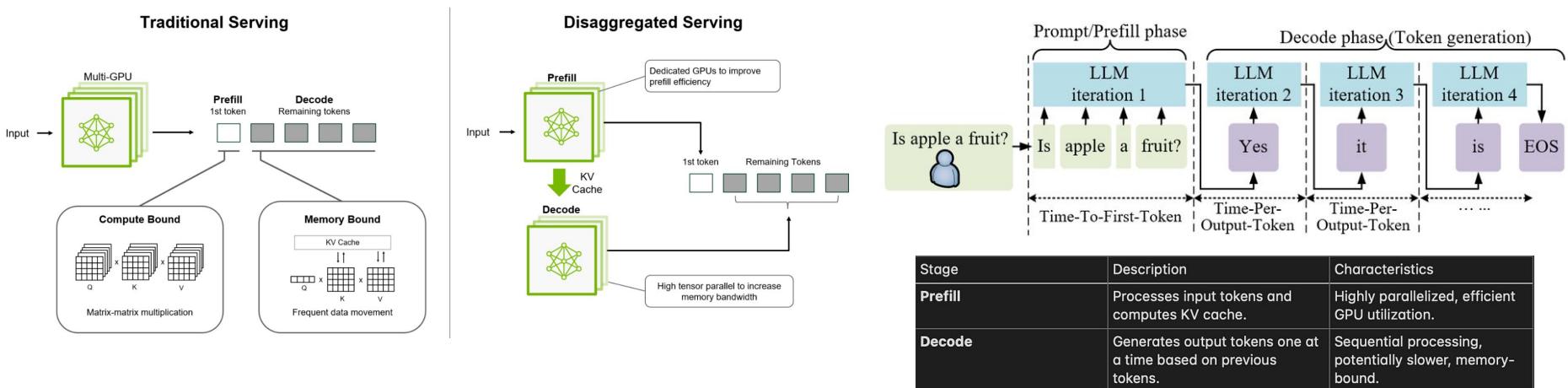
NVIDIA Triton Server vs. NVIDIA Dynamo

Key Differences Summary

| Feature | NVIDIA Triton Inference Server | NVIDIA DynaMo (Dynamic Model Orchestration) |
|--------------------------|---|---|
| Purpose | General model serving engine | Advanced LLM inference orchestrator |
| Built for | Multiple frameworks and use cases | High-throughput LLM serving |
| Model types | Any (CV, NLP, etc.) | LLMs (e.g., Mixtral, LLaMA, GPT-J, etc.) |
| Dynamic batching | <input checked="" type="checkbox"/> Yes | <input checked="" type="checkbox"/> More advanced (in-flight, priority aware) |
| Streaming support | <input checked="" type="checkbox"/> Yes (with TensorRT-LLM) | <input checked="" type="checkbox"/> Yes (optimized with memory paging) |
| KV cache & query routing | Basic via TensorRT-LLM backend | <input checked="" type="checkbox"/> Optimized (paged cache, preemption, scheduling) |
| Scale | Multi-GPU, multi-model | Multi-GPU, multi-node, dynamic orchestration |
| Framework integration | TensorFlow, PyTorch, ONNX, TensorRT, etc. | TensorRT-LLM, NIM, LLM-focused tools |
| When to use | General ML/AI serving | Scalable, multi-query LLM deployment |

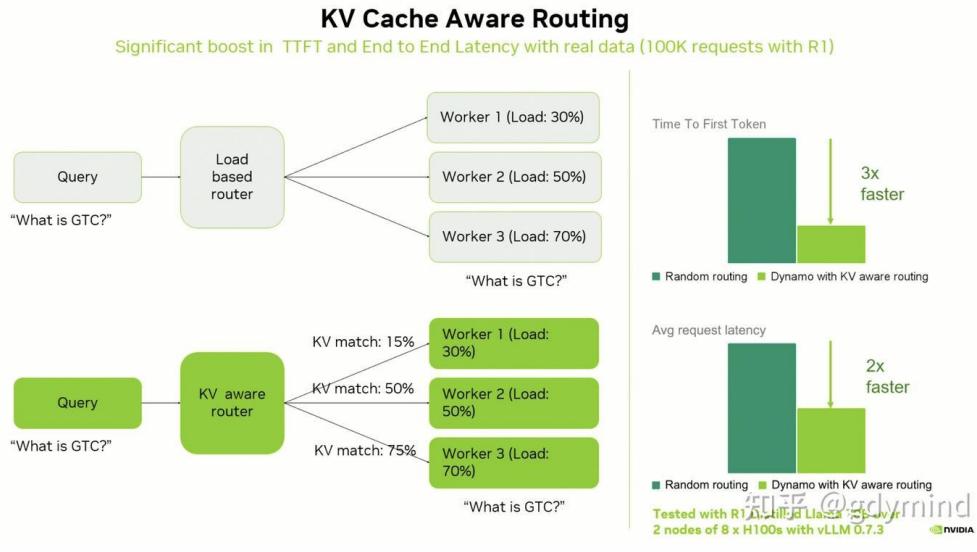
1. Dynamo Disaggregated Serving

Disaggregating these phases (prefill & decode) into specialized LLM engines allows for better hardware allocation, improved scalability, and overall enhanced performance.



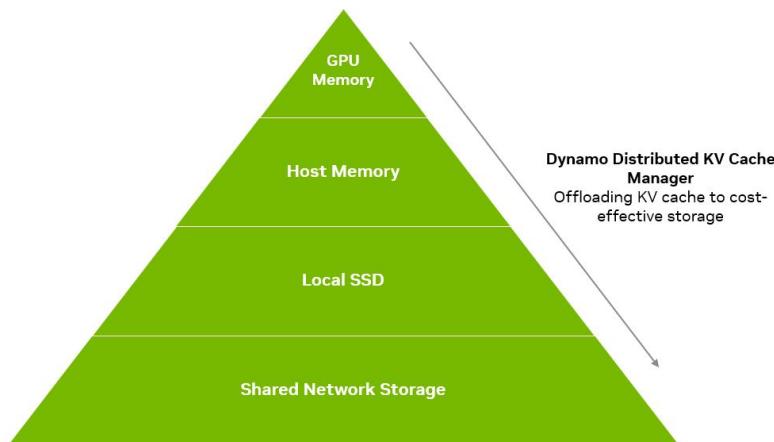
2. Dynamo Smart Router

Providing optimized inference for large language models by intelligently directing requests to workers with the most relevant cached data while simultaneously load balancing based on utilization metrics.



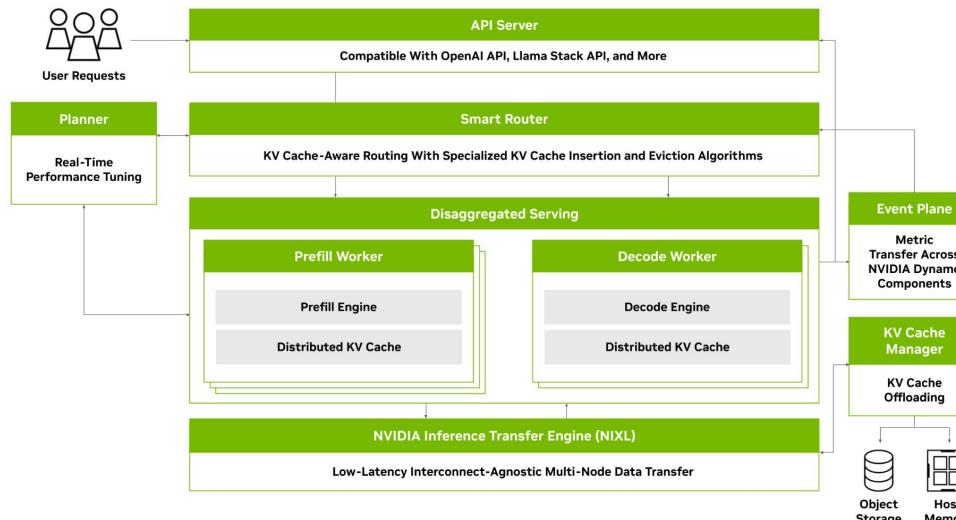
3. Dynamo Distributed KV Cache Manager

Enabling the offloading of older or less frequently accessed KV cache blocks to more cost-effective memory and storage solutions, such as CPU memory, local storage, or networked object or file storage.



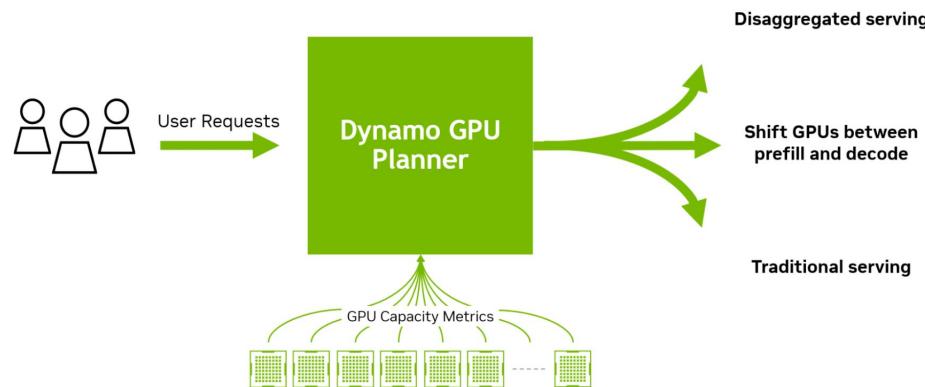
4. NVIDIA Inference Transfer Library (NIXL)

NIXL is designed to support inference frameworks by addressing their challenges while delivering high-bandwidth, low-latency point-to-point data transfers.



5. NVIDIA Dynamo Planner

Ensuring that GPU resources are allocated efficiently across prefill and decode, adapting to fluctuating workloads while maintaining peak system performance.



NVIDIA Dynamo's Performance

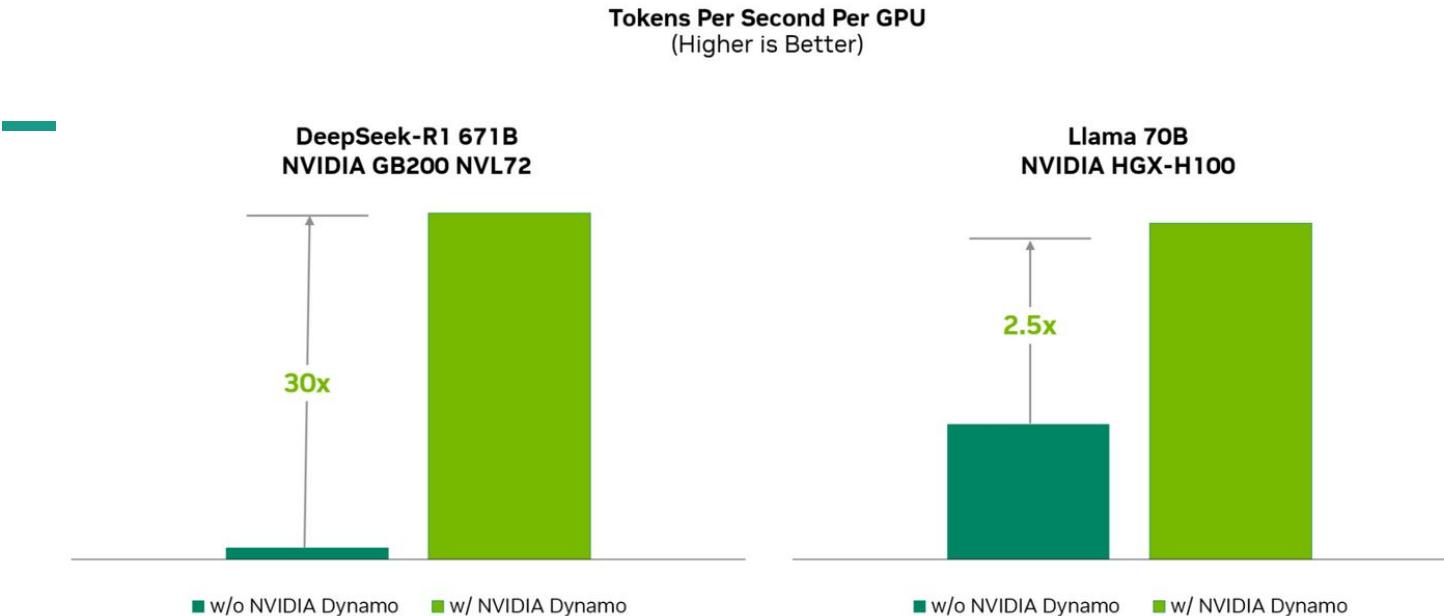
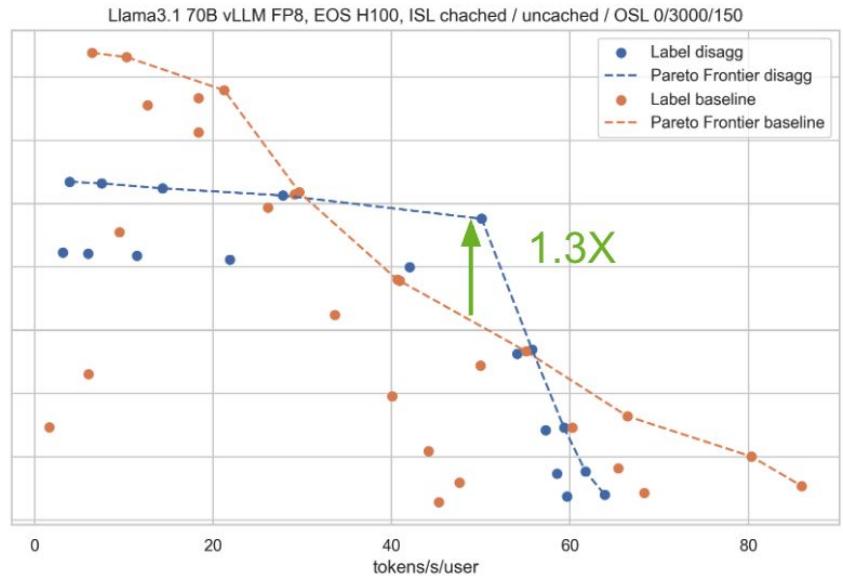


Figure 2. NVIDIA Dynamo delivers exceptional throughput performance when running DeepSeek-R1 671B model on NVIDIA GB200 NVL72 boosting performance by 30x. It more than doubles performance on the Llama 70B model running on NVIDIA Hopper GPUs

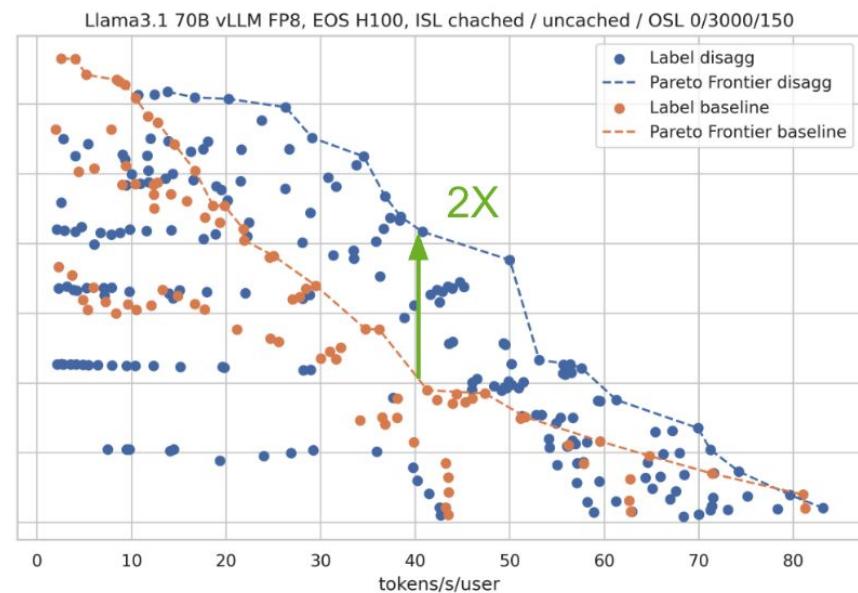
Left: TensorRT-LLM, FP4, ISL/OSL: 32K/8K. Without Dynamo: Inflight Batching, TEP16PP4DP4. With Dynamo: Disaggregated Serving, Context: EP4DP16, Generation: EP64DP3. Projected performance subject to change. Right: vLLM, FP8, ISL/OSL: 3K/50 . Without Dynamo: Inflight Batching, TP8DP2. With Dynamo: Disaggregated Serving, Context: TP2DP4, Generation: TP8.

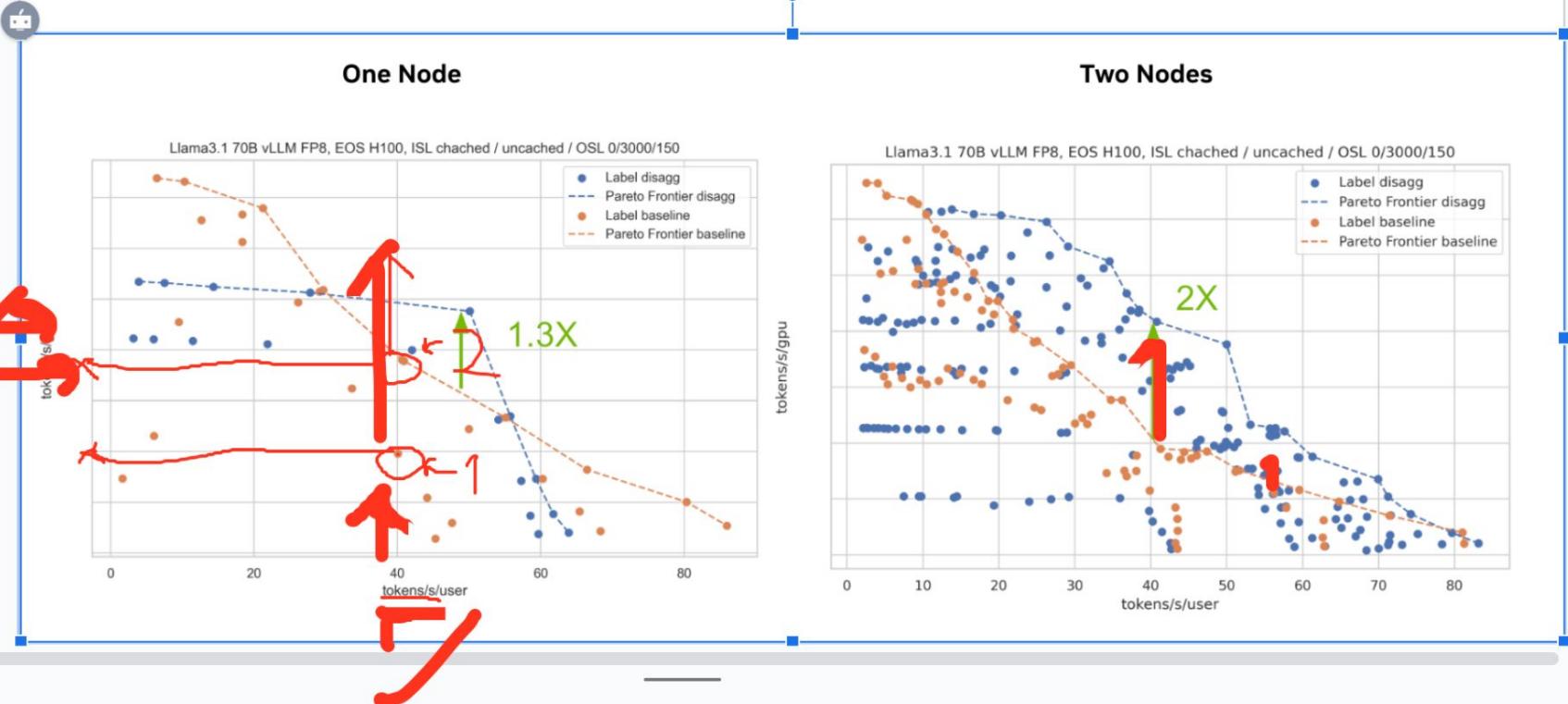
NVIDIA Dynamo's Performance (cont.)

One Node



Two Nodes





throughput ความเร็วในการ inference (ความเร็ว per sec) → วัดแต่ละ user ได้ความเร็วเท่าไหร่
throughput per GPU → วัดแต่ละ GPU ได้ความเร็วเท่าไหร่



NVIDIA Dynamo

Github: <https://github.com/ai-dynamo/dynamo>

Examples: <https://github.com/ai-dynamo/dynamo/blob/main/examples/llm/README.md>

3) Unsloth [fine-tune & inference]

<https://unsloth.ai/>

About Blog Contact Documentation

Easily finetune & train LLMs Get *faster* with unsloth

 Join our Discord  Start for free



Sign up to our newsletter
We'll share important news & more!

Email Submit

Unsloth: Less VRAM & Larger Batch Sizes

The unsloth library is a lightweight and high-performance fine-tuning framework for Large Language Models (LLMs). It's designed to make parameter-efficient fine-tuning (PEFT) – like LoRA (Low-Rank Adaptation) – fast, simple, and memory-efficient, especially on consumer hardware (like 8GB or 16GB VRAM GPUs).

It builds on top of Hugging Face Transformers + PEFT + BitsAndBytes.

It works by overwriting some parts of the modeling code with optimized operations. By manually deriving backpropagation steps and rewriting all Pytorch modules into Triton kernels.

Fine-tune Llama 3.3 with Unsloth

Dec 10, 2024 • By Daniel & Michael

Llama 3.3 (70B)

1xA100 80GB

13x
longer context

Llama 3.3 (70B)

1xA100 80GB

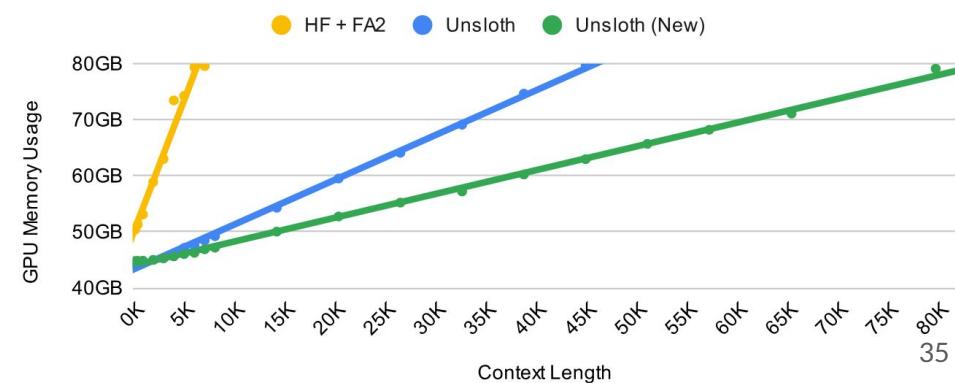
2x
faster

Llama 3.3 (70B)

1xA100 80GB

>75%
less VRAM

Llama 3.3 70B Long Context Memory Usage



Free on a single GPU, but not free for multiple GPUs!

Pricing

Free

Freeware of our standard version of unsloth

[Get started](#)

- ✓ Open-source
- ✓ Supports Mistral, Gemma
- ✓ Supports LLaMA 1, 2, 3
- ✓ Single GPU support
- ✓ Supports 4 bit, 16 bit LoRA

unsloth *Pro*

Unlock multi GPU support + 2.5x faster training + 20% less VRAM

[Contact us](#)

- ✓ 2.5x number of GPUs faster than FA2
- ✓ 20% less memory than OSS
- ✓ Multi GPU support
- ✓ Up to 8 GPUS support
- ✓ For any usecase

unsloth *Enterprise*

Unlock 30x faster training + multi-node support + 30% accuracy

[Contact us](#)

- ✓ 32x number of GPUs faster than FA2
- ✓ up to +30% accuracy
- ✓ 5x faster inference
- ✓ Supports full training
- ✓ All Pro plan features
- ✓ Multi-node support
- ✓ Customer support

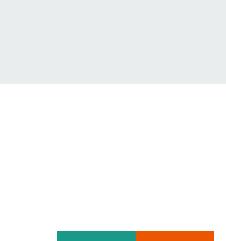


Unsloth supports (1) many models, (2) many PEFT (LoRA) techniques, and (3) many hardware



Unsloth.ai News

- 🎉 NEW! **EVERYTHING** is now supported including: full finetuning, pretraining, ALL models (Mixtral, MOE, Cohere, Mamba) and all training algorithms (KTO, DoRA) etc. MultiGPU support coming very soon.
- Supports 4bit and 16bit QLoRA / LoRA finetuning via [bitsandbytes](#).
- No change of hardware. Supports NVIDIA GPUs since 2018+. Minimum CUDA Capability 7.0 (V100, T4, Titan V, RTX 20, 30, 40x, A100, H100, L40 etc) [Check your GPU!](#) GTX 1070, 1080 works, but is slow.



Unsloth Notebook Examples



✨ Finetune for Free

Notebooks are beginner friendly. Read our [guide](#). Add your dataset, click "Run All", and export your finetuned model to GGUF, Ollama, vLLM or Hugging Face.

| Unsloth supports | Free Notebooks | Performance | Memory use |
|------------------------|--------------------------------|-------------|------------|
| GRPO (R1 reasoning) | Start for free | 2x faster | 80% less |
| Gemma 3 (4B) | Start for free | 1.6x faster | 60% less |
| Llama 3.2 (3B) | Start for free | 2x faster | 70% less |
| Phi-4 (14B) | Start for free | 2x faster | 70% less |
| Llama 3.2 Vision (11B) | Start for free | 2x faster | 50% less |
| Llama 3.1 (8B) | Start for free | 2x faster | 70% less |
| Qwen 2.5 (7B) | Start for free | 2x faster | 70% less |
| Mistral v0.3 (7B) | Start for free | 2.2x faster | 75% less |
| Ollama | Start for free | 1.9x faster | 60% less |
| DPO Zephyr | Start for free | 1.9x faster | 50% less |

- See [all our notebooks](#) and [all our models](#)
- Kaggle Notebooks for [Llama 3.2 Kaggle notebook](#), [Llama 3.1 \(8B\)](#), [Phi-4 \(14B\)](#), [Mistral \(7B\)](#)
- See detailed documentation for Unsloth [here](#).

[https://colab.research.google.com/github/unslotha/notebooks/blob/main/nb/Llama3.2_\(1B_and_3B\)-Conversational.ipynb](https://colab.research.google.com/github/unslotha/notebooks/blob/main/nb/Llama3.2_(1B_and_3B)-Conversational.ipynb)

We aim to fine-tune Llama3.2-3B in the conversational task (this code) on Colab!

Unsloth's performance on different tasks (accuracy)

| Model | Average | IFEval | BBH | MATH | MMLU |
|--|---------|---------|---------|---------|---------|
| unsloth/ phi-4 16-bit | 40.73 % | 68.82 % | 55.25 % | 50.00 % | 48.65 % |
| unsloth/ phi-4 -unsloth/bnb-4bit Dynamic 4-bit | 39.22 % | 67.94 % | 53.84 % | 45.62 % | 47.62 % |
| unsloth/ phi-4 -bnb-4bit BnB 4-bit | 39.06 % | 67.30 % | 53.54 % | 46.07 % | 47.29 % |
| microsoft/ phi-4 Microsoft 16-bit | 30.36 % | 5.85 % | 52.43 % | 31.65 % | 47.63 % |



Unsloth on DeepSeek 671B (around 1.2TB) with different MoE bits

| MoE Bits | Disk Size | Type | Quality | Link | Down_proj |
|----------|--------------|---------|---------|----------------------|--------------|
| 1.58-bit | 131GB | IQ1_S | Fair | Link | 2.06/1.56bit |
| 1.73-bit | 158GB | IQ1_M | Good | Link | 2.06bit |
| 2.22-bit | 183GB | IQ2_XXS | Better | Link | 2.5/2.06bit |
| 2.51-bit | 212GB | Q2_K_XL | Best | Link | 3.5/2.5bit |

Comparison

W/O (No) Unsloth

- Batch size = 1
- Num train steps = 60

W/ Unsloth

- Batch size = 2
- Num train steps = 60

209.5694 seconds used for training.
3.49 minutes used for training.
Peak reserved memory = 14.244 GB.
Peak reserved memory for training = 12.035 GB.
Peak reserved memory % of max memory = 96.628 %.
Peak reserved memory for training % of max memory = 81.643 %.



437.6992 seconds used for training.
7.29 minutes used for training.
Peak reserved memory = 4.131 GB.
Peak reserved memory for training = 0.69 GB.
Peak reserved memory % of max memory = 28.024 %.
Peak reserved memory for training % of max memory = 4.681 %.

4) DeepSpeed [fine-tune & inference]

The screenshot shows the DeepSpeed website homepage. At the top, there is a navigation bar with links to "Getting Started", "Blog", "Tutorials", "Documentation", "GitHub", and a search icon. On the left, there is a sidebar with categories: TRAINING, INFERENCE, COMPRESSION, SCIENCE, GETTING STARTED, DS_CONFIG, and TUTORIALS. The DS_CONFIG category is expanded, listing Autotuning, Batch size, Optimizer, FP16, BFLOAT16, ZeRO optimizations, Logging, Flops Profiler, Monitoring, Communication Logging, Model Compression, and Data Efficiency. The main content area features a heading "Latest News" and a summary of a recent blog post: "DeepSpeed empowers ChatGPT-like model training with a single click, offering 15x speedup over SOTA RLHF systems with unprecedented cost reduction at all scales; [learn how.](#)". Below this, there is a list of six blog posts:

- [2025/03] [DeepSpeed AutoTP: Automatic Tensor Parallel Training of Hugging Face models](#)
- [2024/12] [DeepSpeed Domino: Communication-Free LLM Training Engine](#)
- [2024/08] [DeepSpeed on Windows](#) [日本語] [中文]
- [2024/08] [DeepNVMe: Improving DL Applications through I/O Optimizations](#) [日本語] [中文]
- [2024/07] [DeepSpeed Universal Checkpointing: Efficient and Flexible Checkpointing for Large Scale Distributed Training](#) [日本語]

<https://www.deepspeed.ai/>

DeepSpeed: Extreme Speed and Scale for DL Training and Inference

DeepSpeed is an open-source deep learning optimization library developed by [Microsoft](#), designed to train and serve large-scale models efficiently – especially large language models (LLMs) like GPT, BERT, and others. It makes [distributed training \(multiple GPUs\)](#) and inference easy, efficient, and effective.

DeepSpeed has four innovation pillars:

Training

- Speed Scale Cost
- Democratization
- MoE models
- Long sequence
- RLHF

Inference

- Large models
- Latency
- Serving cost
- Agility

Compression

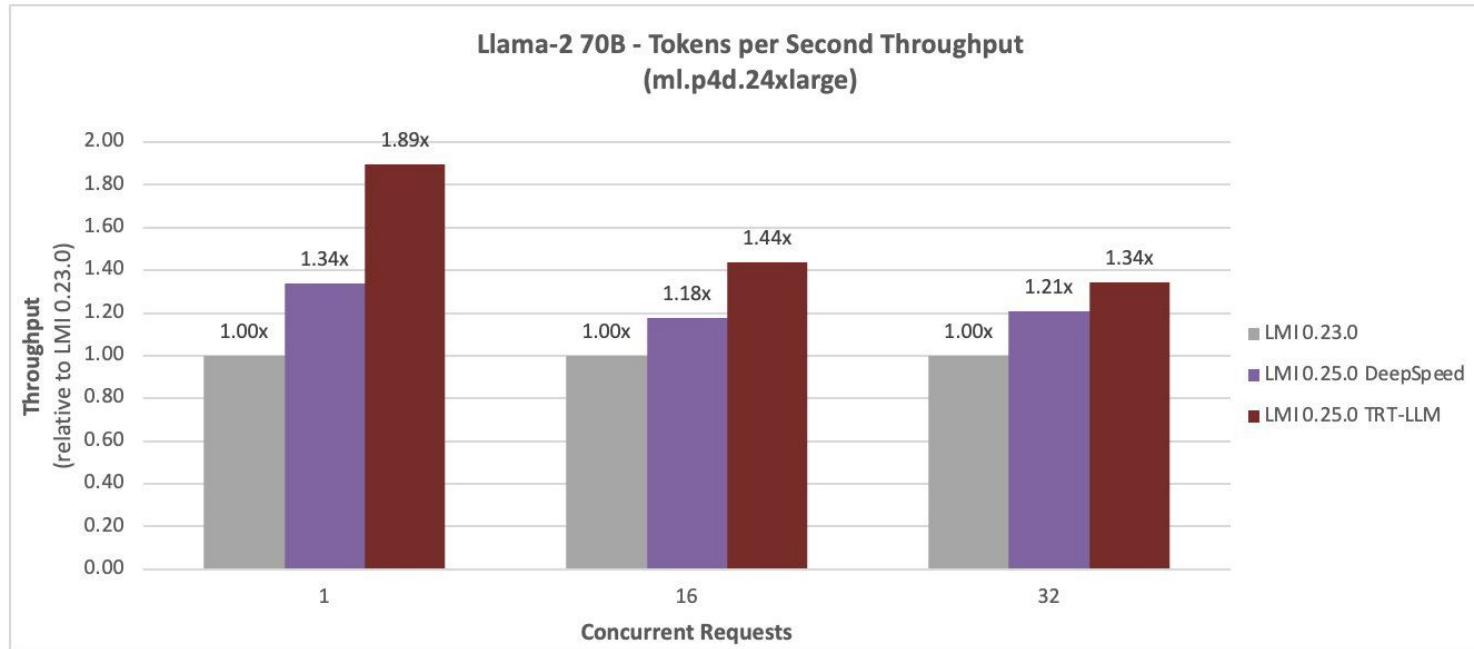
- Model size
- Latency
- Composability
- Runnable on client devices

Science

- Speed
- Scale
- Capability
- Diversity
- Discovery

DeepSpeed also supports inference, but TRT-LLM is better!

Amazon SageMaker launches a new version (0.25.0) of **Large Model Inference (LMI)** Deep Learning Containers that works with many frameworks like DeepSpeed, FastTransformer, TensorRT-LLM, vLLM.

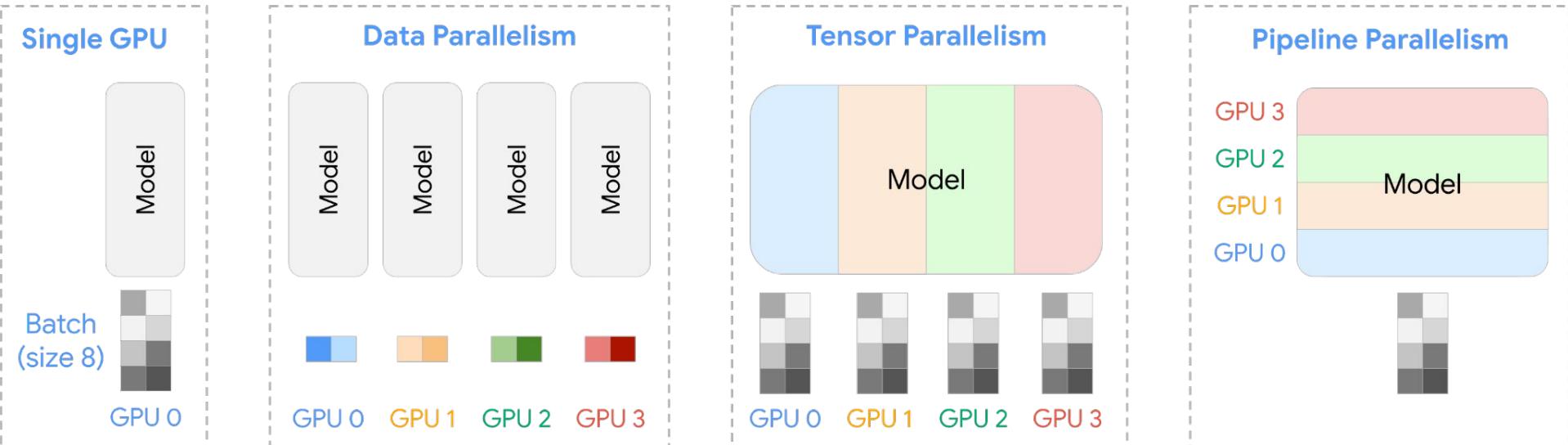


DeepSpeed Main Features

1. Data, Pipeline, and Model (with Megatron) parallelism
2. The Zero Redundancy Optimizer (ZeRO) [1]

| Feature | Description |
|----------------------------------|---|
| Zero Redundancy Optimizer (ZeRO) | Reduces memory usage to enable training huge models across GPUs |
| DeepSpeed-Inference | High-performance, low-latency inference engine (serving large models) |
| Model Parallelism | Splits one model across multiple GPUs or machines |
| Optimizer Offloading | Moves optimizer states to CPU or NVMe to save GPU memory |
| Mixed Precision (FP16/BF16) | Enables faster training with lower precision |
| LoRA + Quantization support | Fine-tune large models with reduced memory footprint |
| Megatron + DeepSpeed | Joint framework for extreme-scale training |

DeepSpeed - (1) Model Parallelism - various modes



A single layer's computation is split across multiple GPUs.

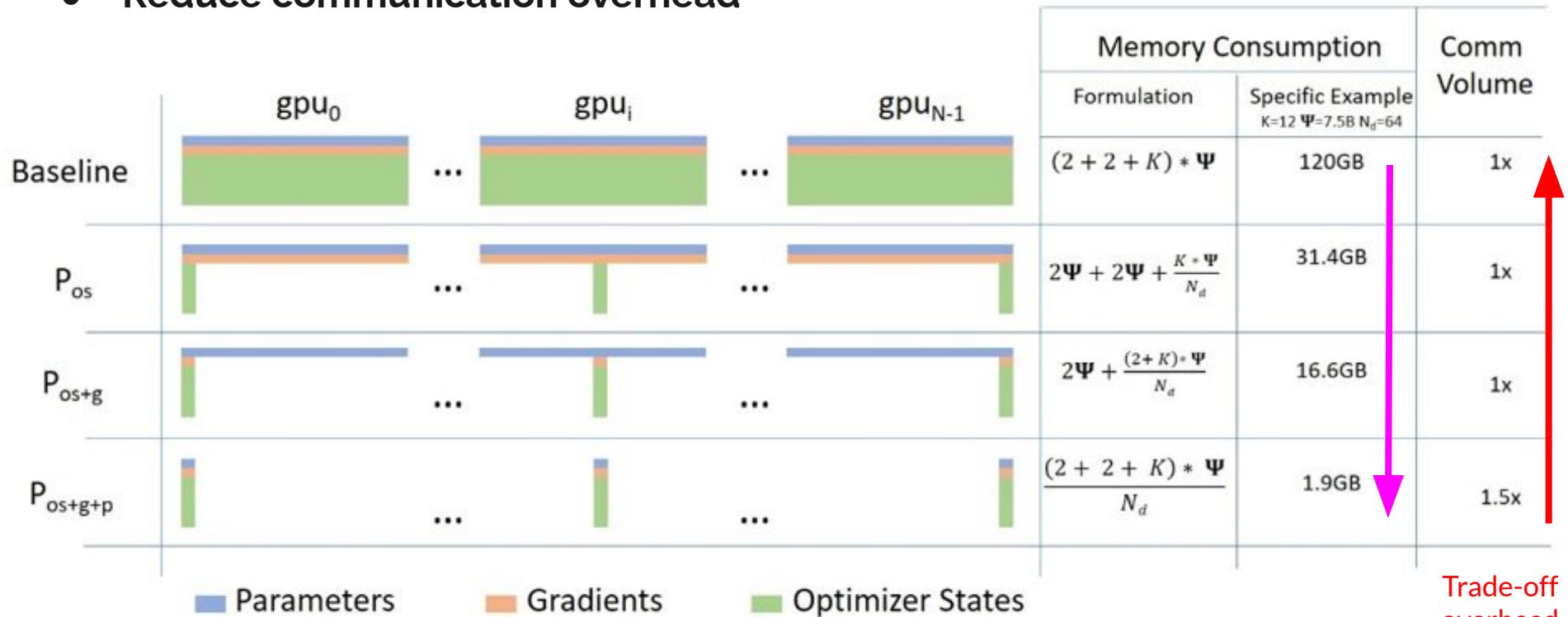
Splits the model across GPUs by layers. Each GPU processes a different stage of the model.

DeepSpeed - (2) Zero Redundant Optimizer (ZeRO)

| Feature | ZeRO-1 | ZeRO-2 | ZeRO-3 |
|--------------------------|--------|----------|--------|
| Optimizer State Sharding | ✓ Yes | ✓ Yes | ✓ Yes |
| Gradient Sharding | ✗ No | ✓ Yes | ✓ Yes |
| Parameter Sharding | ✗ No | ✗ No | ✓ Yes |
| Memory Savings | ~4x | ~8x | ~16x+ |
| Communication Overhead | Low | Moderate | High |

DeepSpeed - (2) Zero Redundant Optimizer (ZeRO)

- Sharding: Parameters, Gradients, States
- Save memory consumption
- Reduce communication overhead



Trade-off overhead



DeepSpeed Resources

Github: <https://github.com/deepspeedai/DeepSpeed>

Transformers with DeepSpeed: <https://huggingface.co/docs/transformers/deepspeed>

Tutorial: <https://www.deepspeed.ai/tutorials/>

Conclusion

- To optimize the inference model & self-use: TensorRT-LLM
- To serve the application with a single GPU: NVIDIA Triton Server
- To serve the application with multiple GPUs: NVIDIA Dynamo
- To fine-tune the model on a single GPU: Unsloth
- To fine-tune a huge model that needs to be distributed into many GPUs: DeepSpeed