Speed-of-Light AI Inference on NVIDIA RTX Windows Workstation and Gaming PCs

Abel Brown, Engineering Mgr – TensorRT | GTC 2024

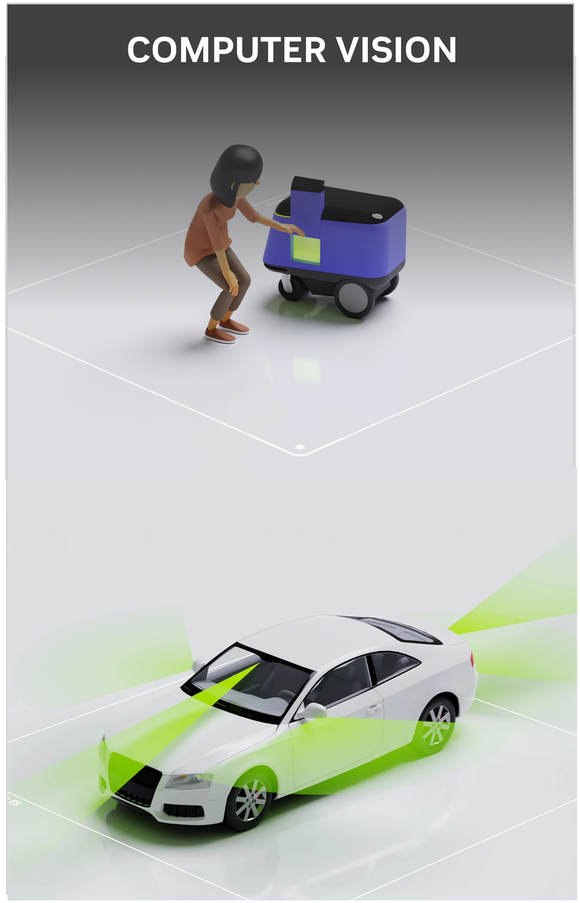
Gunjan Mehta, Product Mgr – Inference Product | GTC 2024

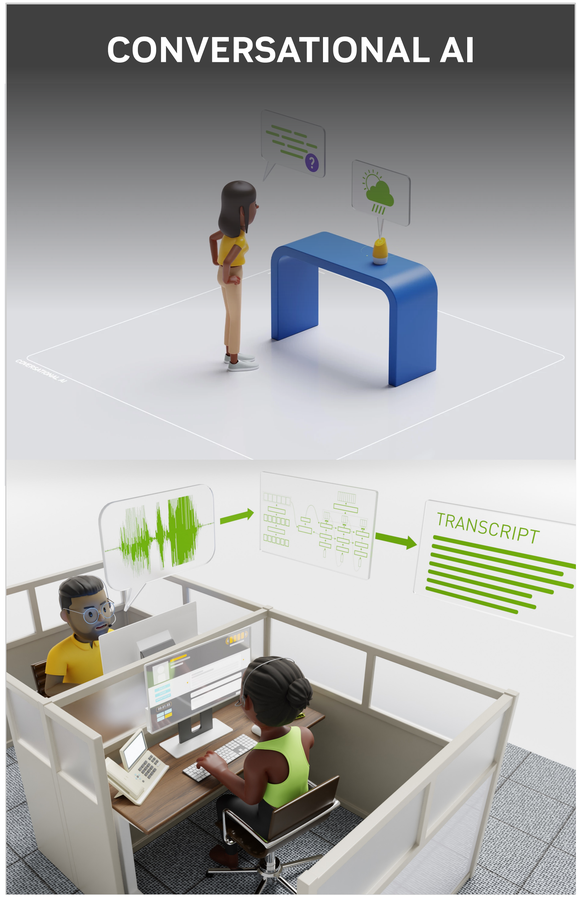
Agenda

- TensorRT Introduction
- AI inference on PCs
- New features for improved PC deployments
- TensorRT-LLM for Windows

AI INFERENCE DRIVES THE MODERN APPLICATIONS

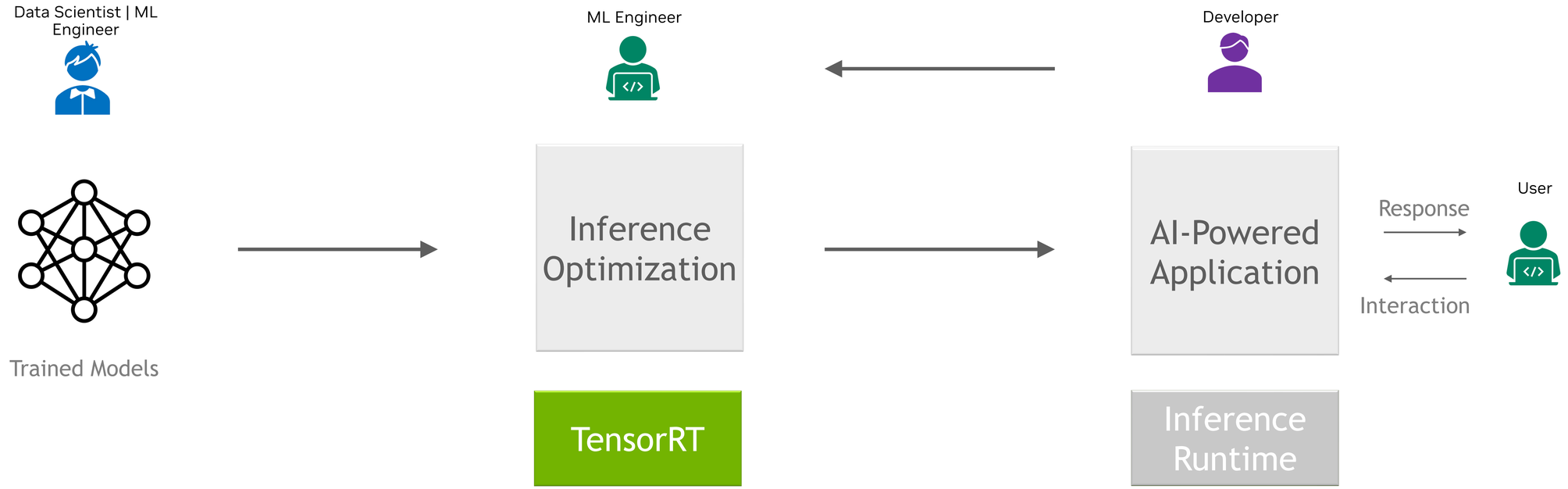
Demand for Fast, Easy Inference Deployment Greater than Ever

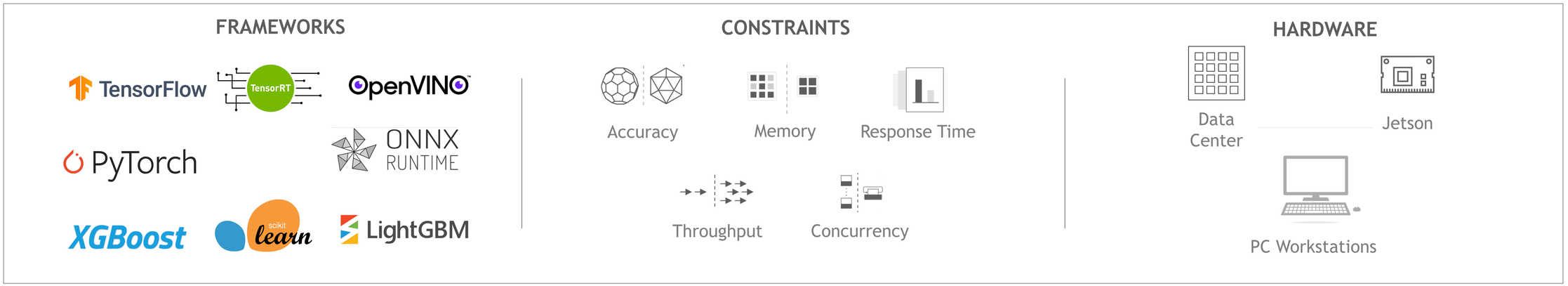




INFERENCE IS COMPLEX

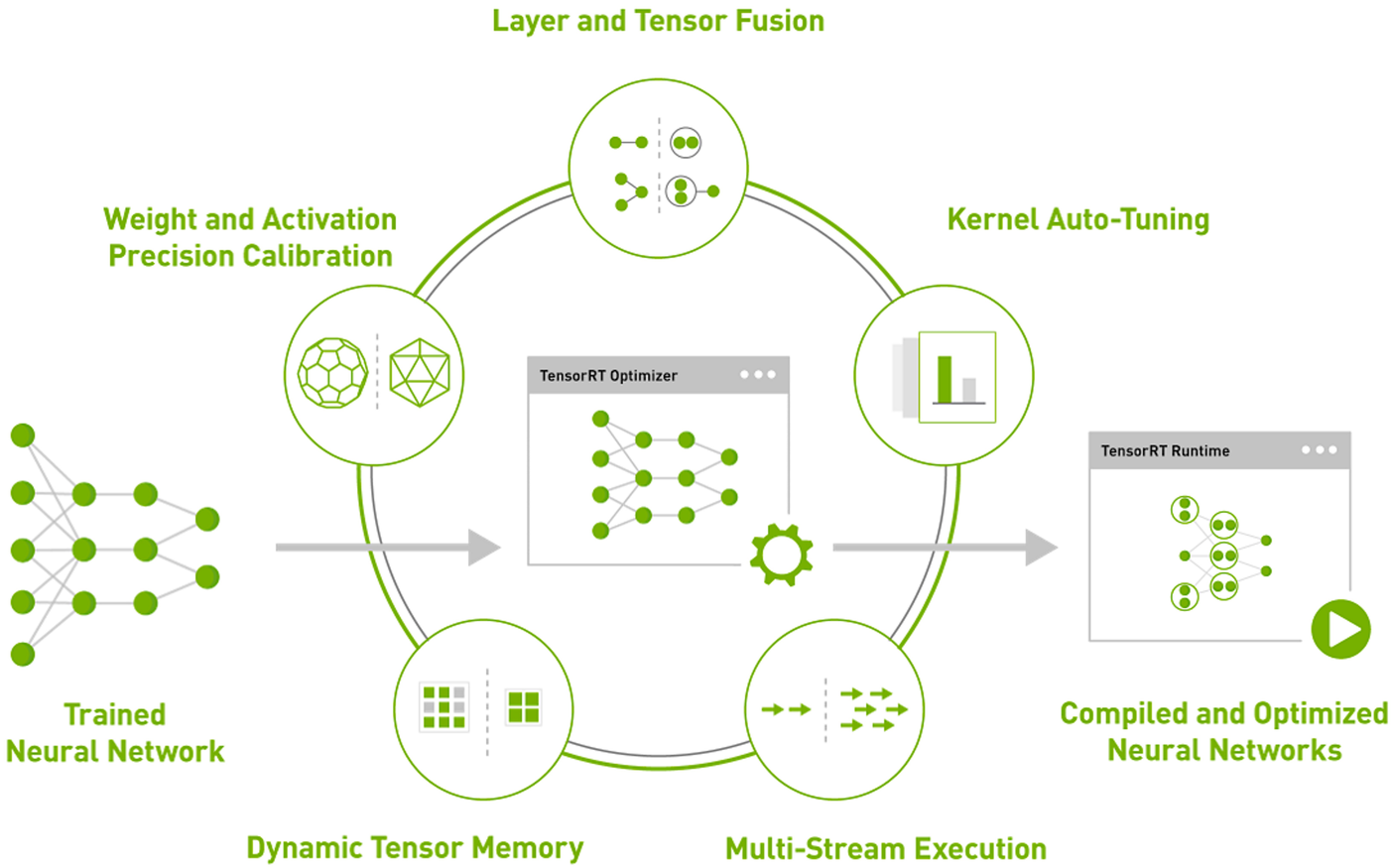
Multiple Frameworks | Competing Constraints | Maximized Utilization | Multiple Personas





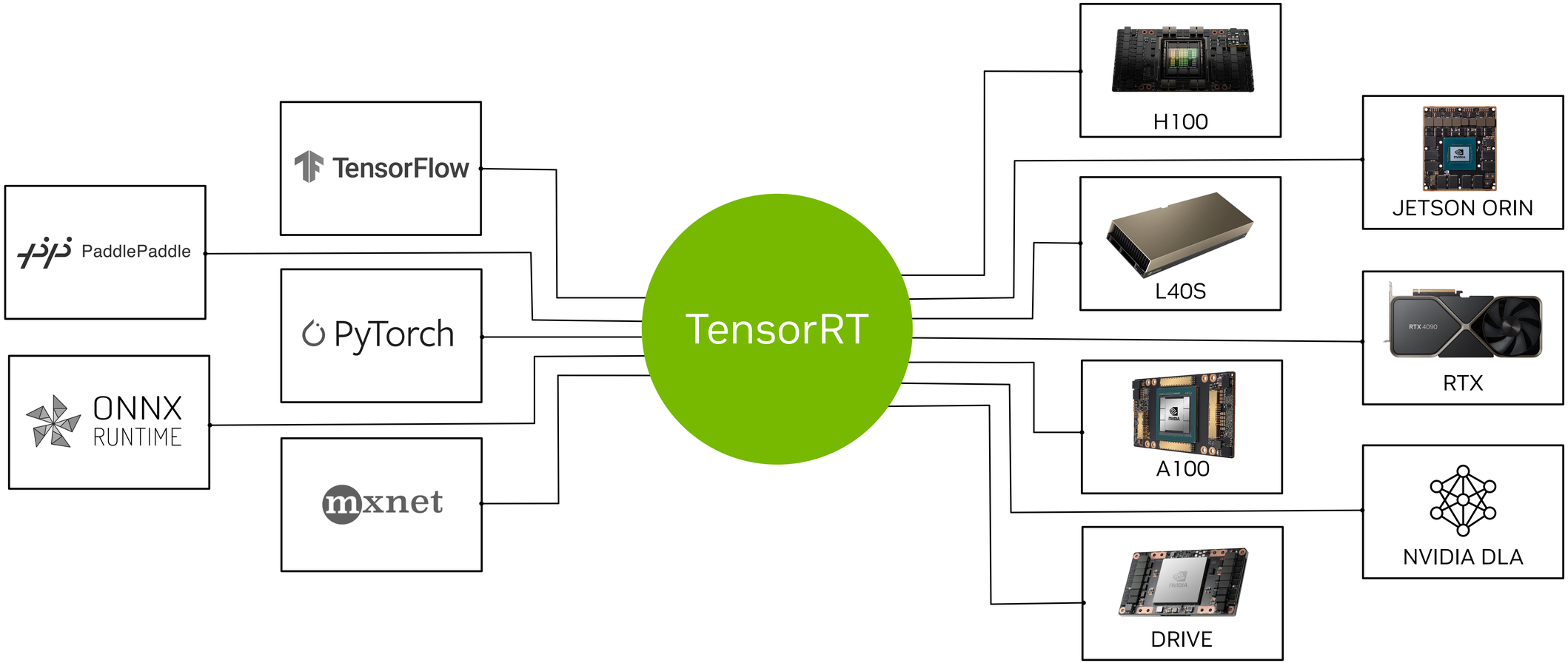
TensorRT Overview

DL Inference Optimization



NVIDIA TensorRT

From Every Framework, Optimized for Each Target Platform



WORLD LEADING INFERENCE PERFORMANCE

TensorRT Accelerates Every Workload

BEST IN CLASS RESPONSE TIME AND THROUGHPUT **vs CPUs**

36X

Computer Vision
< 7ms

583X

Speech Recognition
< 100ms

21X

NLP
< 50ms

10X

Reinforcement Learning

178X

Text-to-Speech
< 100ms

12X

Recommenders
< 1 sec

NVIDIA TensorRT Adopted Everywhere

The World's Most Advanced Inference SDK

GROWING ECOSYSTEM
Embraced by developers

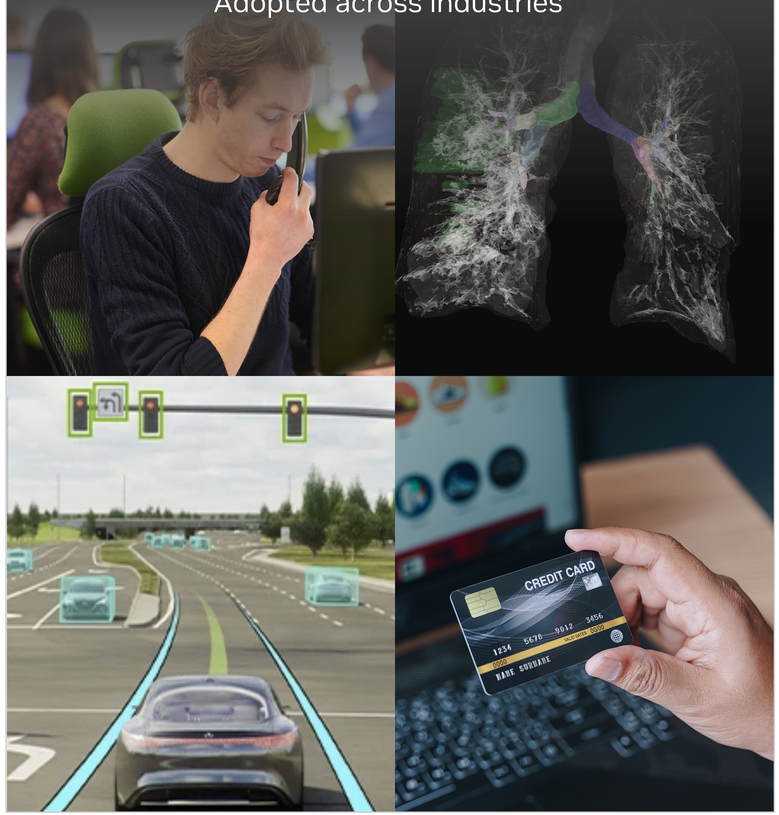
4.4M
Downloads

400K+
Developers

600+
Inference Services

CUSTOMERS
Trusted by 30,000+ companies

CLOUD TO EDGE
Adopted across industries



TensorRT Operation

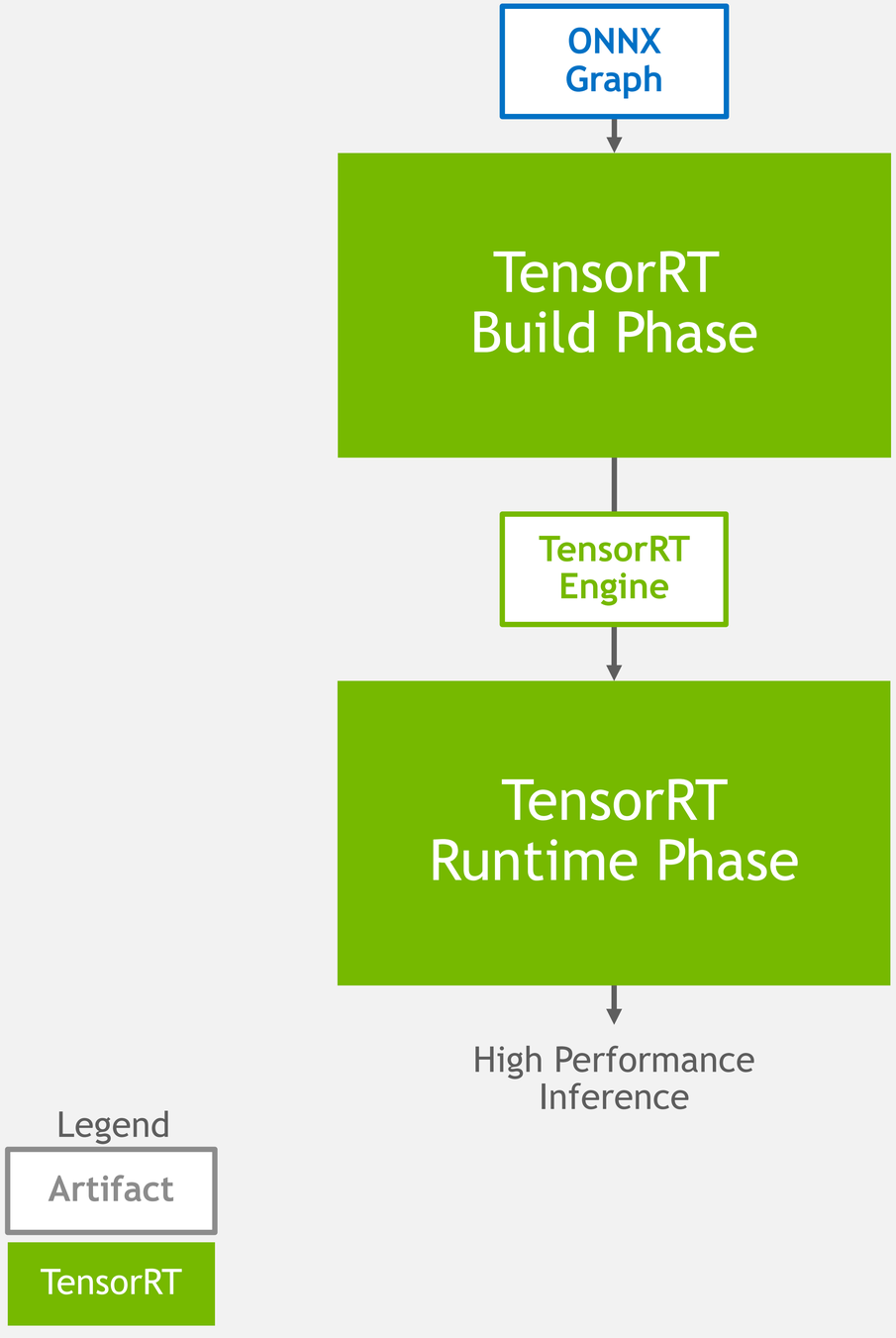
Compilation & Runtime

TensorRT has two steps

- Build phase
  - Compilation / Optimization
- Runtime phase
  - Deserialization / Execution

Each step has discrete components & APIs

- Python & C++ APIs available
- `trtexec` CLI available



TensorRT Operation

Build Phase

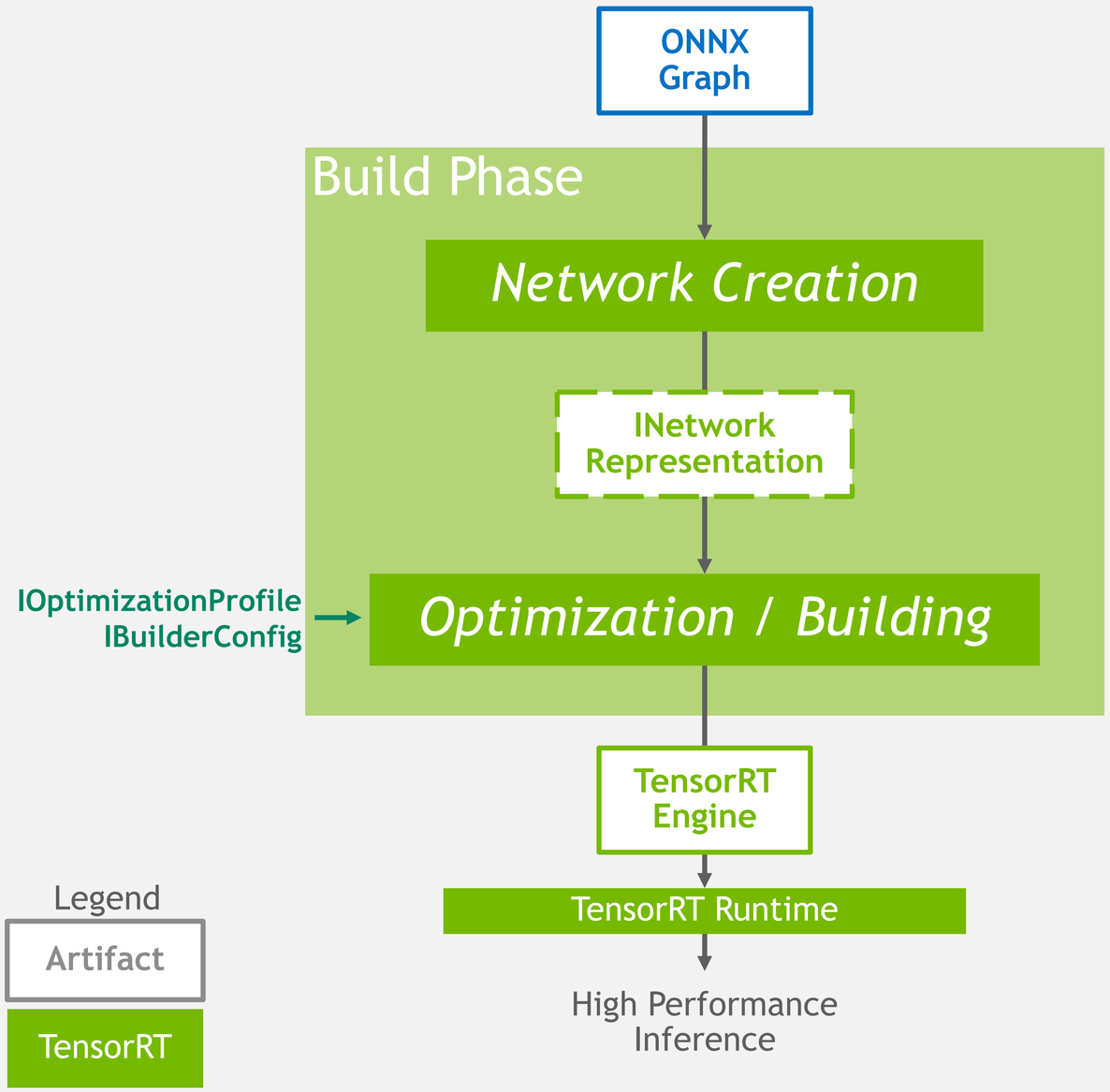
TensorRT performs AoT Compilation with 2 key steps - INetwork Creation & Building

INetwork is a TensorRT definition for the model

- Can be created automatically through the ONNX parser, FW-TRTs, or dedicated APIs
- I/O tensors are mapped at this stage

The Builder performs the compilation step

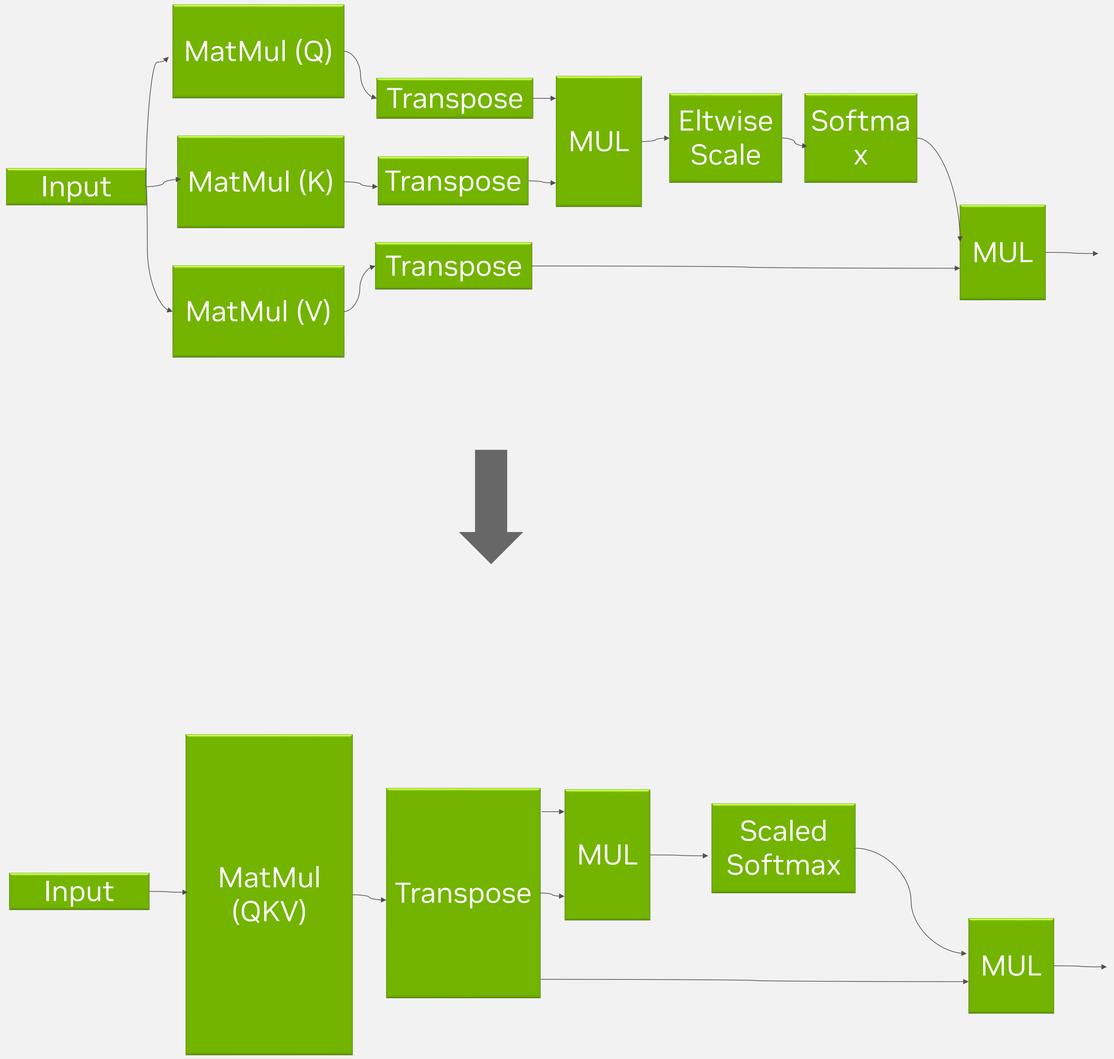
- IOptimizationProfile(s) provide the range of inputs to optimize
- IBuilderConfig allows for configuring TensorRT compilation
  - Allow for trade-offs between performance and memory
- Applies TensorRT graph optimizations
  - Constant folding, dead code elimination, etc.
- Outputs Serialized artifact built for target GPU



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

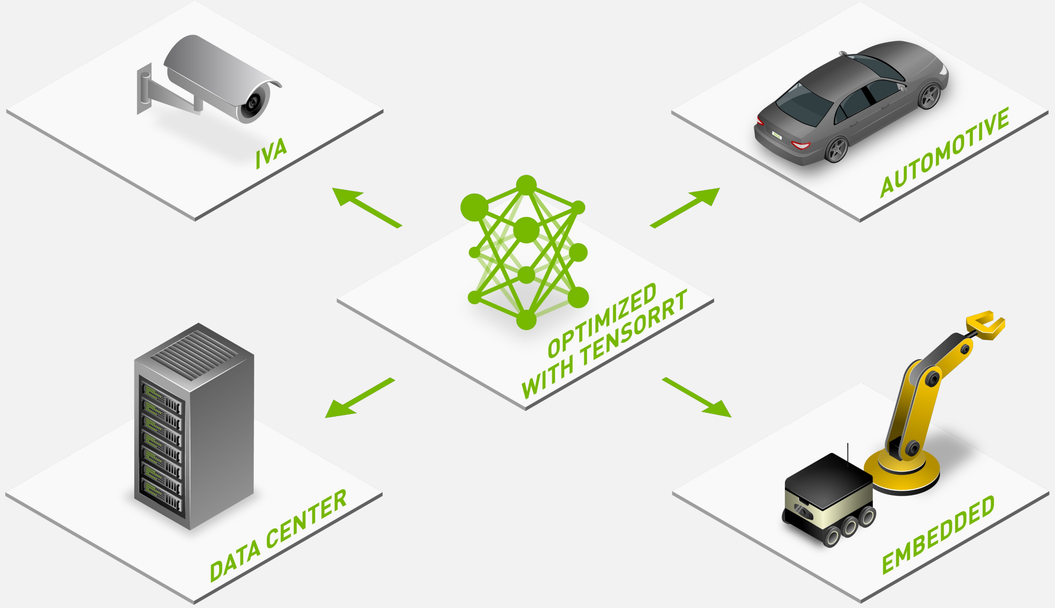


developer.nvidia.com/tensorrt

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



AI Inference on PC Workstations

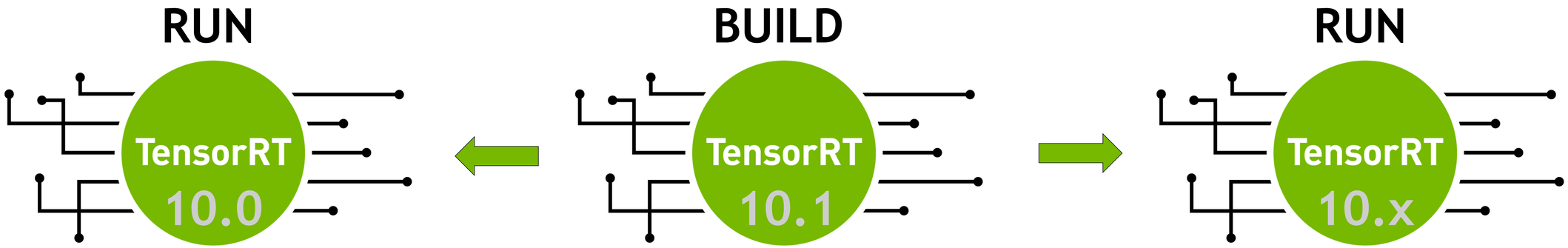
Speed-of-Light AI Inference on NVIDIA RTX

Windows Workstation and Gaming PCs

- What makes AI inference on PCs a unique challenge?
- Dozens of SKUs to support: NVIDIA SKUs {GeForce, Quadro, Laptops, etc}, other IHVs, CPU
- Wide active install base: Pascal, Turing, Ampere, Ada, *Ada-Next*
- Balance between inference perf and usability (4ms vs 2ms $\leftrightarrows$ 1min compile vs 5min compile)
- Interactive inference in Apps & Games; adapt to changing resource availability
- Strict limitations on shipment size
- Time to first inference
- Workflow, Lifecycle Management, Maintainability, Portability, Ease of Deployment

Version Compatability

TensorRT 10.x for Windows



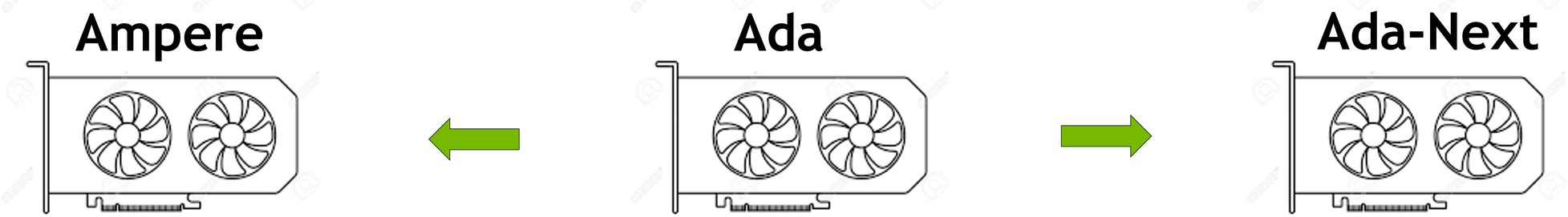
Build with one version of NVIDIA TensorRT, run with another version (within a major version).

- Version-compatible plans built with NVIDIA TensorRT 10.0 will run with NVIDIA TensorRT 10.x
- Avoids the need to rebuild a plan when updating NVIDIA TensorRT
- Must be enabled for a plan at build time

```
builderConfig.setFlag(BuilderFlag::kVERSION_COMPATIBLE);
```

Hardware Compatibility

TensorRT 10.x for Windows



Allows you to run a single engine across different hardware architectures.

- Hardware-compatible engines built with either NVIDIA ***Ampere*** or ***Ada*** or ***Ada-Next*** GeForce GPUs will work on the others with some relative performance loss compared to their native engines
- Avoids the need to rebuild an engine when updating NVIDIA GPU in the workstation
- Does not support Turning & earlier GPUs
- Must be enabled for an engine at build time

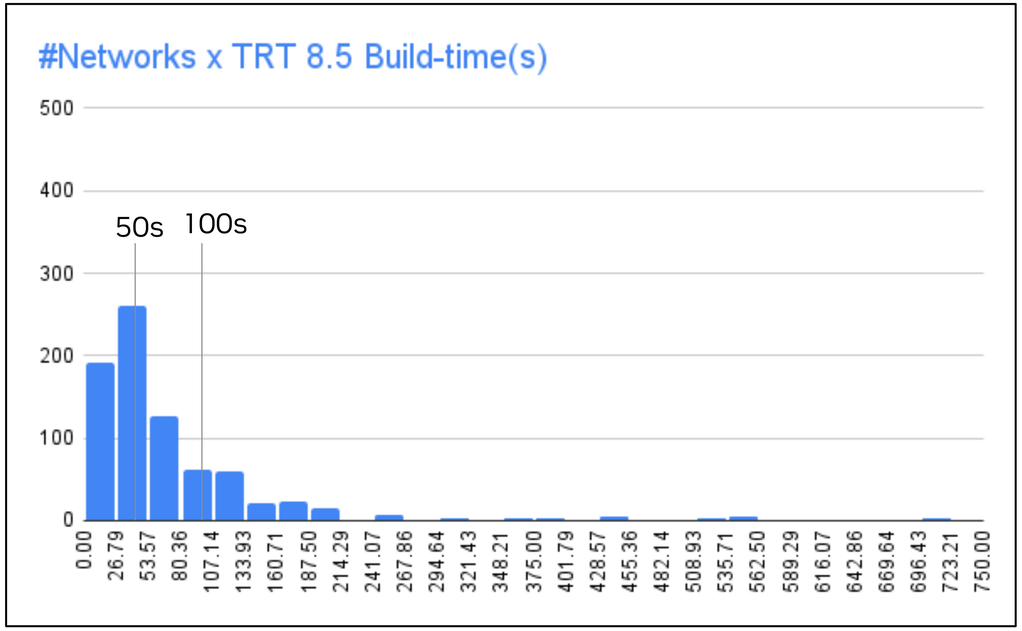
```
config->setHardwareCompatibilityLevel(
                        HardwareCompatibilityLevel::kAMPERE_PLUS);
```

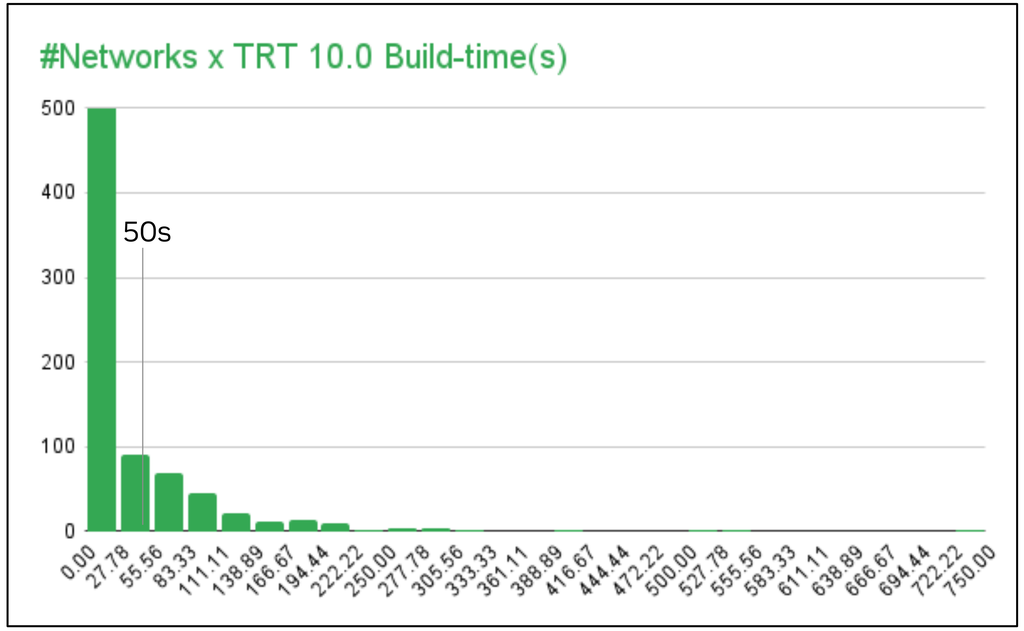
Faster Builds

Constant improvements in build time perf to simplify on-device builds

Goal: <15s
May'24

- CPU Multi-threading
- KGen Caching
- Code Optimization
- Improvements in Timing Model
- And many more tactics....



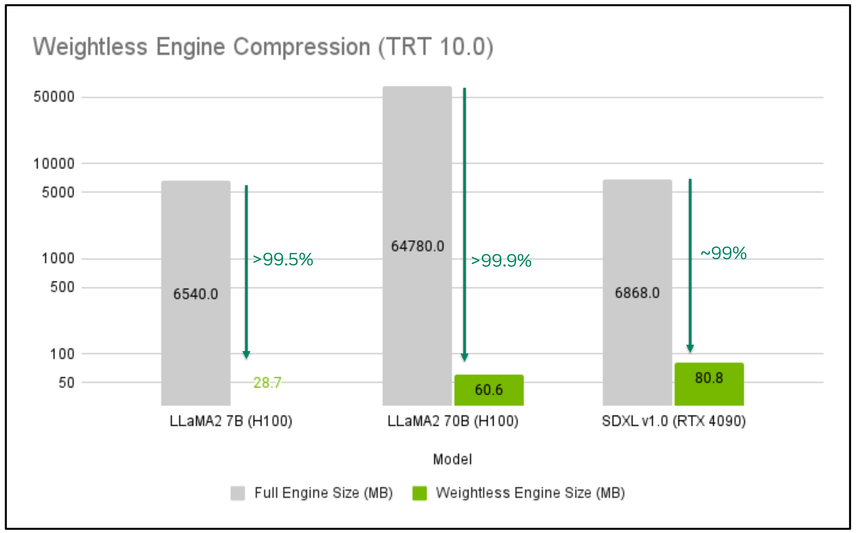


Profiled over 800+ CNN networks from customers

TensorRT Weightless Engines

Reducing a TensorRT Engine to just its execution path

- TRT optimized engines with just CUDA kernels, no weights
- Weights are refitted on end-user device in seconds
- Engine size savings between 85-99%
- Refit functionality within lean runtime (~35 MB); Serializable after refit
- Simple flag for weightless build: **`kSTRIP_PLAN`**
- ONNX Parser contains the Refitter
- Ship optimized engines for each SKU without size explosion
- Works with TRT-LLM engines as well
- Look out for python/sample_weight_stripping
- Will be available in ONNX-Runtime 1.18 (TRT EP) *~end of Jun'24*



With traditional TRT engines :->

Step 1
ISV trains models (N)

eg:
for ~25 models,
100MB each
Size = 2.5GB

Step 2
For each model, ISV builds TRT engines for multiple GPUs (M)

eg:
all_trt_engines/SKU * SKUs/Arch * #Archs
(2.5GB)*(2*3) = 15GB

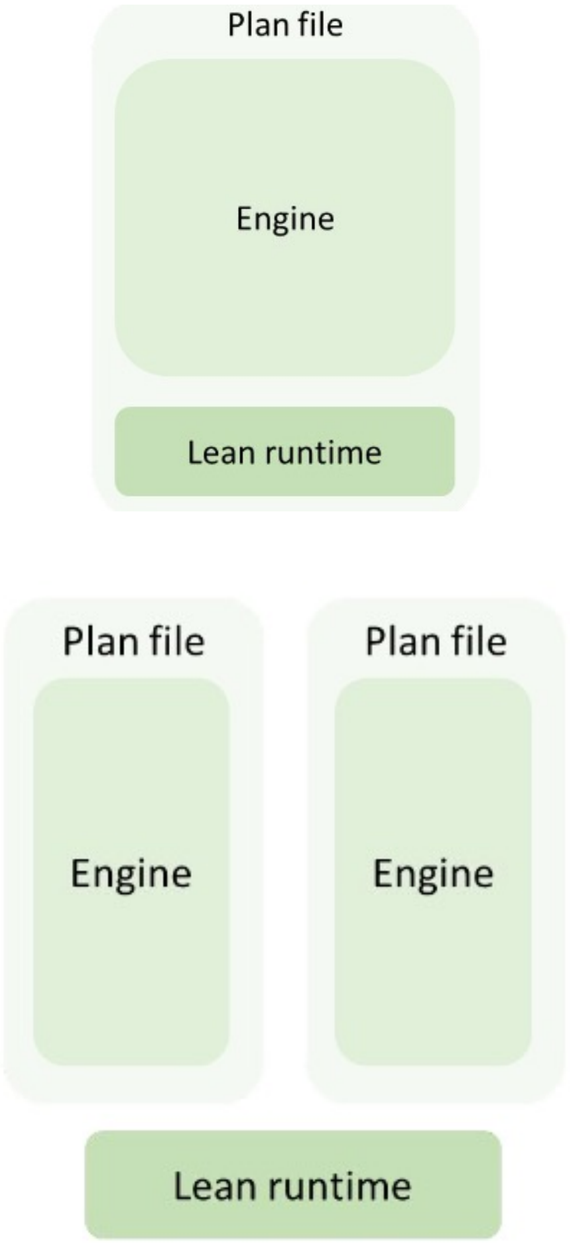
Step 3
ISV ships the app with '*M* * *N*' TRT engines + trained models for other vendors

eg:
Compiled_engines + original models
(2.5GB)*(2*3) + 2.5GB = 17.5GB

TensorRT Lean Runtime

Reduce your deployment footprint

- Lean Runtime (libnvinfer_lean.so/.dll): ~35MB
- Generated for Version Compatible Engines
- Reusable between Engines
- Avoid shipping full builder & runtime (libnvinfer); significant size reduction for pre-built engine deployment:
  2.5GB → 50MB
- Supports weightless engines



Stable Diffusion Improvements

Addressing deployment challenges

- Weight de-duplication across different optimization profiles
- ~99% engine size compression with Weightless engine
- Fine-grained runtime memory allocation for activations with `kUSER_MANAGED`
- Faster weight refitting on GPU than CPU
- LoRA refits without ONNX export & engine rebuilding
- Additional model support coming in demo/Diffusion:
  - SD 1.5, 2.0, 2.1
  - Stable Video Diffusion
  - Stable Cascade
  - LCM-LoRA
  - SDXL Turbo
  - INT8/FP8 quantization of SDXL

CUDA In Graphics

A step forward in supporting DL inside Gaming

- Previously, significant overhead in CUDA * DX context switch / interop
- Simultaneous execution of graphics and compute to increase utilization & avoid context switch penalty
- Benefits apps/games using CUDA with DX/Vulkan frameworks
- Coming in CUDA Driver r555 (early Summer)
- TensorRT will be able to share the GPU with other applications
- TensorRT exposes the functionality using builder flag: `kTACTIC_SHARED_MEMORY`
  - Set to device max shared memory size (eg: 48kB for Ampere)

More features in the pipeline…(after TRT 10.0)

TensorRT is committed to improving Windows adoption

- INT4 weight-only quantization on RTX
  - Q/DQ node definitions added to ONNX spec
  - Explicit (Q/DQ) quantization support
  - Model Quantization Toolkit to calibrate and export INT4 ONNX models
  - Usable with DXML
- Weight Streaming on RTX
  - Fit much larger models than size of the GPU VRAM using weight streaming from DRAM
- Faster build times (<15s) and enhanced portability (run anywhere)
  - More details to come around Microsoft BUILD Conference (May'24)

TRT 10.0 coming in end-of-April'24

TensorRT
Framework Integrations

Torch-TensorRT Overview

PyTorch TensorRT Integration

PyTorch + TensorRT

The power of NVIDIA TensorRT integrated in PyTorch.

High-performance deep learning inference directly from framework.

PyTorch, the PyTorch logo and any related marks are trademarks of The Linux Foundation.

TensorRT Logo Source: https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorrt/

Torch-TensorRT GitHub Repository: https://github.com/pytorch/TensorRT

NVIDIA

Torch-TensorRT JIT Approach

Highlights

The benefits of *torch.compile…*

- Complex Python Code Handling
- Automatic Graph Segmentation
- Just-In-Time Recompilation
- Easy-to-use API

…with the optimization of TensorRT!

- Layer + Tensor Fusion
- Kernel Auto-Tuning targeting specific GPUs
- Layer Precision Customization
- Dynamic Shape Support*

- Stable Diffusion† Text-To-Image FP16 with 50 UNet Iterations takes **0.94s**

* Coming soon in Torch-TensorRT
† Benchmarked on an RTX4090 24GB

PyTorch, the PyTorch logo and any related marks are trademarks of The Linux Foundation.
TensorRT Logo Source: https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorrt/

TensorRT-LLM for Windows (Beta)

Large Language Model Ecosystem

Rapid evolution makes optimization challenging

- Increasing rate of new foundational LLMs being released
  - Llama, Falcon, Starcoder, ChatGLM, MPT, & more
- New operators & customization techniques makes optimization a moving target
- Latest models continue to be very large for best accuracy
  - 70-200 Billion parameter or more

Need a **performant**, **robust**, & **extensible solution** for **cost-effective**, **real-time** LLM deployments

LLaMa | GPT | Falcon | Starcoder | ChatGLM | MPT | ...

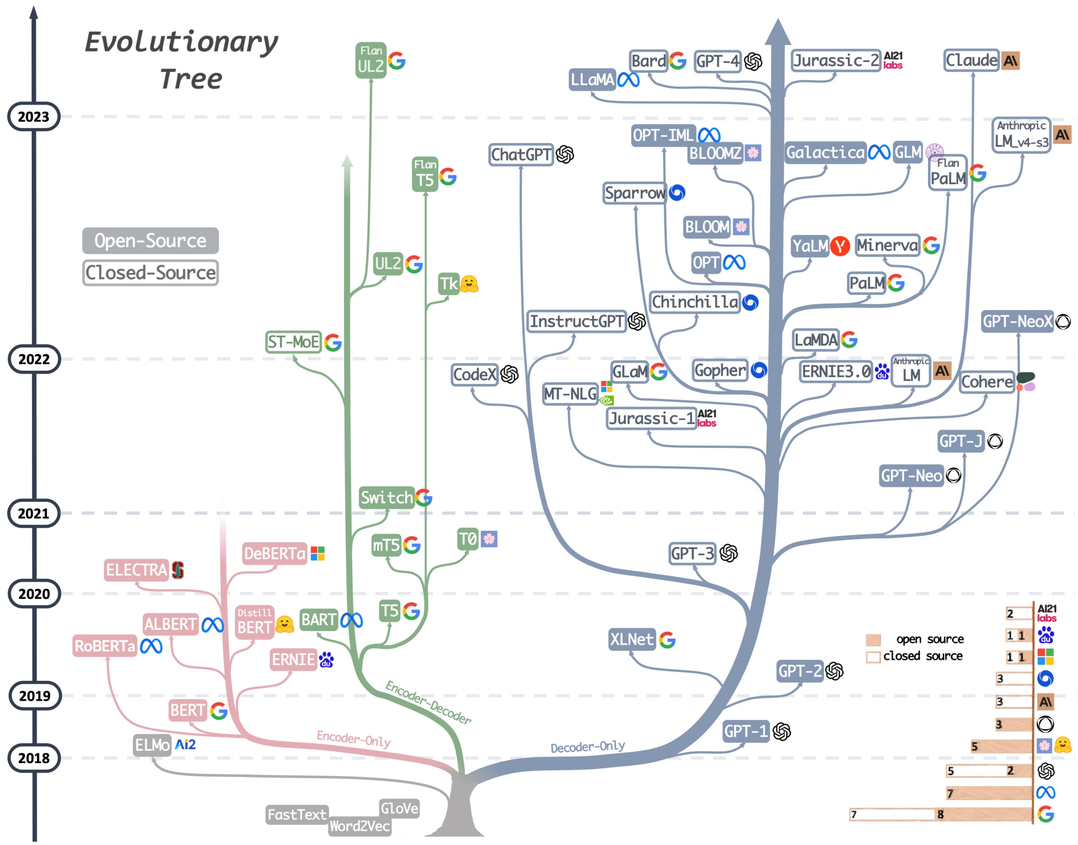


Image from [Mooler0410/LLMsPracticalGuide]

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., ... Hu, X. (2023). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/2304.13712

TensorRT-LLM in the *DL Compiler* Ecosystem

TensorRT-LLM builds on TensorRT Compilation

- **TensorRT-LLM**
  - Built *on-top of* TensorRT
    - Leverages TensorRT for general graph optimizations & fast kernels
  - Adds LLM specific optimizations:
    - KV Caching & Custom MHA Kernels
    - Inflight batching, Paged KV Cache (Attention)
    - Multi-GPU, Multi-Node
    - *& more*
  - *ONLY for LLMs*

- **TensorRT**
  - General purpose Deep Learning Inference Compiler
    - Graph rewriting, constant folding, kernel fusion
    - Optimized GEMMs & pointwise kernels
    - Kernel Auto-Tuning
    - Memory Optimizations
    - *& more*
  - *All AI Workloads*

TensorRT-LLM

LLM specific optimizations:
- KV Caching
- Multi-GPU, Muti-Node
- Custom MHA optimizations
- Paged KV Cache (Attention)
- *etc…*

TensorRT "Core"

General Purpose Compiler
- Optimized GEMMs & general kernels
- Kernel Fusion
- Auto Tuning
- Memory Optimizations
- Multi-stream execution

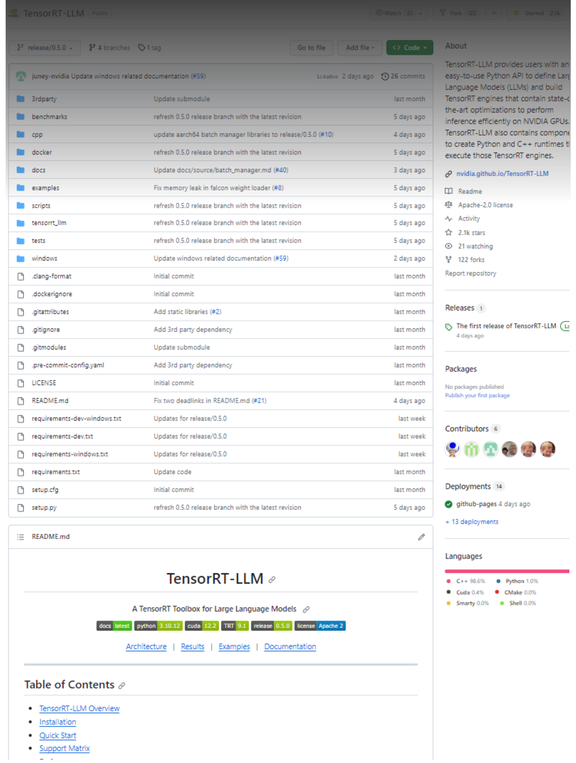
NVIDIA CONFIDENTIAL. DO NOT DISTRIBUTE.

TensorRT-LLM Available Now!

Key resources for TensorRT-LLM

TensorRT-LLM Github

Source for TensorRT-LLM



Getting Started Blog

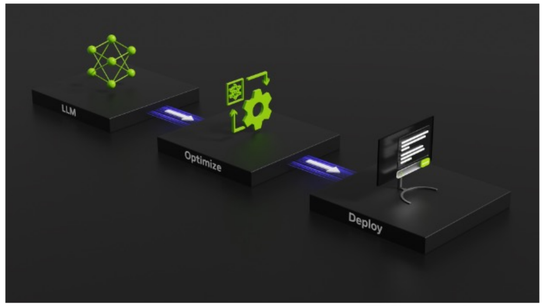
Learn to optimize and deploy TensorRT-LLM with Triton Server

Generative AI

Optimizing Inference on Large Language Models with NVIDIA TensorRT-LLM, Now Publicly Available

Oct 19, 2023

By Neal Vaidya, Fred Oh and Nick Comly



Today, NVIDIA announces the public release of TensorRT-LLM to accelerate and optimize inference performance for the latest LLMs on NVIDIA GPUs. This open-source library is now available for free on the /NVIDIA/TensorRT-LLM GitHub repo and as part of the NVIDIA NeMo framework.

Large language models (LLMs) have revolutionized the field of artificial intelligence and created entirely new ways of interacting with the digital world. But, as organizations and application developers around the world look to incorporate LLMs into their work, some of the challenges with running these models become apparent.

Put simply, LLMs are large. That fact can make them expensive and slow to run without the right techniques.

Many optimization techniques have risen to deal with this, from model optimizations like kernel fusion and quantization to runtime optimizations like C++ implementations, KV caching, continuous in-flight batching, and paged attention. It can be difficult to decide which of these are right for your use case, and to navigate the interactions between these techniques and their sometimes-incompatible implementations.

That's why NVIDIA introduced TensorRT-LLM, a comprehensive library for compiling and optimizing LLMs for inference.

TensorRT-LLM documentation

API docs, arch overviews, & perf data

Welcome to TensorRT-LLM's documentation! View page source

Welcome to TensorRT-LLM's documentation!

Contents:

- TensorRT-LLM Architecture
- C++ GPT Runtime
- The Batch Manager in TensorRT-LLM
- Multi-head, Multi-query and Group-query Attention
- Numerical Precision
- Performance of TensorRT-LLM
- Build From Sources
- How to debug
- How to add a new model
- Graph Rewriting Module

Python API

- tensorrt_llm.layers
- tensorrt_llm.functional
- tensorrt_llm.models
- tensorrt_llm.plugin
- tensorrt_llm.qunatization
- tensorrt_llm.runtime

C++ API

- cpp/runtime

Indices and tables

- Index
- Module Index
- Search Page

Now Available!
TensorRT-LLM v0.8

- Models:
  - Llama, CodeLlama, Phi-2, Gemma
- New Features
  - Speculative Decoding
  - System Prompt Caching
  - Python bindings for GPTManager & GPTSession
- New *X*QA kernel
  - Up to 2x improvement on models with GQA & MQA
- Cool Examples:
  - ChatWithRTX
  - trt-llm-rag-windows
  - Gemma on RTX

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

Github: NVIDIA/TensorRT-LLM/v0.8.0

TensorRT-LLM 0.7.1 Release

Latest

Compare

kaiyux released this Dec 26, 2023 · 39 commits to main since this release · v0.7.1 · 80bc075

Hi,

We are very pleased to announce the 0.7.1 version of TensorRT-LLM. It has been an intense effort, and we hope that it will enable you to easily deploy GPU-based inference for state-of-the-art LLMs. We want TensorRT-LLM to help you run those LLMs very fast.

This update includes:

- Models
  - BART and mBART support in encoder-decoder models
  - FairSeq Neural Machine Translation (NMT) family
  - Mixtral-8x7B model
    - Support weight loading for HuggingFace Mixtral model
  - OpenAI Whisper
  - Mixture of Experts support
  - MPT - Int4 AWQ / SmoothQuant support
  - Baichuan FP8 quantization support
- Features
  - [Preview] Speculative decoding
  - Add Python binding for `GptManager`
  - Add a Python class `ModelRunnerCpp` that wraps C++ `gptSession`
  - System prompt caching
  - Enable split-k for weight-only cutlass kernels
  - FP8 KV cache support for XQA kernel
- Bug fixes
  - Fix LLaMA GPTQ failure #580
  - Fix Python binding for InferenceRequest issue #528
  - Fix CodeLlama SQ accuracy issue #453
  - Minor bug fixes
- Performance
  - MMHA optimization for MQA and GQA
  - LoRA optimization: cutlass grouped gemm
  - Optimize Hopper warp specialized kernels
  - Optimize AllReduce for parallel attention on Falcon and GPT-J
  - Enable split-k for weight-only cutlass kernel when SM>=75
- Documentation
  - Add documentation for new builder workflow

Currently, there are two key branches in the project:

- The rel branch is the stable branch for the release of TensorRT-LLM. It has been QA-ed and carefully tested.
- The main branch is the dev branch. It is more experimental.

We are updating the main branch regularly with new features, bug fixes and performance optimizations. The stable branch will be updated less frequently, and the exact frequencies depend on your feedback.

Thanks,

The TensorRT-LLM Engineering Team

TensorRT-LLM v0.8

Questions?

For any queries related to Windows Inference, please reach out to:

gmehta@nvidia.com