Tutorial on GPU Optimization

Ziv Ilan- Solution Architect, NVIDIA

Sergio Perez - Solution Architect, NVIDIA

Harshita Seth - Solution Architect, NVIDIA

Agenda of the tutorial

- Demo of TensorRT + Triton
- Build a TensorRT-LLM engine of Gemma 2B
- Evaluate the engine on MMLU
- Launch the Triton inference server
- Measure the throughput of Triton inference server
- Optional - Compare to quantized versions of Gemma 2B

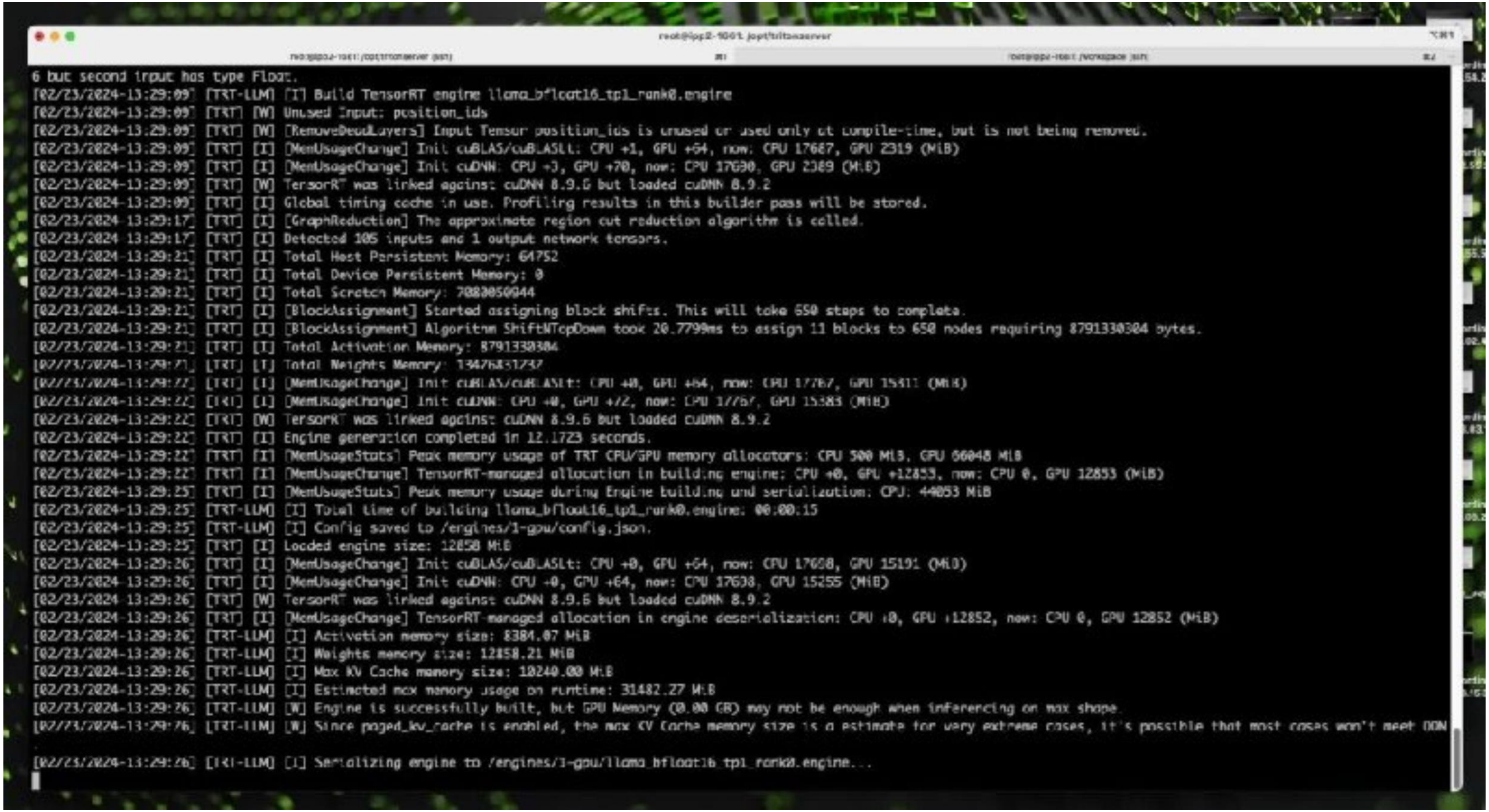
How to connect to your tutorial instance?

- Create your NVIDIA account **https://learn.nvidia.com/join**
- Navigate to **https://learn.nvidia.com/dli-event**
- Enter the event code: **CERN_XLAB_SE24**
- Click on Start – this will spin up an Nvidia A10 32GB cloud instance
- It takes 10-15 minutes for the environment and the model artifacts to load

Demo: LLaMA 7B with TensorRT-LLM + Triton

Source code available in our Github repo

Demo Video



6 but second input has type Float.
[02/23/2024-13:29:09] [TRT-LLM] [I] Build TensorRT engine llama_bfloat16_tp1_rank0.engine
[02/23/2024-13:29:09] [TRT] [W] Unused Input: position_ids
[02/23/2024-13:29:09] [TRT] [W] [RemoveDeadLayers] Input Tensor position_ids is unused or used only at compile-time, but is not being removed.
[02/23/2024-13:29:09] [TRT] [I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +1, GPU +64, now: CPU 17687, GPU 2319 (MiB)
[02/23/2024-13:29:09] [TRT] [I] [MemUsageChange] Init cuDNN: CPU +3, GPU +70, now: CPU 17690, GPU 2389 (MiB)
[02/23/2024-13:29:09] [TRT] [W] TensorRT was linked against cuDNN 8.9.6 but loaded cuDNN 8.9.2
[02/23/2024-13:29:09] [TRT] [I] Global timing cache in use. Profiling results in this builder pass will be stored.
[02/23/2024-13:29:17] [TRT] [I] [GraphReduction] The approximate region cut reduction algorithm is called.
[02/23/2024-13:29:17] [TRT] [I] Detected 105 inputs and 1 output network tensors.
[02/23/2024-13:29:21] [TRT] [I] Total Host Persistent Memory: 64752
[02/23/2024-13:29:21] [TRT] [I] Total Device Persistent Memory: 0
[02/23/2024-13:29:21] [TRT] [I] Total Scratch Memory: 7080050944
[02/23/2024-13:29:21] [TRT] [I] [BlockAssignment] Started assigning block shifts. This will take 650 steps to complete.
[02/23/2024-13:29:21] [TRT] [I] [BlockAssignment] Algorithm ShiftNTopDown took 20.7799ms to assign 11 blocks to 650 nodes requiring 8791330304 bytes.
[02/23/2024-13:29:21] [TRT] [I] Total Activation Memory: 8791330304
[02/23/2024-13:29:21] [TRT] [I] Total Weights Memory: 13476831232
[02/23/2024-13:29:22] [TRT] [I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +64, now: CPU 17767, GPU 15411 (MiB)
[02/23/2024-13:29:22] [TRT] [I] [MemUsageChange] Init cuDNN: CPU +0, GPU +72, now: CPU 17767, GPU 15383 (MiB)
[02/23/2024-13:29:22] [TRT] [W] TensorRT was linked against cuDNN 8.9.6 but loaded cuDNN 8.9.2
[02/23/2024-13:29:22] [TRT] [I] Engine generation completed in 12.1723 seconds.
[02/23/2024-13:29:22] [TRT] [I] [MemUsageStats] Peak memory usage of TRT CPU/GPU memory allocators: CPU 500 MiB, GPU 66048 MiB
[02/23/2024-13:29:22] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in building engine: CPU +0, GPU +12853, now: CPU 0, GPU 12853 (MiB)
[02/23/2024-13:29:25] [TRT] [I] [MemUsageStats] Peak memory usage during Engine building and serialization: CPU: 44053 MiB
[02/23/2024-13:29:25] [TRT-LLM] [I] Total time of building llama_bfloat16_tp1_rank0.engine: 00:00:15
[02/23/2024-13:29:25] [TRT-LLM] [I] Config saved to /engines/1-gpu/config.json.
[02/23/2024-13:29:25] [TRT] [I] Loaded engine size: 12858 MiB
[02/23/2024-13:29:26] [TRT] [I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +64, now: CPU 17608, GPU 15191 (MiB)
[02/23/2024-13:29:26] [TRT] [I] [MemUsageChange] Init cuDNN: CPU +0, GPU +64, now: CPU 17608, GPU 15255 (MiB)
[02/23/2024-13:29:26] [TRT] [W] TensorRT was linked against cuDNN 8.9.6 but loaded cuDNN 8.9.2
[02/23/2024-13:29:26] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in engine deserialization: CPU +0, GPU +12852, now: CPU 0, GPU 12852 (MiB)
[02/23/2024-13:29:26] [TRT-LLM] [I] Activation memory size: 8384.07 MiB
[02/23/2024-13:29:26] [TRT-LLM] [I] Weights memory size: 12858.21 MiB
[02/23/2024-13:29:26] [TRT-LLM] [I] Max KV Cache memory size: 10240.00 MiB
[02/23/2024-13:29:26] [TRT-LLM] [I] Estimated max memory usage on runtime: 31482.27 MiB
[02/23/2024-13:29:26] [TRT-LLM] [W] Engine is successfully built, but GPU Memory (0.00 GB) may not be enough when inferencing on max shape.
[02/23/2024-13:29:26] [TRT-LLM] [W] Since paged_kv_cache is enabled, the max KV Cache memory size is a estimate for very extreme cases, it's possible that most cases won't meet OOM
[02/23/2024-13:29:26] [TRT-LLM] [I] Serializing engine to /engines/1-gpu/llama_bfloat16_tp1_rank0.engine...

MMLU Overview

Academic benchmarks to evaluate LLMs

The MMLU (Measuring Massive Multitask Language Understanding) metric is a benchmark designed to evaluate the performance of large language models across a wide range of tasks and domains, providing a comprehensive assessment of a model's general knowledge, reasoning, and language understanding abilities.

Quantization

How to Choose a Precision

- Best precision varies by application
  - FP8 activations generally provides best performacne
- Weight quantization reduces memory footprint & traffic
  - Reduces latency
  - Can fit larger models
  - Costs compute time to unpack the weights
- Activation quantization saves on compute
  - Improves throughput
  - Can run larger batch sizes
- *WXAY* = weights quantized to *X* bits, and activations to *Y*
- [Quantization Guide]

Method	Performance Improvement small batch BS <=4	 large batch BS>=16	Accuracy impact	Calibration time
FP8 (W8A8)	Medium	Medium	**Very low / None**	**O(1min)**
INT8 SQ (W8A8)	Medium	Medium	Medium	O(1min)
INT8 WO (W8A16)	Medium	***None***	**Low**	***None***
INT4 WO (W4A16)	**High**	***None***	**High**	***None***
INT4 AWQ (W4A16)	**High**	***None***	**Low**	O(10min)
INT4 GPTQ (W4A16)	**High**	***None***	**Low**	O(10min)
INT4-FP8 AWQ (W4A8)	**High**	Medium	**Low**	O(10min)

SQ = Smooth Quant
WO = Weight Only
AWQ = Activation Aware Quantization

Wrapping up: Trends in model compression

Distilling the Knowledge of LLMs into SLMs

Train only the largest LLM and get smaller models with similar quality

How to Prune and Distill Llama-3.1 8B to an NVIDIA Llama-3.1-Minitron 4B Model

Aug 14, 2024

+32 Like Discuss (5)

By Sharath Sreenivas, Vinh Nguyen, Saurav Muralidharan, Marcin Chochowski and Raviraj Joshi

See blog https://developer.nvidia.com/blog/how-to-prune-and-distill-llama-3-1-8b-to-an-nvidia-llama-3-1-minitron-4b-model/
See paper https://arxiv.org/pdf/2407.14679

NVIDIA

FP4 Format Supported in Blackwell Platform

New FP4 format for inference

Data Center / Cloud

English

NVIDIA Blackwell Platform Sets New LLM Inference Records in MLPerf Inference v4.1

Aug 28, 2024

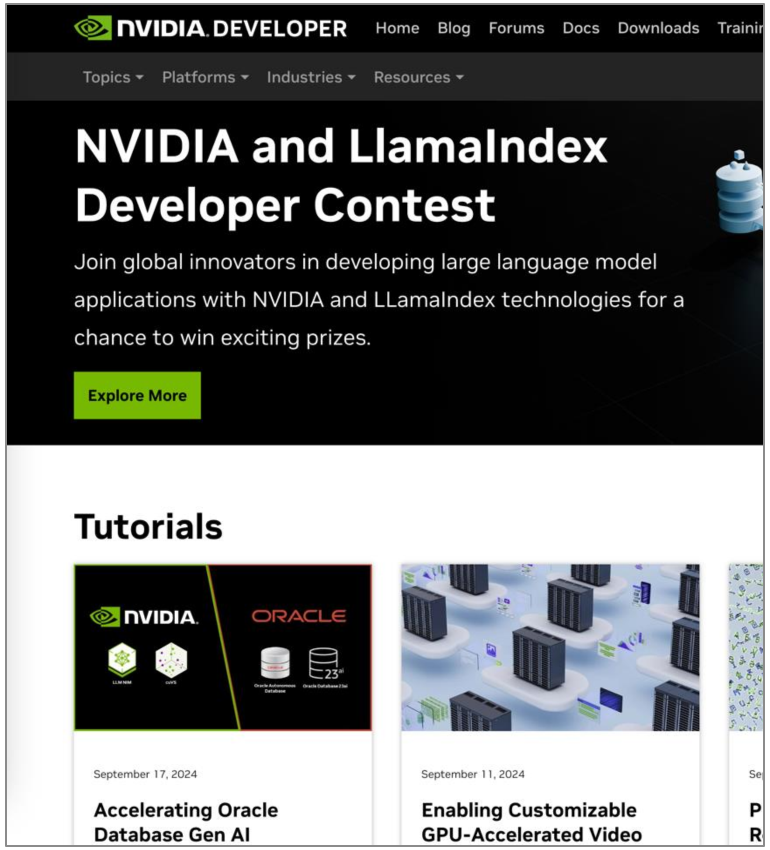
+19 Like Discuss (1)

By Ashraf Eassa, Ashwin Nanjappa, Zhihan Jiang, Yiheng Zhang, Jun Yang, Zihao Kong and Shengliang Xu

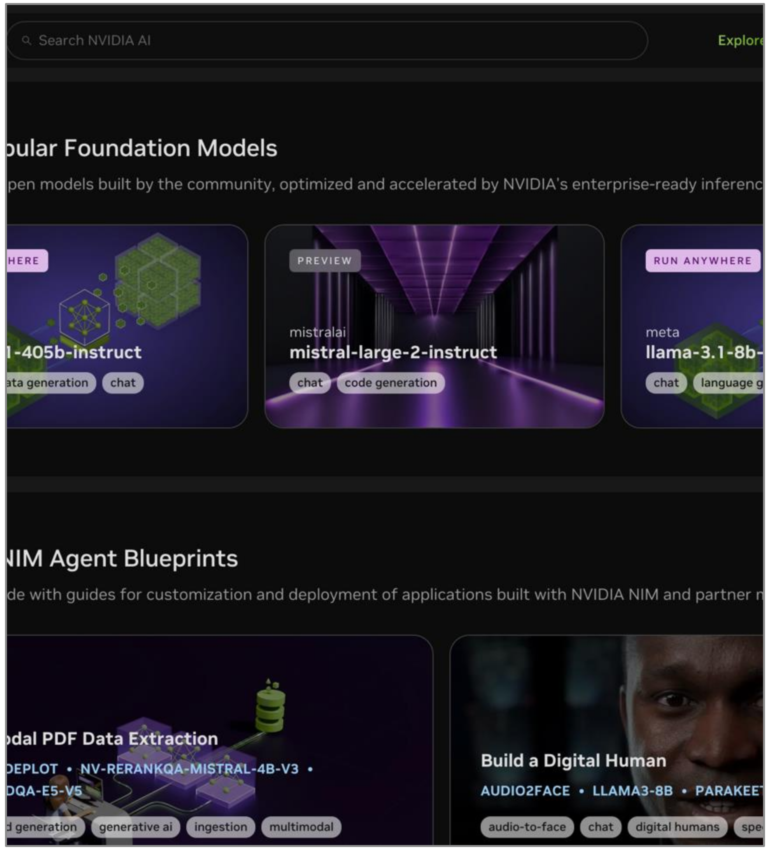
See blog https://developer.nvidia.com/blog/nvidia-blackwell-platform-sets-new-llm-inference-records-in-mlperf-inference-v4-1/

The GPU Journey Continues: Stay Ahead of the Curve and Keep Innovating

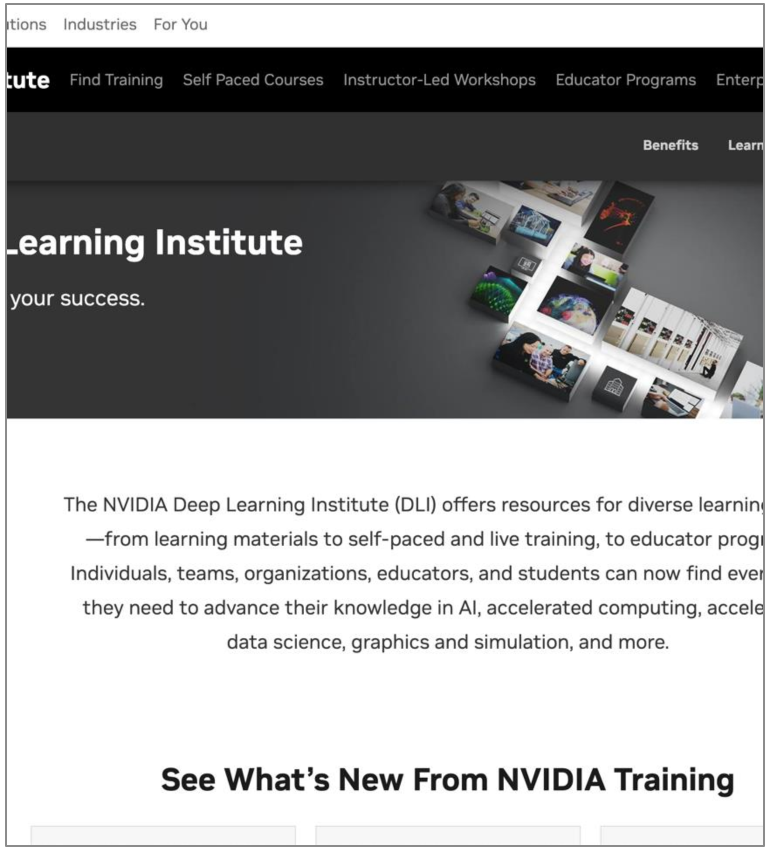
Take your next steps in one of the following platforms



https://developer.nvidia.com/



https://build.nvidia.com/



https://learn.nvidia.com/

Thank you!

Ziv Ilan - Solution Architect, NVIDIA

Sergio Perez - Solution Architect, NVIDIA

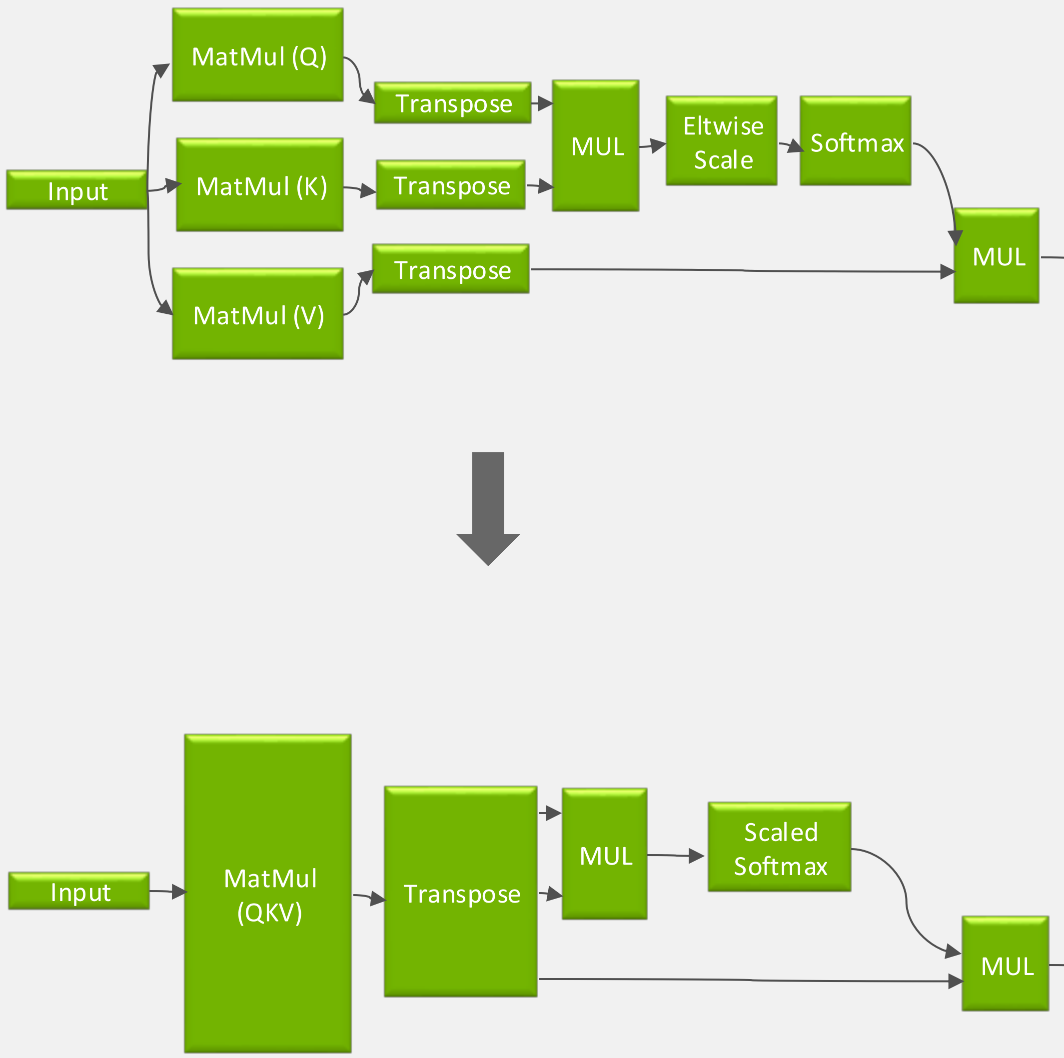
Harshita Seth - Solution Architect, NVIDIA

Extra slides about TensorRT features

LAYER & TENSOR FUSION

Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel

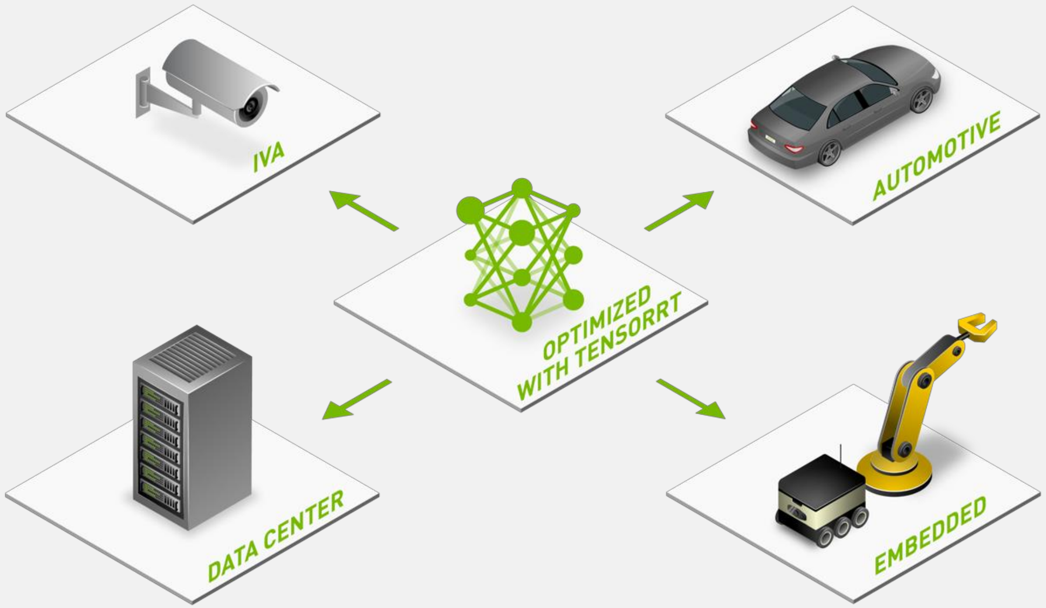
- Combines successive nodes into a single node, making single kernel execution
- Significantly reduces number of layers to compute, resulting in faster performance
- Eliminates unnecessary memory traffic by removing concat/slice layers
- See the supported fusion list



KERNEL AUTO-TUNING

Selects best data layers and algorithms based on the target GPU platform

- Hundreds of specialized kernels optimized for every GPU Platform
- TensorRT optimizer uses runtime profile to select the best performance kernels
- Ensures best performance for specific deployment platform and specific neural network

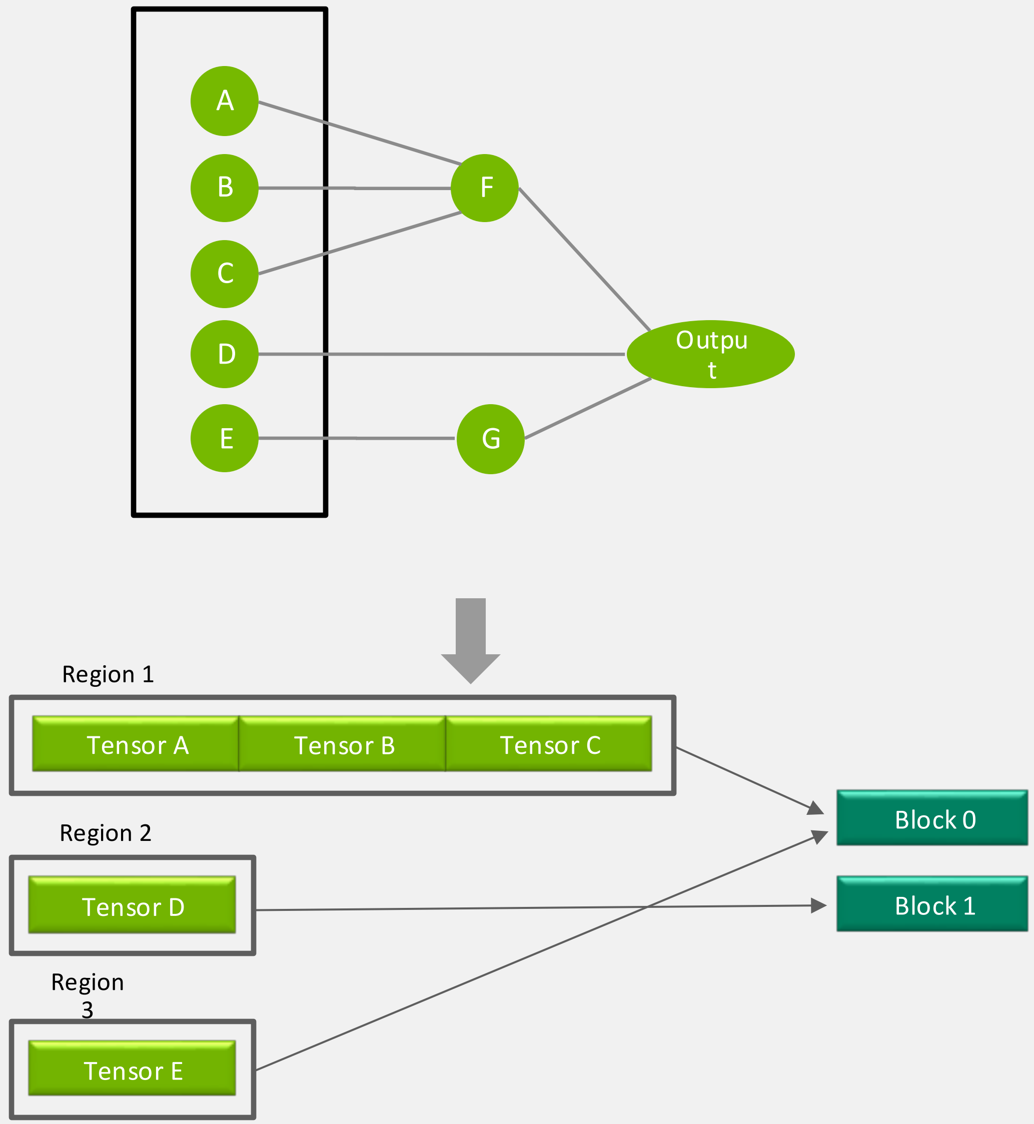


DYNAMIC TENSOR MEMORY

Minimizes memory footprint and reuses memory for tensors efficiently

- Reduces memory footprint and improves memory re-use
- Graph optimizer combines tensors into regions
- Region lifetime is a section of network execution time
- Memory Optimizer assigns regions to blocks; regions assigned to a block have disjoint lifetimes
- Just like register allocation

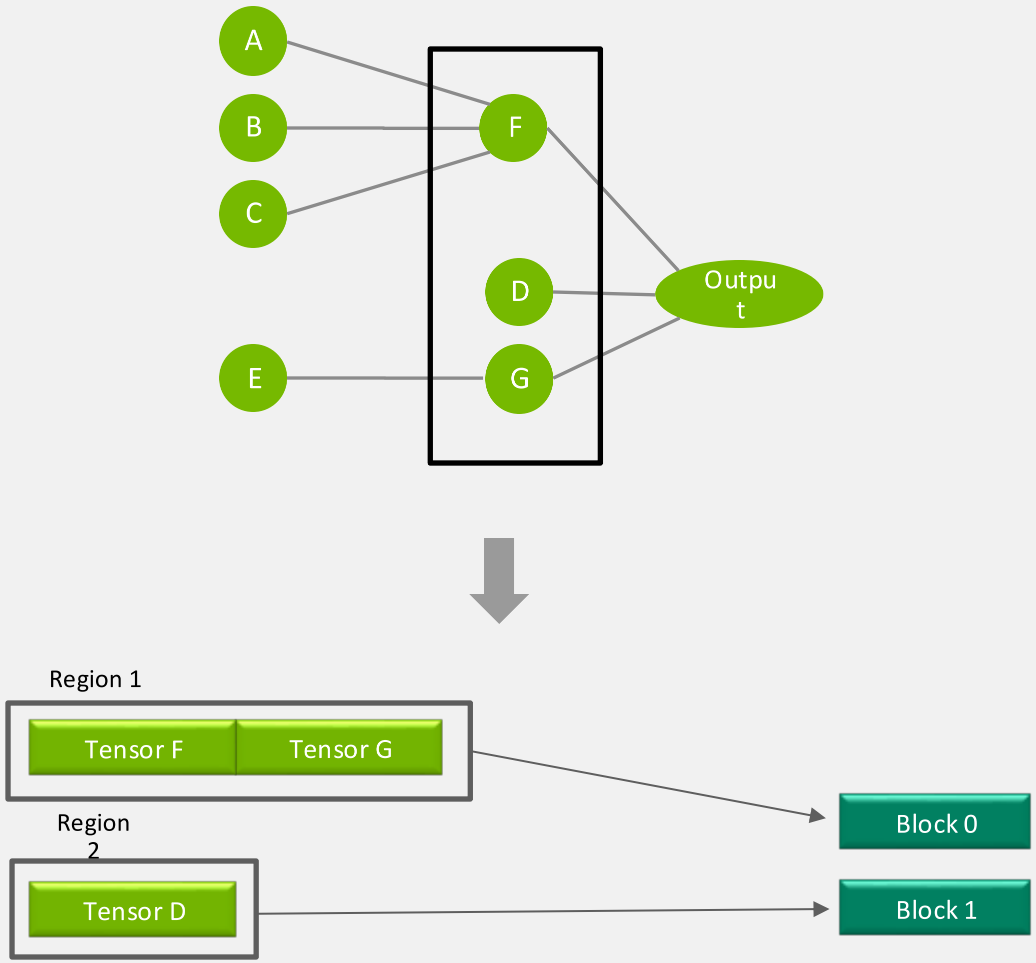
developer.nvidia.com/tensorrt



DYNAMIC TENSOR MEMORY

Minimizes memory footprint and reuses memory for tensors efficiently

- Reduces memory footprint and improves memory re-use
- Graph optimizer combines tensors into regions
- Region lifetime is a section of network execution time
- Memory Optimizer assigns regions to blocks; regions assigned to a block have disjoint lifetimes
- Just like register allocation

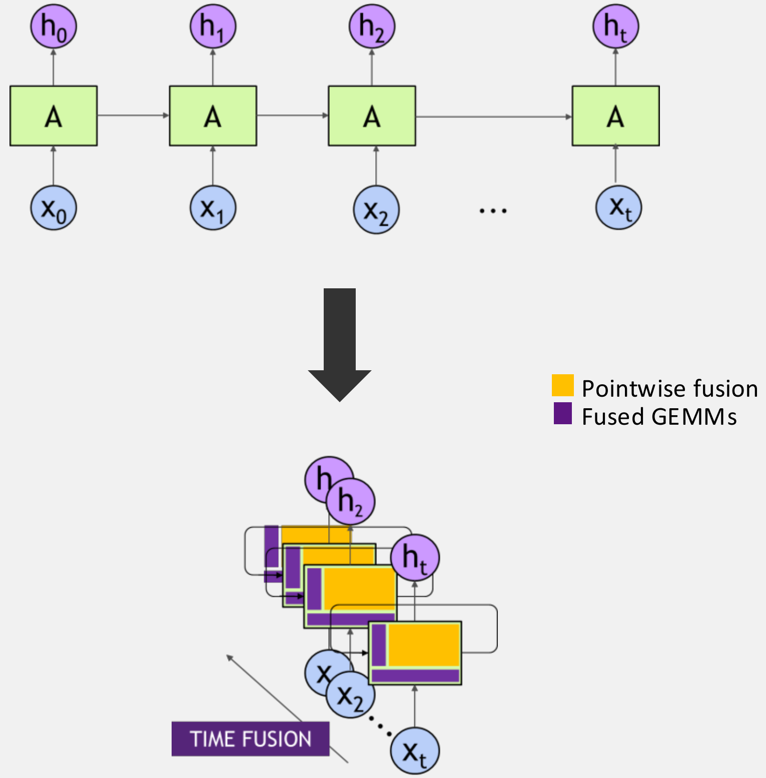


developer.nvidia.com/tensorrt

TIME FUSION

Optimizes recurrent neural networks over time steps with dynamically generated kernels

- Recurrent Neural Network Optimizations
- Deploy highly optimized ASR and TTS
- Compiler fuses pointwise ops, fuses GEMMs and compute efficiently across time steps



developer.nvidia.com/tensorrt

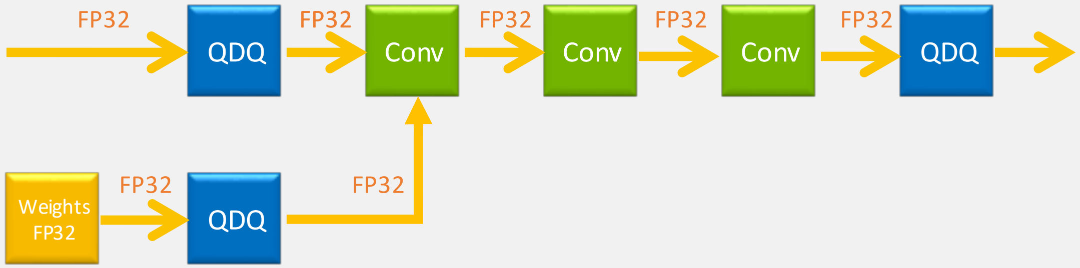
QUANTIZATION AWARE TRAINING

Improved accuracy for INT8 inference

- Better accuracy compared to Post Training Quantization (PTQ)
- Quantize state of the art models with minimal loss of accuracy
- TensorRT optimizes the Q/DQ graph for inference without compromising performance
- Quantization Toolkit available for PyTorch and TensorFlow in OSS supporting QAT, PTQ and export to ONNX

developer.nvidia.com/tensorrt

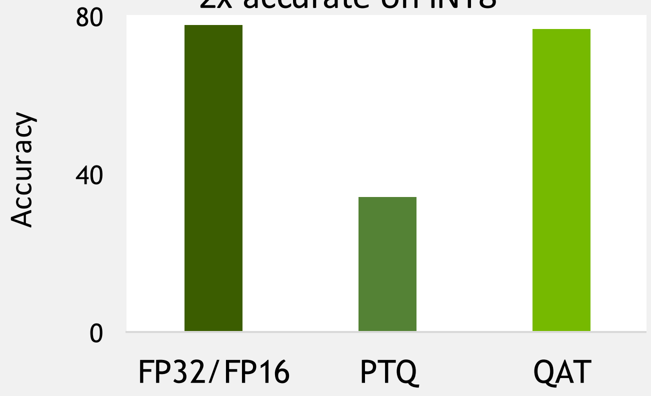
Training with Q/DQ



PyTorch GitHub

TensorFlow GitHub

2x accurate on INT8

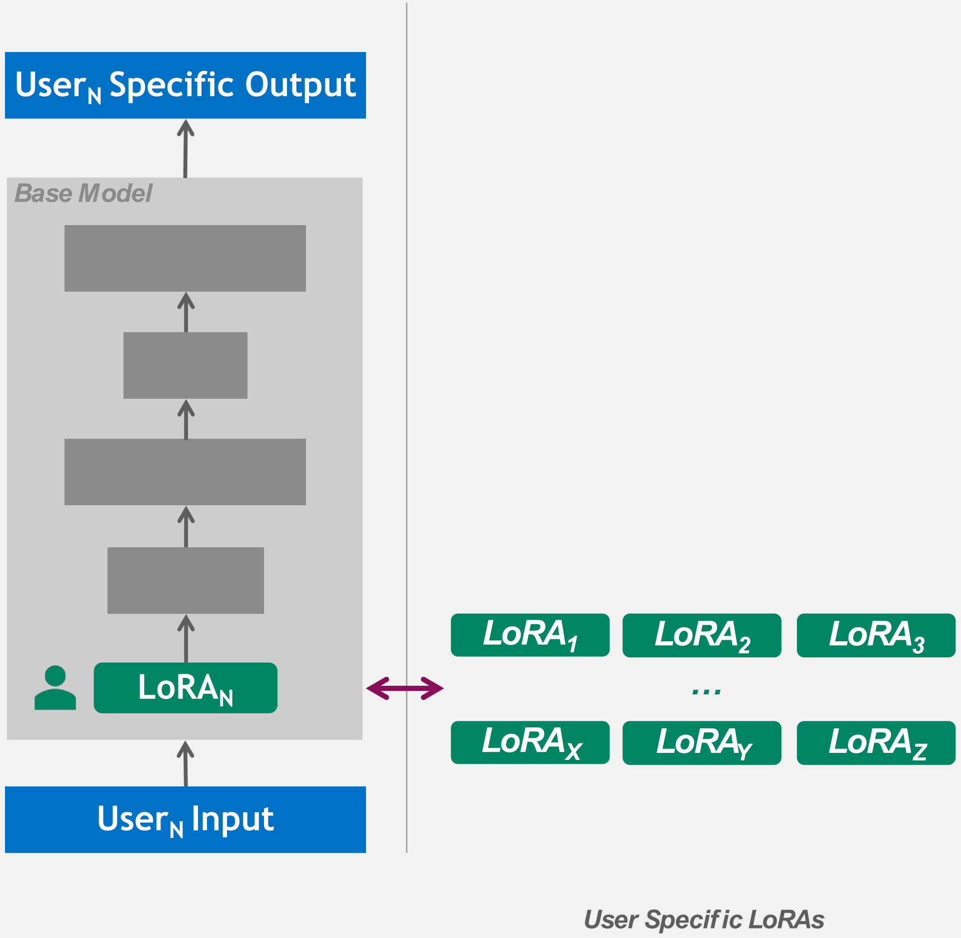


Network: EfficientNet-B0, Framework: PyTorch

LoRA & Customization

Efficiently Supporting Customer User Experience

- LoRA & Prompt tuned models are support in TRT-LLM
- Support mulitple customers with a single model
- Dynamically swap LoRA's at runtime
- SLoRA / LoRAx caching adapters on device
- Base model can be quantized for memory savings
    - QLoRA in progress

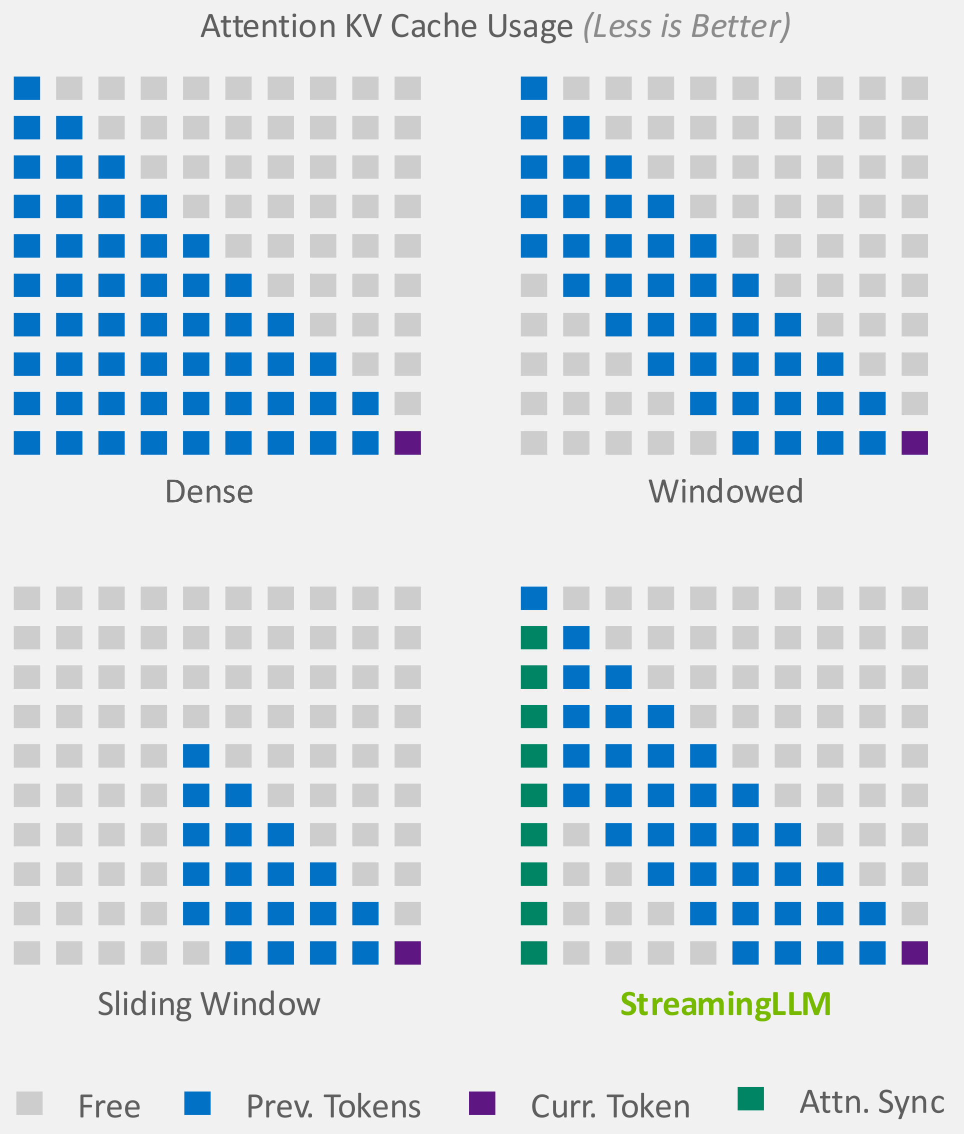


Dynamically Swap LoRAs based on User

KV Cache & Attention Techniques

(Sliding) Window Attention, & Streaming LLM

- Allow for longer (sometimes unlimited) sequence length
  - Reduces KV Cache Memory usage
  - Avoids OOM Errors
- (Sliding) Windowed Attention evict tokens based on arrival
  - Significantly reduces memory usage
  - Can negatively impact accuracy or require recomputing KV
- Streaming-LLM allows for unlimited sequence length
  - Does not evict Attention Sinks (important elements)
  - KV Cache stays constant size
  - Does not require recompute & does not impact accuracy
  - Particulary beneficial for multi-turn (ie. chat) usecases

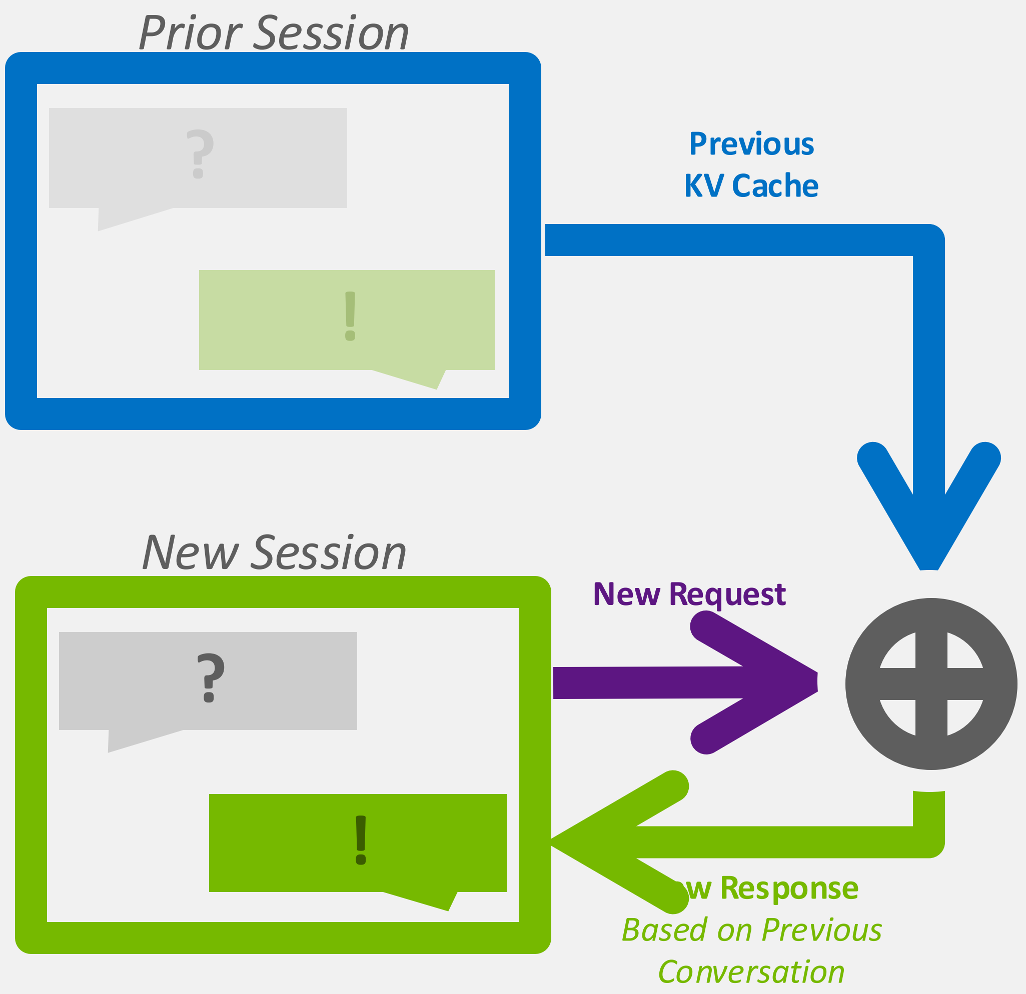


KV Cache Reusage

System Prompt Caching & Block reusage

Allows for interactive/ turn based systems & System Prompts

- Load prior KV cachce blocks to avoid recomupation
  - Saves significant compute
  - Reduces Start-up time
- Block resuage allows for turn-based (chat) applications
  - Allows for additional options for intelligently reusing blocks
- System prompts allows for a preset KV cache for the LLM
  - E.g. to give rules, personality, or prior knowledge

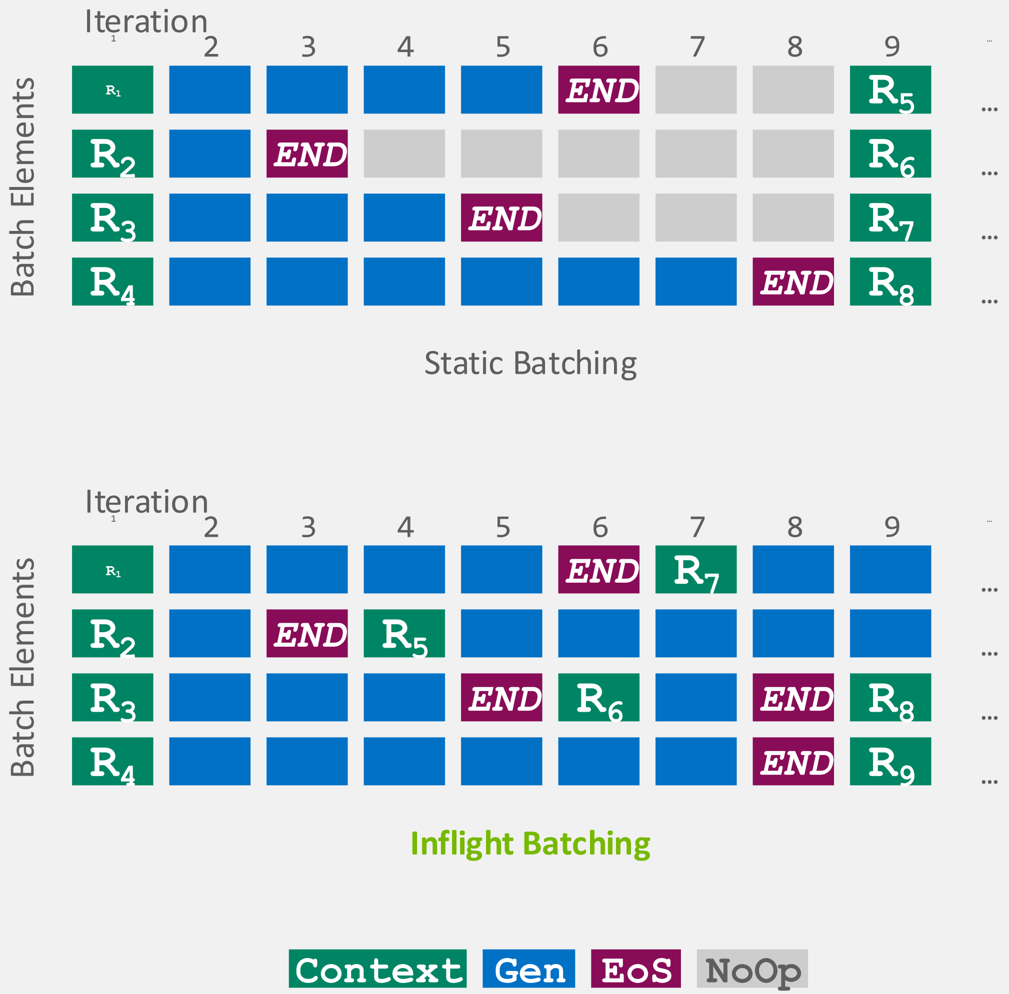


Inflight Batching

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



KV Cache Optimizations

Paged & Quantized KV Cache

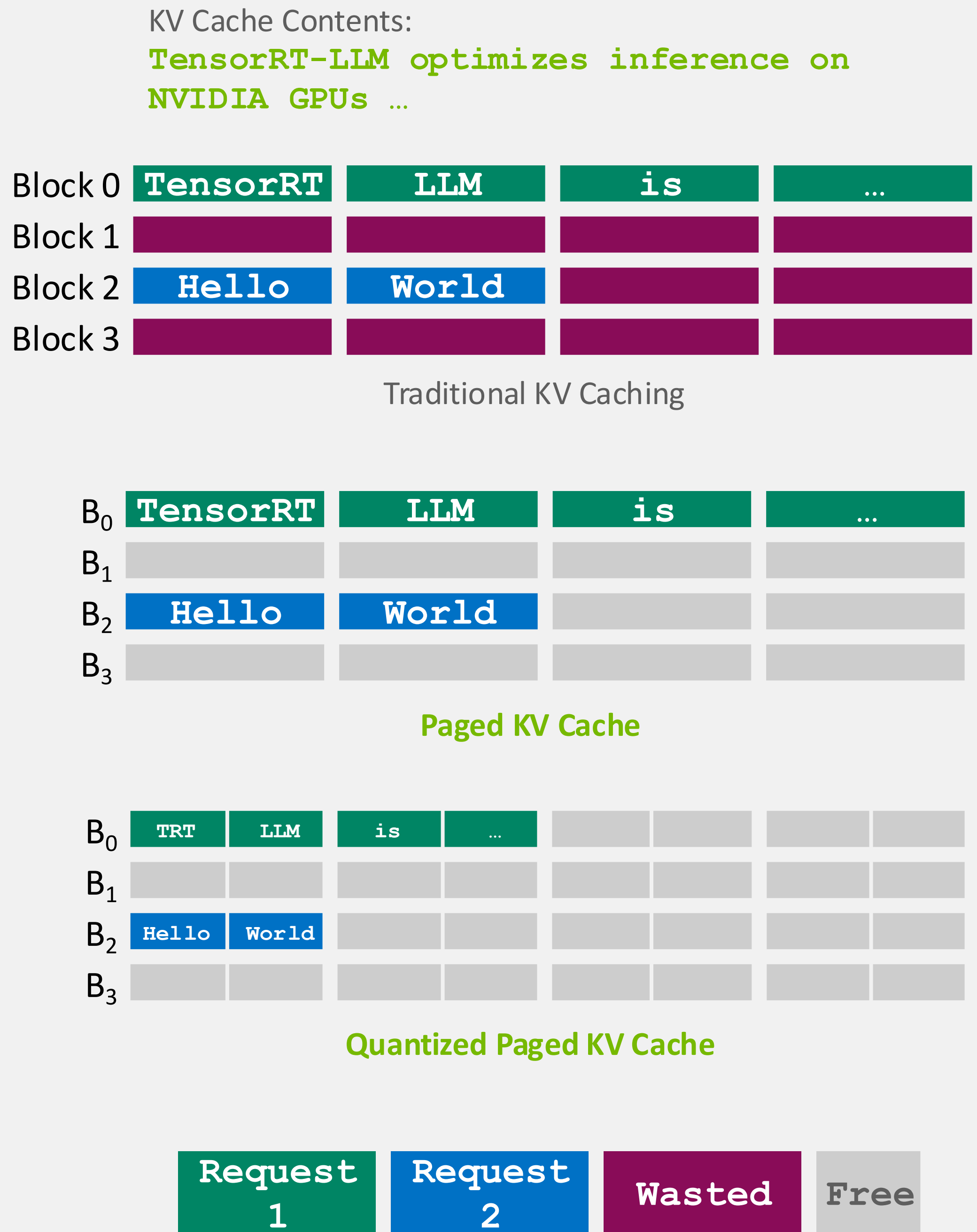
Paged KV Cache improves memory consumption & utilization

- Stores keys & values in non-contiguous memory space
- Allows for reduced memory consumption of KV cache
- Allocates memory on demand

Quantized KV Cache improves memory consumption & perf

- Reduces KV Cache elements from 16b to 8b (or less!)
- Reduces memory transfer improving performance
- Supports INT8 / FP8 KV Caches

Both allow for increased peak performance



Multi-Modal Support

Current support & adding more

- TensorRT-LLM supports BLIP, LLaVa, & Nougat VLMs
  - Including many derivatives of these models
- Utilizes TensorRT & TensorRT-LLM
  - Vision encoder in TensorRT
    - Standard ONNX export path to TRT
  - LLM running in TensorRT-LLM
  - Output of Vision encoder passed to TensorRT-LLM
- Any model similar to the supported can be added
  - Replace vision encoder or LLM with appropriate model
  - See [examples/multimodal]

Multi-Modal

This document shows how to run multimodal pipelines with TensorRT-LLM, e.g. from image+text input modalities to text output.

Multimodal models' LLM part has an additional parameter `--max_multimodal_len` compared to LLM-only build commands. Under the hood, `max_multimodal_len` and `max_prompt_embedding_table_size` are effectively the same concept, i.e., prepended/concatenated embeddings (either multimodal feature embeddings or prompt tuning embeddings) to the LLM input embeddings. The multimodal features from the visual encoder of shape `[batch_size, num_visual_features, visual_hidden_dim]` is flattened as `[batch_size * num_visual_features, visual_hidden_dim]` and passed like a prompt embedding table.

We first describe how to run each model on a single GPU. We then provide general guidelines on using tensor parallelism for LLM part of the pipeline.

- BLIP2-T5
- BLIP2-OPT
- LLaVA and VILA
- Nougat
- Enabling tensor parallelism for multi-GPU

BLIP2-T5

1. Download Huggingface weights and convert original checkpoint to TRT-LLM checkpoint format following example in `examples/enc_dec/README.md`.

```
export MODEL_NAME=flan-t5-xl
git clone https://huggin

python ../enc_dec/t5/con
    -o tmp/trt_models/${MODEL_NAME} --weight_data_type float32 \
    --inference_tensor_para_size 1
```

Multi-Modal Examples

2. Build TRT-LLM engine from TRT-LLM checkpoint

```
python ../enc_dec/build.py --model_type t5 \
    --weight_dir tmp/trt_models/${MODEL_NAME}/tp1 \
    --output_dir trt_engines/${MODEL_NAME}/1-gpu \
    --engine_name ${MODEL_NAME} \
    --remove_input_padding \
    --use_bert_attention_plugin \
    --use_gpt_attention_plugin \
    --use_gemm_plugin \
    --dtype bfloat16 \
    --max_beam_width 1 \
    --max_batch_size 8 \
    --max_encoder_input_len 924 \
    --max_output_len 100 \
    --max_multimodal_len 256 # 8 (max_batch_size) * 32 (num_visual_features)
```

NOTE: `max_multimodal_len = max_batch_size * num_visual_features`, so if you change max_batch_size, max multimodal length **MUST** be changed accordingly.

The built T5 engines are located in `./trt_engines/${MODEL_NAME}/1-gpu/bfloat16/tp1`.

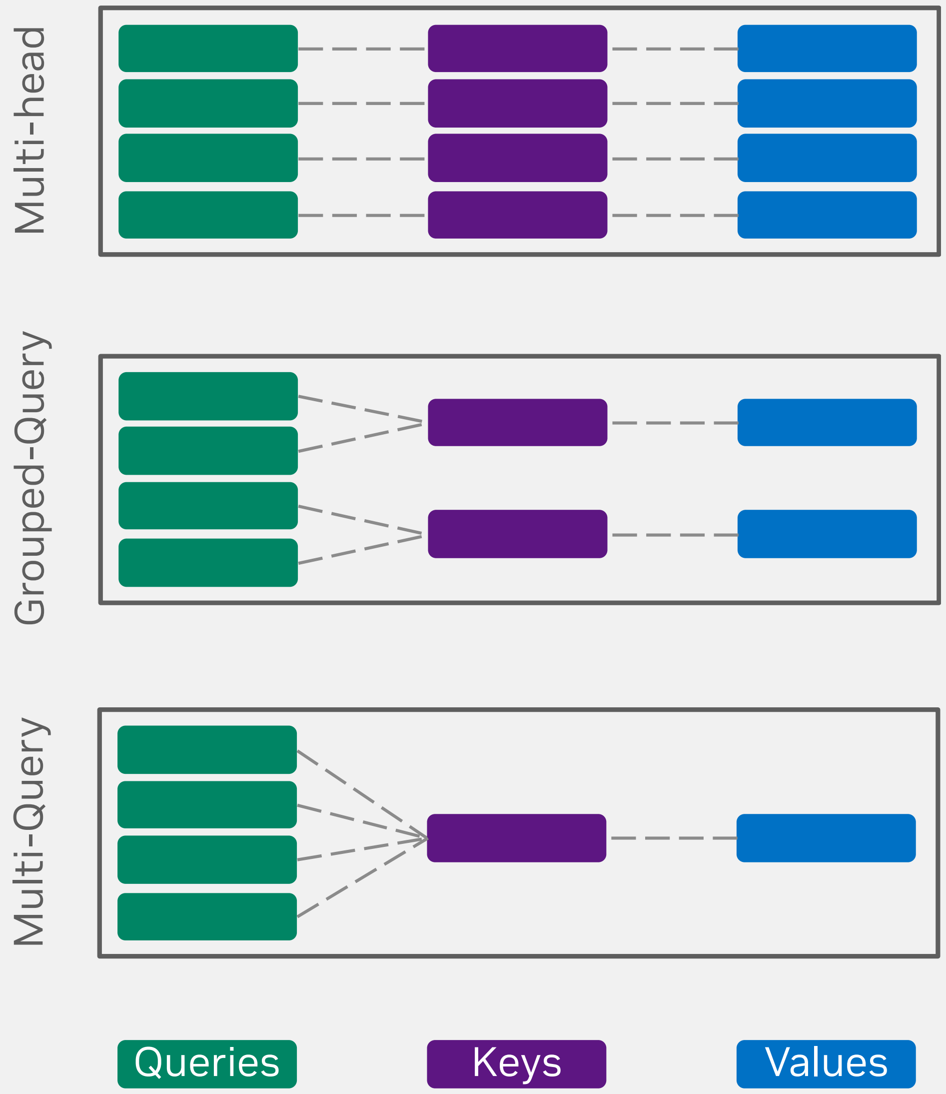
3. Build TensorRT engines for visual components

```
python build_visual_engine.py --model_type ${MODEL_NAME} --model_path tmp/hf_models/${MODEL_NAME} --max_batch_size 8
```

Optimized Attention

Custom Implementations for Attention

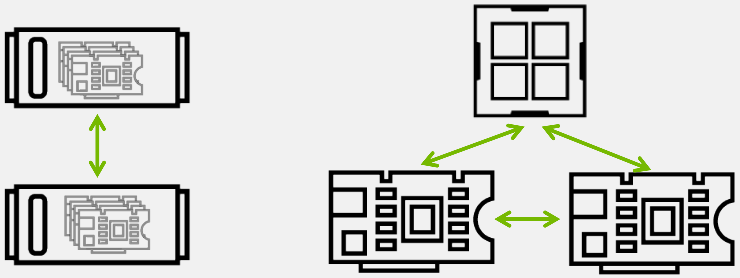
- Custom optimized CUDA kernels for Attention
  - Similar to FlashAttentionV2
- Optimized for A100 & H100
- Kernels for Encoder & Decoder, as well as context & prefill
- Supports MHA, MQA, GQA



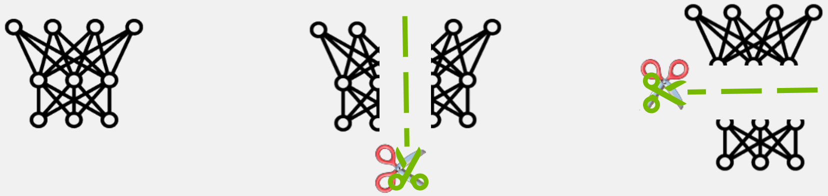
Multi-GPU Multi-Node

Sharding Models across GPUs

- Supports Tensor & Pipeline parallelism
- Allows for running very large models (tested up to 530B)
- Supports multi-GPU (single node) & multi-node
- TensorRT-LLM handles communication between GPUs
- Examples are parametrized for sharding across GPUs



Multi-Node   Multi-GPU



No Parallelism   Tensor Parallel   Pipeline Parallel