# NN Optimisation Frameworks

PRESENTED BY SIDDHARTH SHAH, LAKSHMI SABARI PRIYA JAINI
03/14/2024

# INTRODUCTION

## Neural Network optimization frameworks

- Tools and libraries designed to improve the **performance and efficiency of deep learning models**, especially during the **inference (prediction) phase**
- Deep learning models can be computationally expensive and resource-intensive when running on hardware like CPUs or GPUs.
- Examples - **Torch2Compile, TensorRT-LLM, and Optimum-ONNXruntime, Hugging Face TGI**

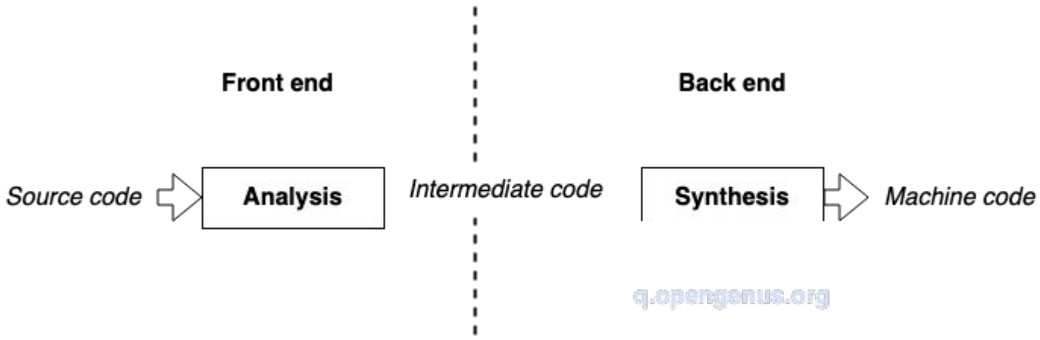
NYU

# Torch2Compile

# Introduction

- PyTorch 1.x code is largely interpreted
- Makes things slow…
- Since 2017, GPUs have become ~15x faster in compute and about ~2x faster in memory access. This lead to PyTorch internals → C++.
- This makes code less hackable and increases the barrier of entry
- PyTorch 2.0 moves internals back to Python and introduces **torch.compile**!
- New technologies: **TorchDynamo**, **AOTAutograd**, **PrimTorch** and **TorchInductor**
- **compiled_model = torch.compile(model)**

# Compiler Basics



# TorchDynamo: Acquiring Graphs reliably and fast

- It converts a NN model written in Pytorch into a computation graph
  - a graphical representation of the computations performed during the execution of a program. It captures the dependencies between different operations and variables.
- Uses Python Frame Evaluation Hook to do so
  - Frame Evaluation Hooks enable users to execute custom code before and after the execution of a Python frame.
- TorchDynamo acquires the graph 99% of the time

NYU

# TorchInductor and PrimTorch

- TorchInductor is the backend for 2.0
- Takes IR to automatically map PyTorch models into Triton code
- Writing a backend for PyTorch is challenging (2000+ operators), hence PrimTorch!
- It defines smaller, stable operator sets
  - Prim ops with about ~250 operators, which are fairly low-level. These are suited for compilers because they are low-level enough, so their fusion leads to good performance.
  - ATen ops with about ~750 operators and suited for exporting as-is, meaning that they can be directly integrated with runtime environments without the need for additional optimization steps.

NYU

# AOTAutograd: reusing Autograd for ahead-of-time graphs

- This feature optimizes the backward pass (efficient training)
- Uses autograd engine
  - Autograd engine allows for gradients to be computed automatically during backward pass
- Before torch.compile, autograd computed gradients (needed for the backpass) after the forward pass
- AOTAutograd captures the computational graph of the backward pass (gradient computation) AOT

NYU

# Supported hardware, DL models, and Performance

- Default backend TorchInductor supports CPUs and NVIDIA Volta and Ampere GPUs. It does not (yet) support other GPUs, xPUs or older NVIDIA GPUs.
- torch.compile is validated on 163 diverse open-source models across Image Classification, Object Detection, NLP, Recommender Systems, and RL.
- Models sourced from HuggingFace Transformers, TIMM, and TorchBench.
- **Results**
    - 43% faster training time on an NVIDIA A100 GPU.
    - float32 precision, average speedup = 21%; AMP precision, average speedup = 51%.

NYU

# Optimum-ONNX runtime

NYU

# Introduction

- Growing Model Sizes: Transformer-based models are expanding for complex multi-modal tasks, demanding more resources.
- Collaboration: Hugging Face and Microsoft's ONNX Runtime teams
- Optimum is a library by Hugging Face and ONNX is a runtime environment
- Optimum with ONNX Runtime: Optimum improves training times by >=35% for many popular Hugging Face models

# Support

**Supported Hardware -**

- Nvidia GPUs
- AMD Instinct GPUs, Ryzen AI NPUs
- Graphcore IPUs, Habana Gaudi Processors

**Supported Models -**

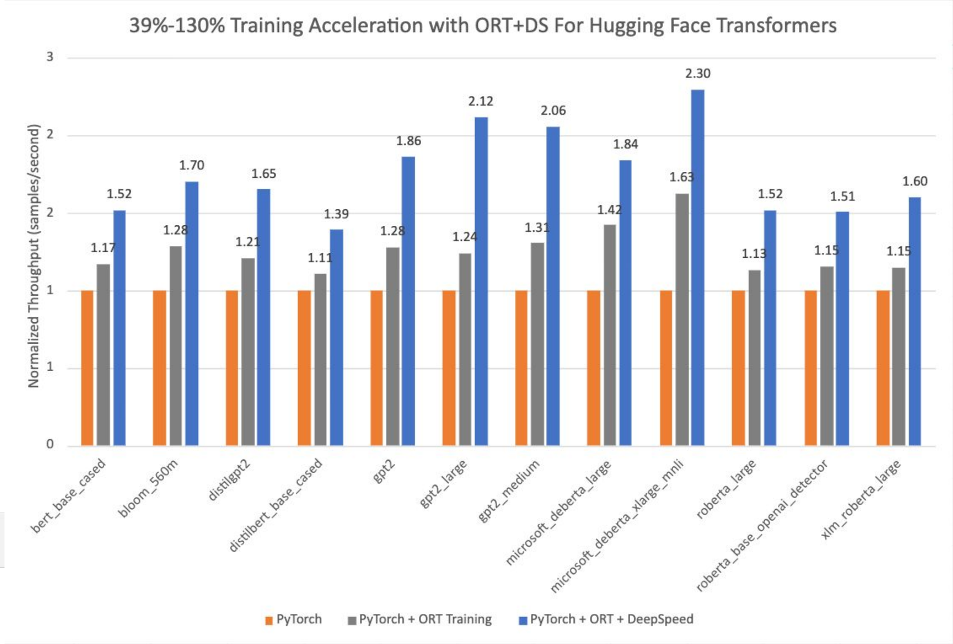
- This library supports models in the ONNX format, which can be exported from various deep learning frameworks, including PyTorch, TensorFlow, and others.

# Optimisation Techniques and features

- ONNX Runtime - Efficient memory planning, multi tensor apply for Adam Optimizer, mixed precision training, graph optimizations like node fusions and node eliminations
- ONNX Runtime + Optimum - supports features like hyperparameter search, mixed-precision training and distributed training with multiple GPUs
- ORTTrainer enables developers to combine ONNX Runtime with other third-party acceleration techniques when training models, which helps accelerate the training further and gets the best out of the hardware. e.g. DeepSpeed ZeRO-1

# Usability, Performance and Accuracy

- Speed gain of **39-130%** for Hugging Face models with Optimum when **using ONNX Runtime and DeepSpeed ZeRO Stage 1 for training.**
- PyTorch as the baseline run, only ONNX Runtime for training as the second run, and ONNX Runtime + DeepSpeed ZeRO Stage 1 as the final run
- Performed on a single **Nvidia A100 node with 8 GPUs.**




NYU

# HuggingFace TGI

NYU

# Introduction

- Library/Toolkit designed to **optimize the inference of large language models**
- TGI is an optimization framework that integrates seamlessly with the **Hugging Face Transformers library**, which is widely used in natural language processing tasks.
- Enables high-performance text generation for the most popular open-source LLMs, including Llama, Falcon, StarCoder, BLOOM, GPT-NeoX, and T5 etc.
- Easy usage for developers to optimise and deploy LLMs.

# Sample usage

Example -

1. model=teknium/OpenHermes-2.5-Mistral-7B
2. docker run --gpus all --shm-size 1g -p 8080:80 -v $volume:/data ghcr.io/huggingface/text-generation-inference:1.4 --model-id $model

```python
import requests

headers = {
    "Content-Type": "application/json",
}

data = {
    'inputs': 'What is Deep Learning?',
    'parameters': {
        'max_new_tokens': 20,
    },
}

response = requests.post('http://127.0.0.1:8080/generate', headers=headers, json=data)
print(response.json())
# {'generated_text': '\n\nDeep Learning is a subset of Machine Learning that is concerned with the dev
```

# Supported Hardware

- TGI optimized models are supported on NVIDIA A100, A10G and T4 GPUs with CUDA 12.2+.
- ROCm-enabled AMD Instinct MI210 and MI250 GPUs, with paged attention, GPTQ quantization, flash attention v2 support. Some other features are not available for AMD (ROCm)
- TGI is also supported on the following AI hardware accelerators:
  - Habana first-gen Gaudi and Gaudi2
  - AWS Inferentia2

# Supported Models

The following models are optimized and can be served with TGI, which uses custom CUDA kernels for better inference.

- BLOOM
- FLAN-T5
- Galactica
- GPT-Neox
- Llama
- OPT
- SantaCoder
- Starcoder
- Falcon 7B
- Falcon 40B
- MPT
- Llama V2
- Code Llama
- Mistral
- Mixtral
- Phi

# Optimisation Techniques

- Token streaming using Server-Sent Events (SSE)
- Quantization with bitsandbytes and GPT-Q
- Safetensors weight loading
- Tensor Parallelism for faster inference on multiple GPUs
- Optimized transformers code for inference using Flash Attention and Paged Attention on the most popular architectures

# Usability, Performance and Accuracy

- Usability - It provides a simple and user-friendly interface for optimizing and running inference on the supported large language models. Seamless integration with Hugging Face Transformers Library
- Hugging Face TGI aims to provide significant performance improvements for large language model inference, particularly on GPUs.
- Significantly reduce the memory footprint and computational requirements of the model, often with minimal impact on accuracy with optimisation techniques like Quantisation

# Nvidia TensorRT-LLM

NYU

# Introduction

- NVIDIA® TensorRT™, an SDK for high-performance deep learning inference, includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for inference applications.
- TensorRT-LLM is an extension of NVIDIA's TensorRT inference optimizer and runtime, specifically designed to accelerate large language model (LLM) inference.
- Simple open-source Python API for defining, optimizing, and executing LLMs for inference in production.
- Architectured to look similar to the PyTorch API.

# Supported Hardware

- TensorRT-LLM targets NVIDIA's GPUs, which are optimized for accelerating transformer-based models, including large language models.
- TensorRT-LLM is rigorously tested on the following GPUs:
  - H100
  - L40S
  - A100
  - A30
  - V100 (experimental)

# Supported Models

TensorRT-LLM is designed to optimize and accelerate the inference of large language models -

- Baichuan
- BART
- BERT
- Blip2
- BLOOM
- ChatGLM
- FairSeq NMT
- Falcon
- Flan-T5
- GPT
- MPT
- mT5
- OPT
- Phi-1.5/Phi-2
- Qwen
- Replit Code
- RoBERTa
- SantaCoder
- StarCoder1/StarCoder2
- T5
- GPT-J
- GPT-Nemo
- GPT-NeoX
- InternLM
- LLaMA
- LLaMA-v2
- mBART
- Mistral

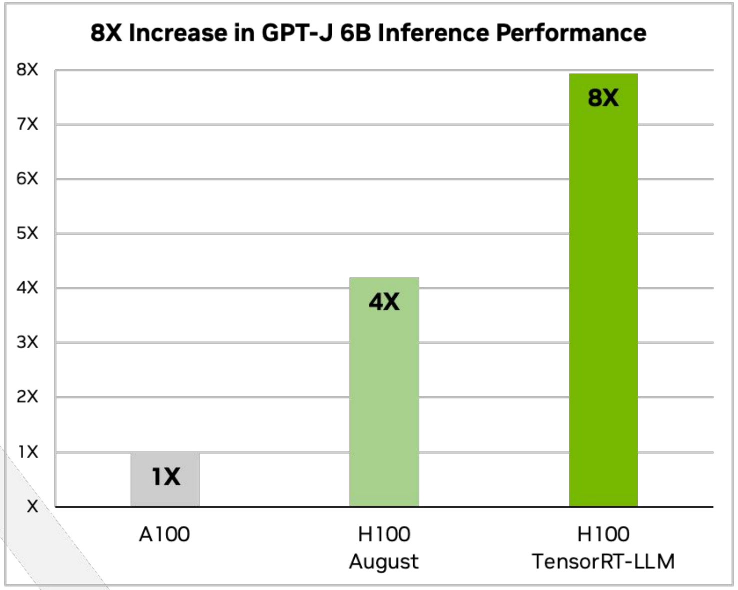
# Optimisation Techniques and features

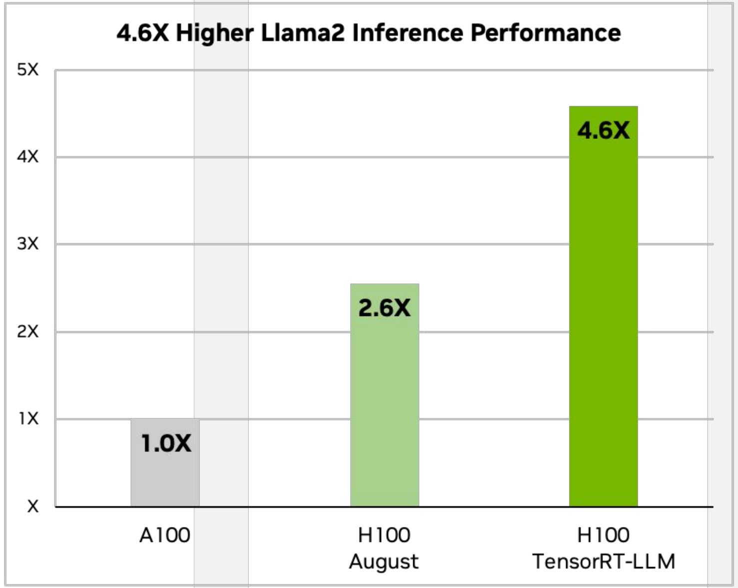
- Optimized scheduling technique called **In-flight batching**
- **Precision Optimization** - INT4/INT8 Weight-Only Quantization (W4A16 & W8A16), GPTQ Quantization, AWQ Quantization
- **Multi-GPU multi-node (**MGMN) inference
- H100 Transformer Engine with FP8
- Paged Attention - aged KV Cache for the Attention, Flash Attention,
- Tensor Parallelism, Pipeline Parallelism

NYU

# Usability, Performance and Accuracy

- **8x performance speedup on NVIDIA H100 GPU** on small language models like **GPT-J 6B leads to a 5.3x reduction in total cost of ownership (TCO) and a 5.6x reduction in energy** (electricity bill savings) over the **A100 GPU baseline**.
- On state-of-the-art LLMs like **Llama2,** even with **70B parameters, you can realize a 4.6x performance speedup**, which results in a **3x reduction in TCO and a 3.2x reduction in energy consumed** compared to the **A100 baseline.**


8X Increase in GPT-J 6B Inference Performance
8X
7X
6X
5X
4X
3X
2X
1X
X
1X
4X
8X
A100
H100
August
H100
TensorRT-LLM


4.6X Higher Llama2 Inference Performance
5X
4X
3X
2X
1X
X
1.0X
2.6X
4.6X
A100
H100
August
H100
TensorRT-LLM

# References


- HuggingFace TGI Documentation - https://huggingface.co/docs/text-generation-inference/main/en/index#text-generation-inference
- TGI toolkit - https://github.com/huggingface/text-generation-inference
- TGI Supported Models and Hardware - https://huggingface.co/docs/text-generation-inference/en/supported_models
- TensorRT-LLM Toolbox and Documentation - https://github.com/NVIDIA/TensorRT-LLM?tab=readme-ov-file#key-features
- TensorRT-LLM Performance Analysis by Nvidia - https://developer.nvidia.com/blog/nvidia-tensorrt-llm-supercharges-large-language-model-inference-on-nvidia-h100-gpus

# References


- PyTorch 2.0 - https://pytorch.org/get-started/pytorch-2.0/
- Performance Analysis - https://github.com/huggingface/blog/blob/main/optimum-onnxruntime-training.md
- Optimum Documentation - https://huggingface.co/docs/optimum/main/en/index
- ONNX Runtime - https://onnxruntime.ai/getting-started