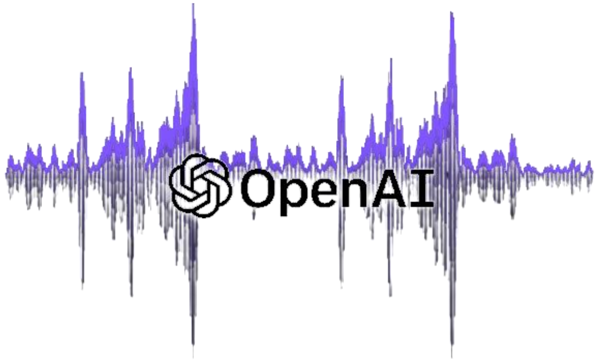
Outlines

Speach2Text
Whisper-OpenAI

NVIDIA NeMo
Framework

Quantization
TF32, FP16



NVIDIA NeMo

Framework

Speach2Text

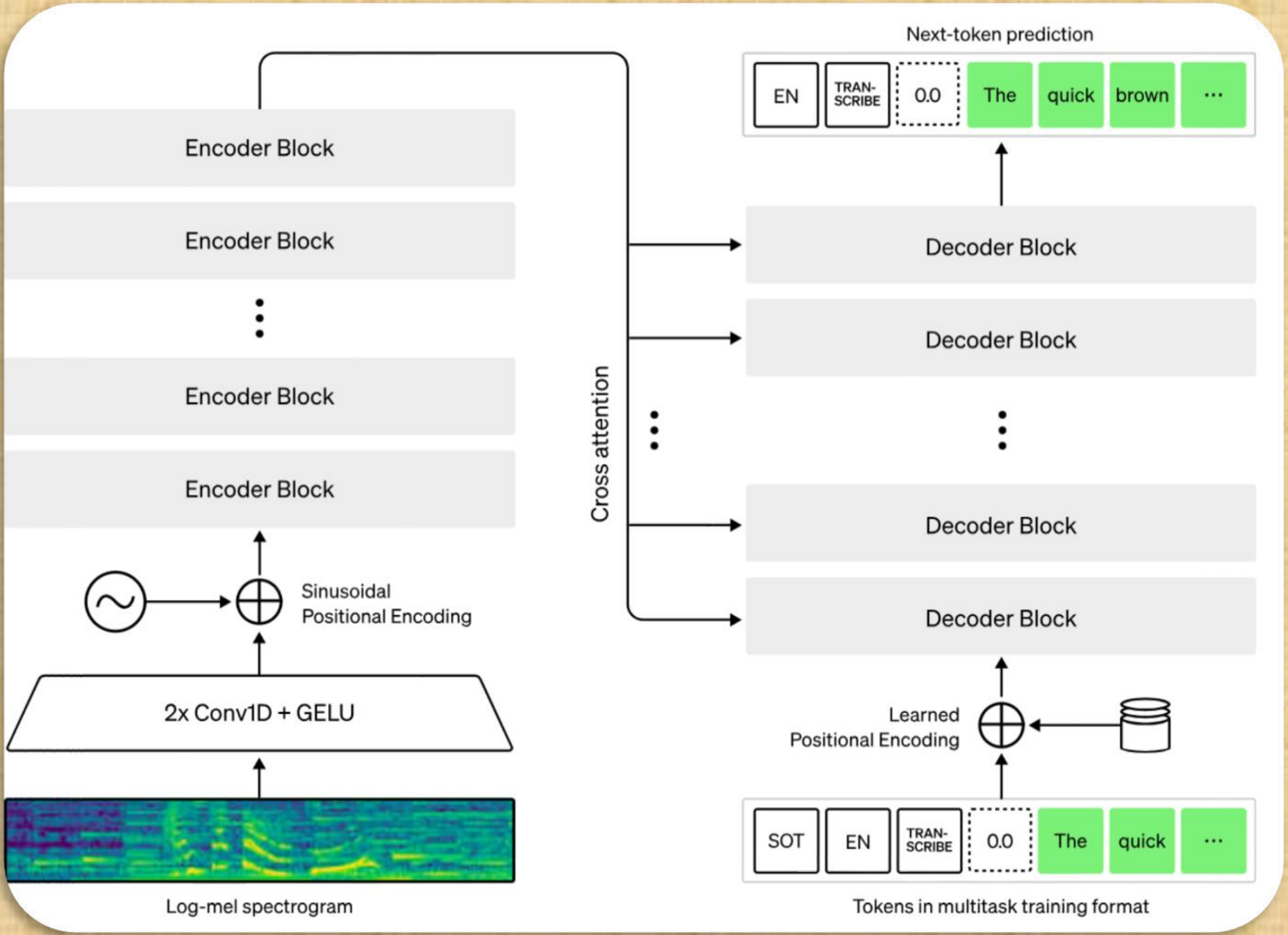
Whisper

OpenAI

Encoder

Converted into a log-Mel spectrogram

Input audio is split into 30-second chunks



Multilingual Speech Recognition
to-English Speech Translation
Language Identification
phrase-level timestamps
Voice Activity Detection

Python 3.9.9
PyTorch 1.10.1
OpenAI's tiktoken
Ex. Py 3.8-3.11

```
pip install -U openai-whisper
sudo apt update
sudo apt install ffmpeg
pip install setuptools-rust
```

NVIDIA NeMo

Framework

Speach2Text

Whisper

OpenAI

Whisper

https://openai.com/research/whisper

https://github.com/openai/whisper/tree/main

My Github with Dockerfile and example

https://github.com/HFarkhari/Whisper_OpenAI

Docker Image (Large-v2 model included)

https://hub.docker.com/r/hfarkhari/whisper_openai_speech2text

```
docker pull  hfarkhari/whisper_openai_speech2text:large-v2
docker run --gpus all -p 8888:8888 -it --rm  hfarkhari/whisper_openai_speech2text:large-v2
```

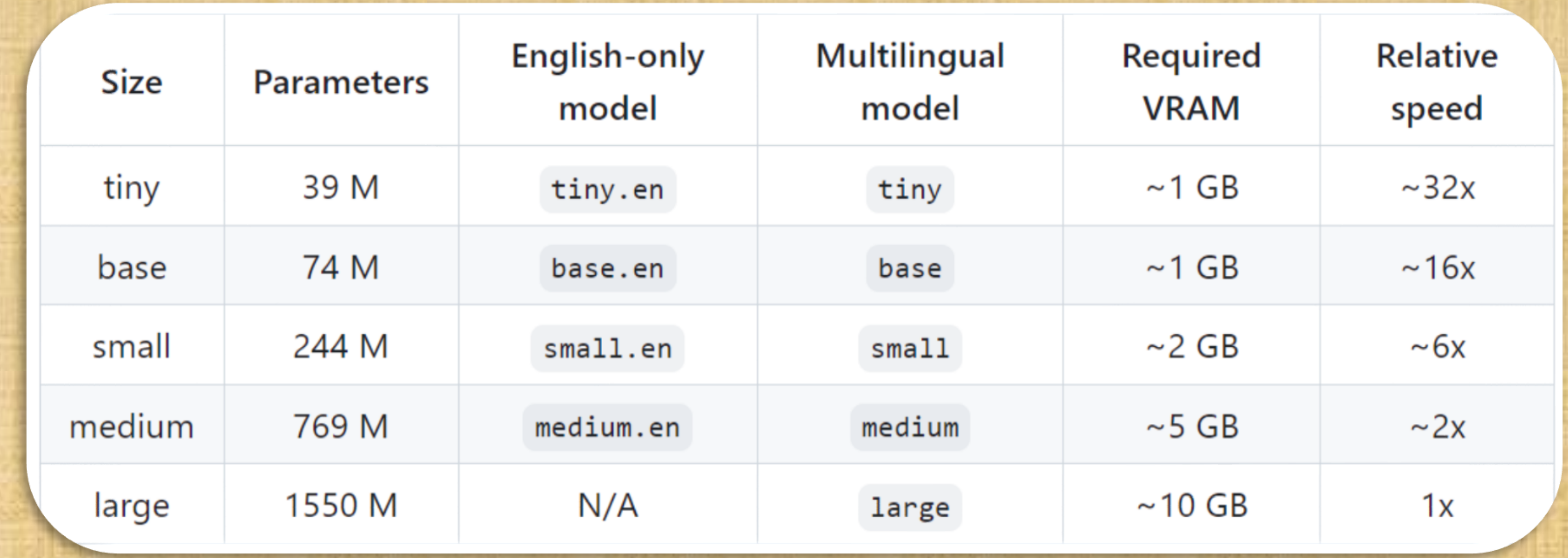
http://127.0.0.1:8888   password: 123

NVIDIA NeMo
Framework

Speach2Text
Whisper
OpenAI

https://huggingface.co/openai/whisper-large-v2

HUGGING FACE



Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~32x
base	74 M	`base.en`	`base`	~1 GB	~16x
small	244 M	`small.en`	`small`	~2 GB	~6x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x

Large V2 ~10GB VRAM (GPU)

More Info

https://www.assemblyai.com/blog/how-to-run-openais-whisper-speech-recognition-model/

Quantization
TF32, FP16

NVIDIA NeMo Framework

Speach2Text
Whisper
OpenAI

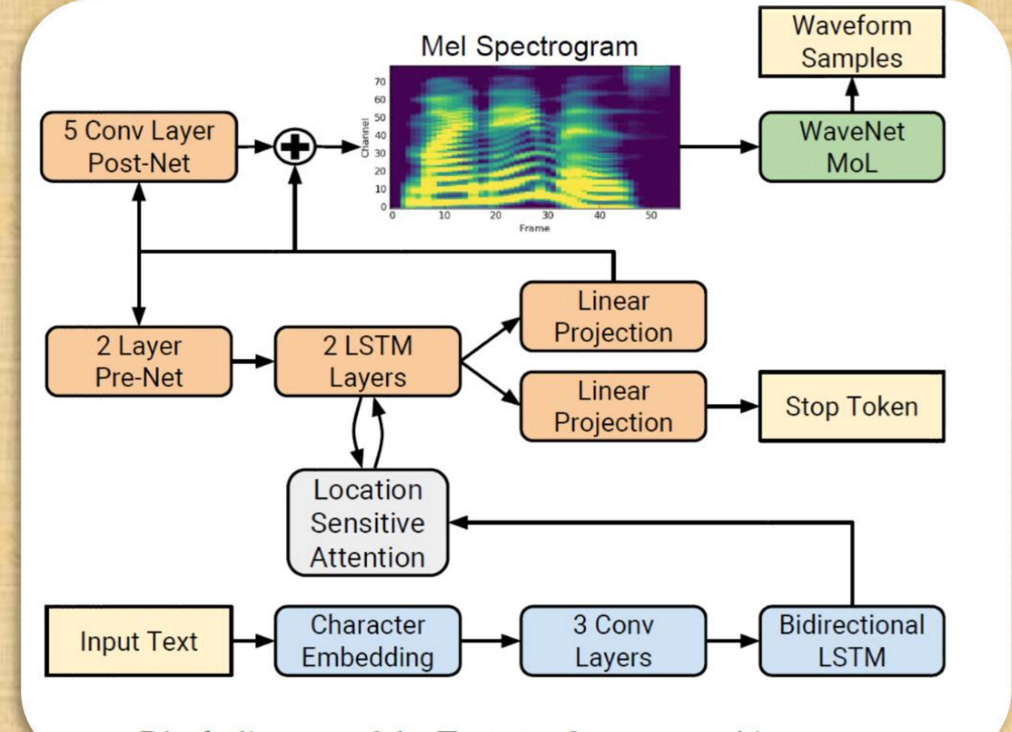
Tacotron2 - Text 2 Speech (TTS)

https://paperswithcode.com/task/text-to-speech-synthesis
https://github.com/NVIDIA/tacotron2
https://pytorch.org/hub/nvidia_deeplearningexamples_tacotron2/

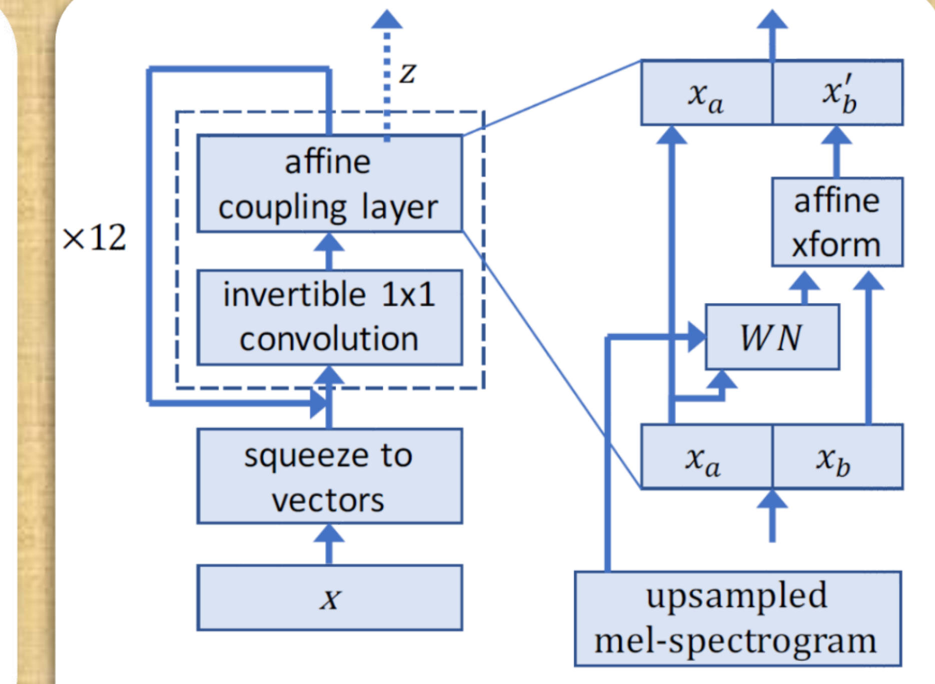
WaveGlow: a Flow-based Generative Network for Speech Synthesis

https://github.com/NVIDIA/waveglow
https://nv-adlr.github.io/WaveGlow

WAVEGLOW



Block diagram of the Tacotron 2 system architecture.



WaveGlow network

Quantization
TF32, FP16

NVIDIA NeMo Framework

Speach2Text
Whisper
OpenAI

NVIDIA NeMo Framework

https://github.com/NVIDIA/NeMo/tree/main/examples

- Automatic Speech Recognition (ASR)
- NLP (Language Modeling, Information Retrieval, Machine Translation, Question Answering, Text Classification, …)
- Speech Intent Classification and Slot Filling on SLURP Dataset
- TTS (FastPitch, Tacotron2, WaveGlow, Spectrogram Enhancer, …)

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/nemo

```
docker pull nvcr.io/nvidia/nemo:23.04
docker run  --gpus all  -it  --rm  -p 8888:8888   nvcr.io/nvidia/nemo:23.04
```

```
docker run --gpus all -it --rm --shm-size=8g -p 8888:8888 -p 6006:6006 --ulimit memlock=-1
--ulimit stack=67108864 nvcr.io/nvidia/pytorch:23.04-py3
```

Quantization
TF32, FP16

NVIDIA NeMo Framework

Speach2Text
Whisper
OpenAI

NVIDIA NeMo Framework
https://github.com/NVIDIA/NeMo/tree/main/examples

Tutorials
https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/starthere/tutorials.html

- Automatic Speech Recognition (ASR)
- NLP (Language Modeling, Information Retrieval, Machine Translation, Question Answering, Text Classification, …)
- Speech Intent Classification and Slot Filling on SLURP Dataset
- TTS (FastPitch, Tacotron2, WaveGlow, Spectrogram Enhancer, …)

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/nemo

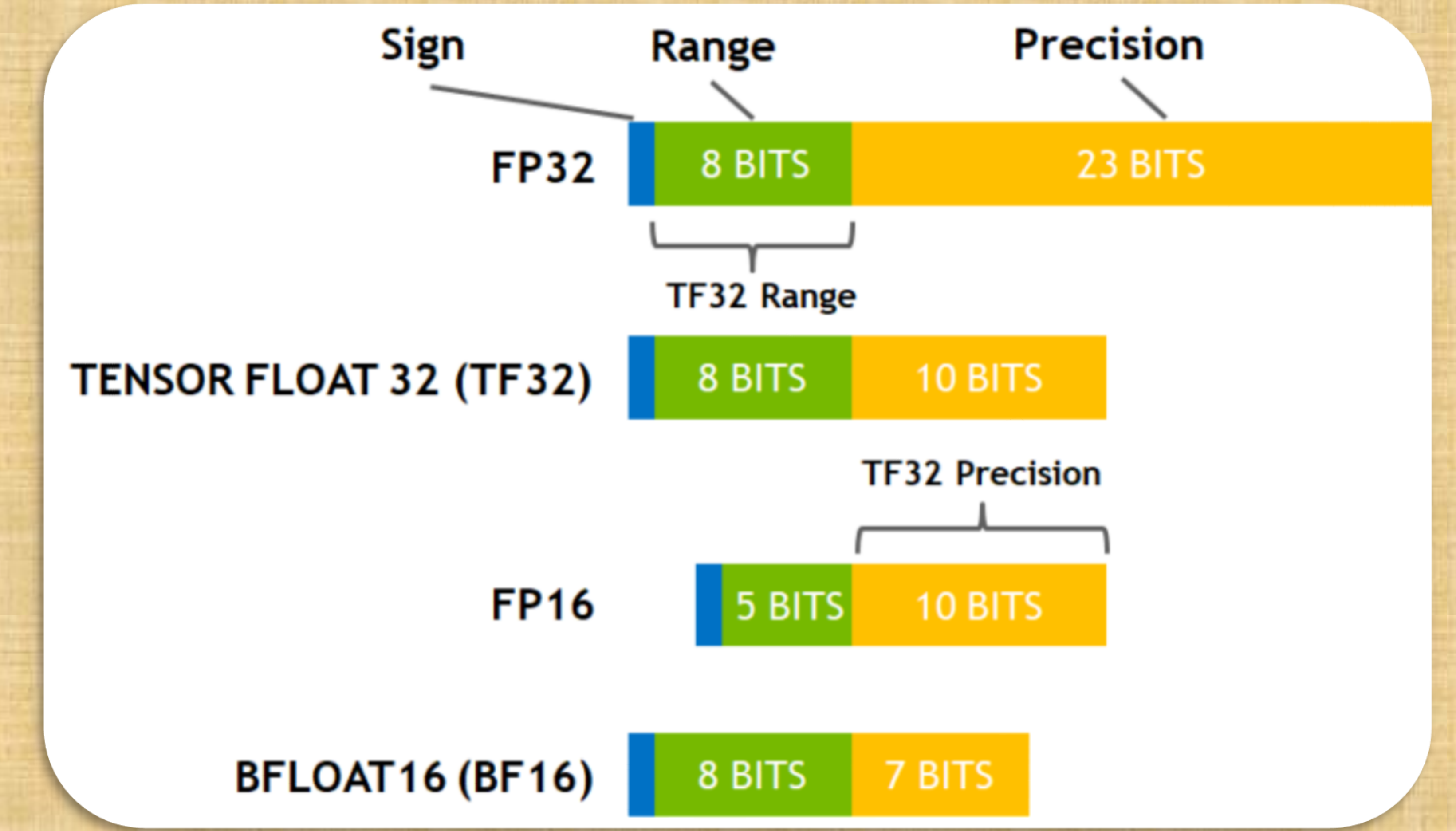
```
docker pull nvcr.io/nvidia/nemo:23.04
docker run  --gpus all  -it  --rm  -p 8888:8888   nvcr.io/nvidia/nemo:23.04

docker  run  --runtime=nvidia  -it  --rm  -v   --shm-size=16g   -p 8888:8888   -p 6006:6006
--ulimit memlock=-1  --ulimit stack=67108864   nvcr.io/nvidia/nemo:23.04
```

Quantization
TF32, FP16

NVIDIA NeMo Framework

FP32, TF32, FP16, BF16

https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/



Quantization
TF32, FP16

NVIDIA NeMo Framework

FP32, TF32, FP16, BF16

https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

TensorFloat32 (TF32) support

- Accelerating FP32 Only convolutions and matrix multiplications.
- All storage in memory and other operations remain completely in FP32, only convolutions and matrix-multiplications convert their inputs to TF32 right before multiplication.
- TF32 mode is the default option for AI training with 32-bit variables on Ampere GPU architecture (A100).
- Rounds FP32 inputs to TF32, computes the products without loss of precision, then accumulates those products into an FP32 output.
- TF32 is only exposed as a Tensor Core operation mode, not a type.
- achieves the same accuracy as FP32 training, requires no changes to hyperparameters for training scripts

FP32, TF32, FP16, BF16

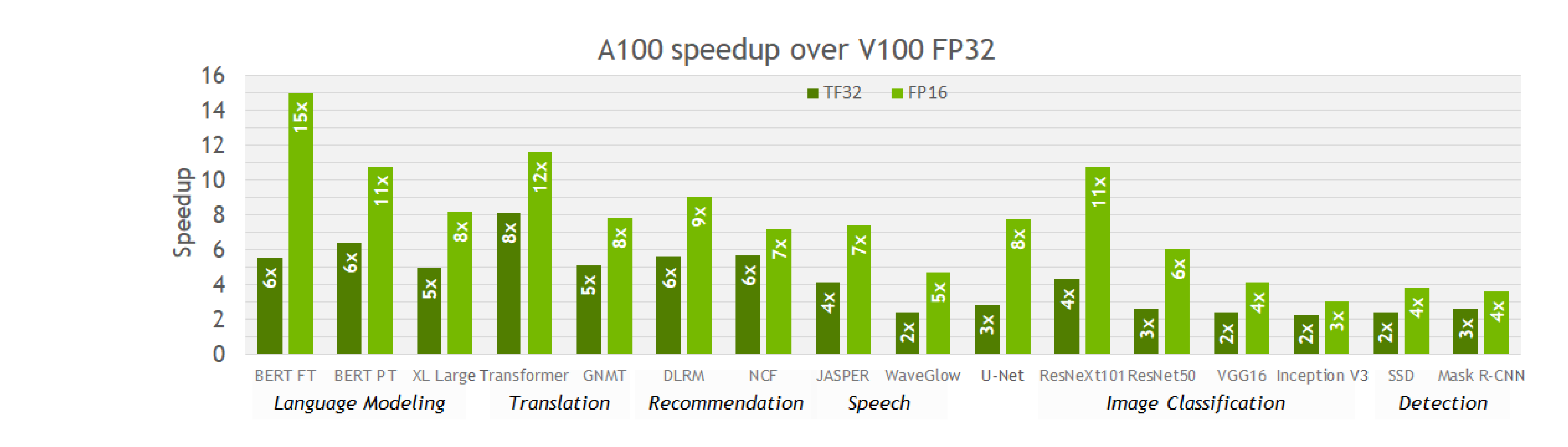
https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

Quantization
TF32, FP16

NVIDIA NeMo
Framework

TensorFloat32 (TF32) support

- FP16 gives a further speedup of up to ~2x, as 16-bit Tensor Cores are 2x faster than TF32 mode



Quantization TF32, FP16

NVIDIA NeMo Framework

TensorFloat32 (TF32) Tips

- TF32 is the default mode for AI on A100 when using the NVIDIA optimized deep learning framework containers for TensorFlow, PyTorch, and MX Net, starting with the 20.06.
- PyTorch 1.7, TensorFlow 2.4, nightly builds for MXNet 1.8 (minimum version).
- cuDNN version 8.0 and greater.
- cuBLAS version 11.0 and greater.
- cuSOLVER
- cuTENSOR version 1.1.0 and greater.
- BF16 is introduced as Tensor Core math mode in cuBLAS 11.0 and as a numerical type in CUDA 11.0.
- Deep learning frameworks and AMP will support BF16 soon.

Quantization
TF32, FP16

NVIDIA NeMo Framework

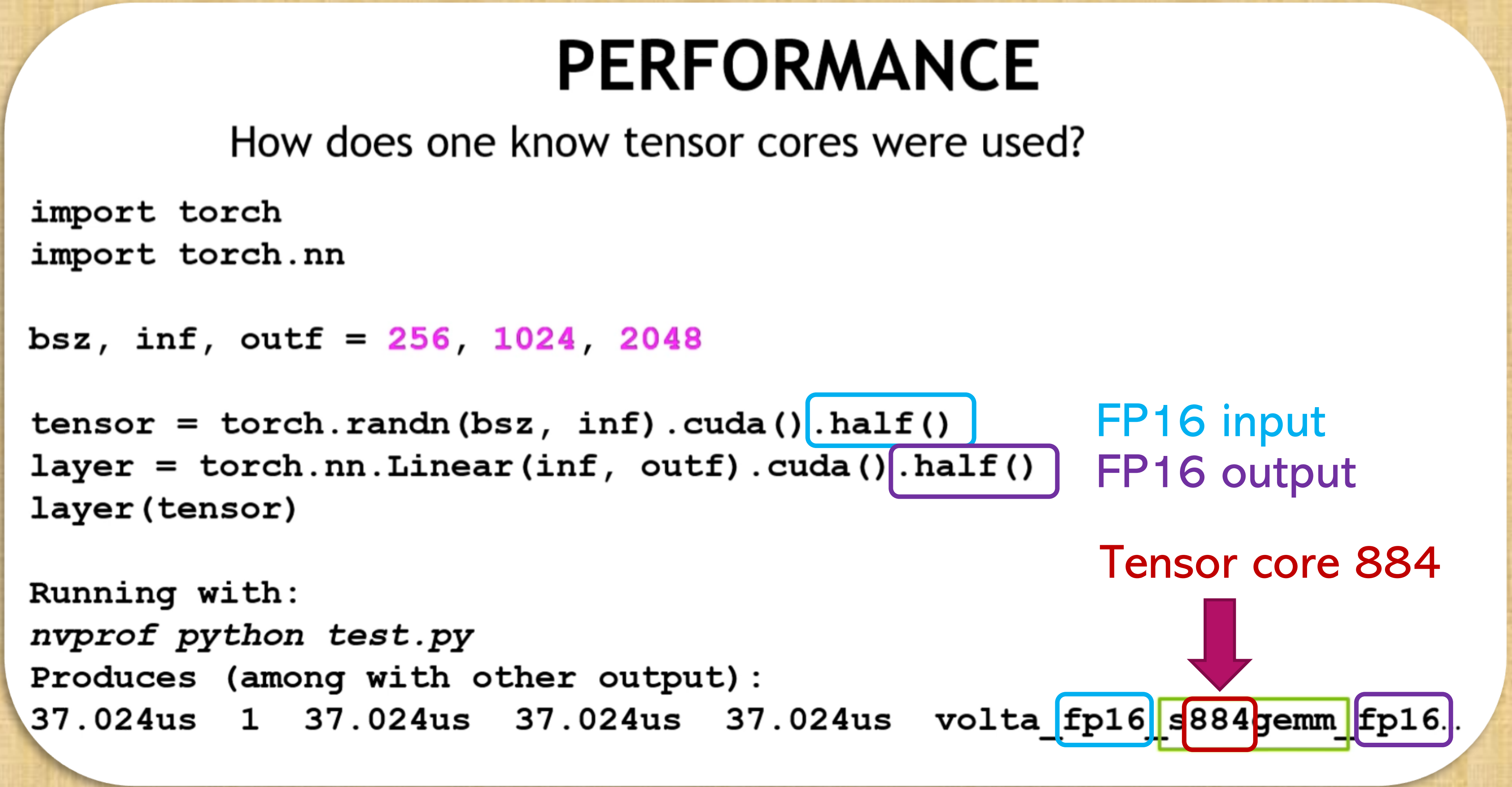
TensorFloat32 (TF32) Tips

- TF32 mode accelerates convolution and matrix-multiply layers, including linear and fully connected layers, recurrent cells, and attention blocks.
- TF32 does not accelerate layers that operate on non-FP32 tensors, such as 16-bits, FP64, or integer precisions.
- TF32 also does not apply to layers that are not convolution or matrix-multiply operations (for example, batch normalization), as well as optimizer or solver operations.
- Tensor storage is not changed when training with TF32. Everything remains in FP32, or whichever format is specified in the script.

Quantization
TF32, FP16

NVIDIA NeMo
Framework

TensorFloat32 (TF32) Tips



PERFORMANCE

How does one know tensor cores were used?

```
import torch
import torch.nn

bsz, inf, outf = 256, 1024, 2048

tensor = torch.randn(bsz, inf).cuda().half()
layer = torch.nn.Linear(inf, outf).cuda().half()
layer(tensor)

Running with:
nvprof python test.py
Produces (among with other output):
37.024us   1   37.024us   37.024us   37.024us   volta_fp16_s884gemm_fp16..
```

FP16 input

FP16 output

Tensor core 884

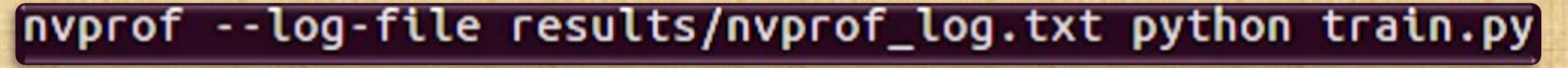
https://youtu.be/tAIakfEt-tI

Quantization

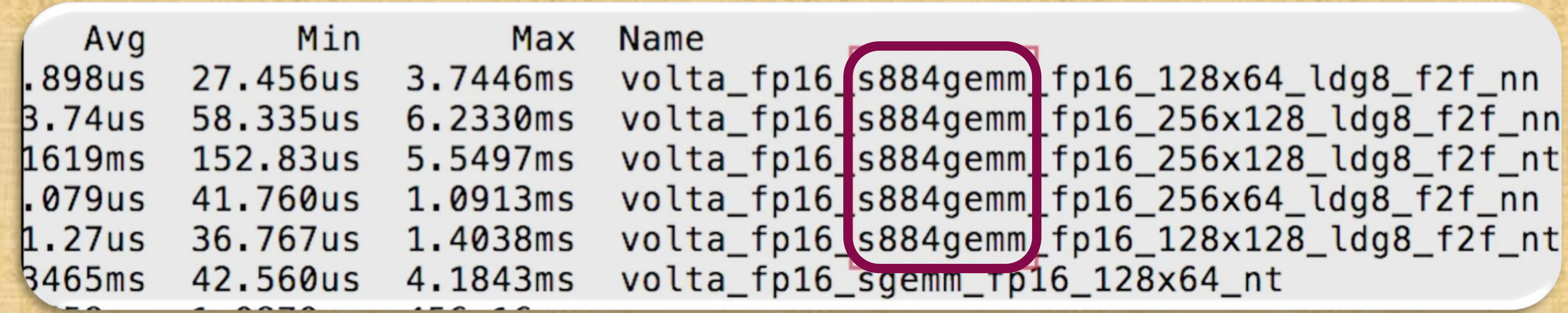
TF32, FP16

NVIDIA NeMo Framework

TensorFloat32 (TF32) Tips



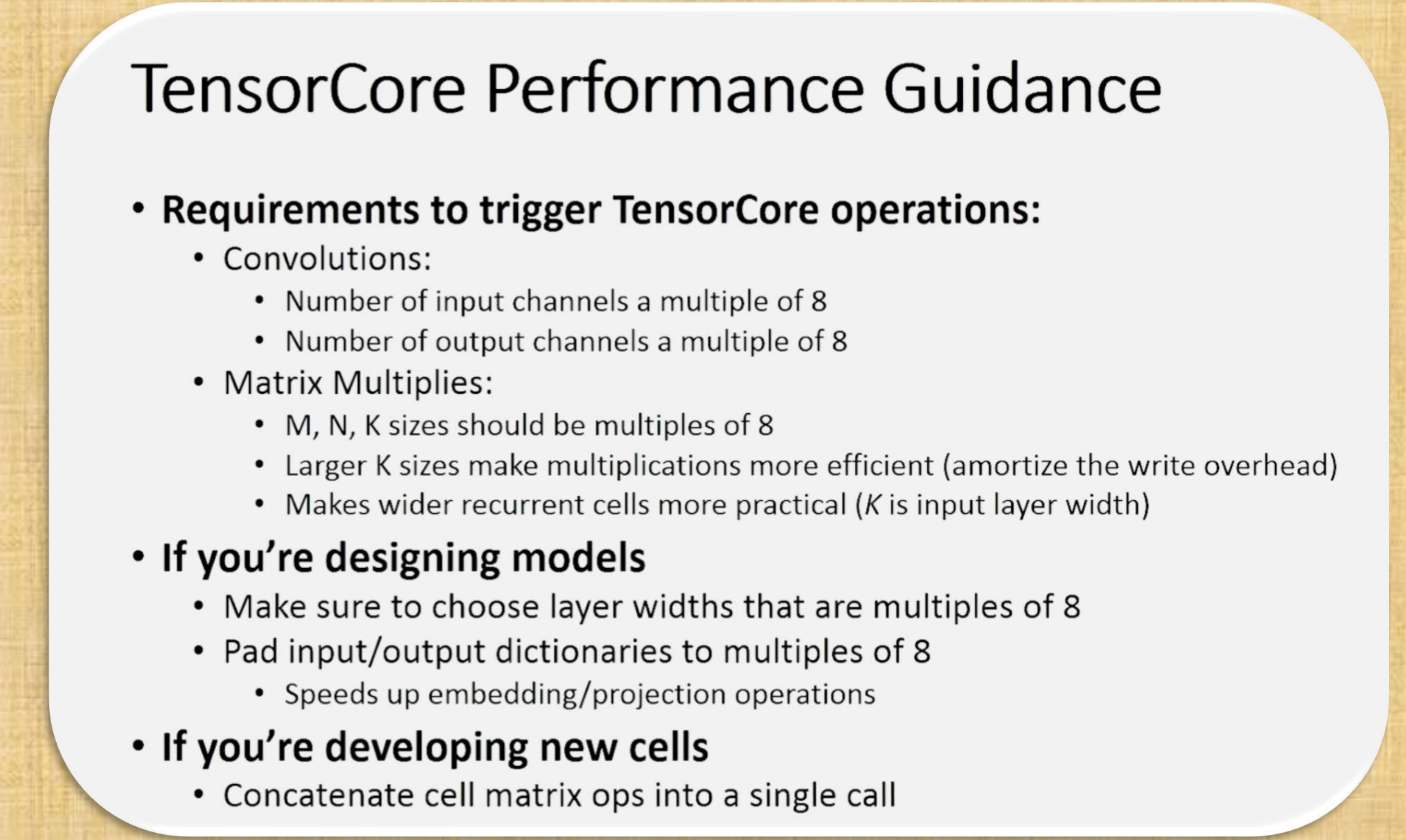
```
nvprof --log-file results/nvprof_log.txt python train.py
```



```
    Avg          Min          Max    Name
.898us    27.456us    3.7446ms    volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
3.74us    58.335us    6.2330ms    volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
1619ms    152.83us    5.5497ms    volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt
.079us    41.760us    1.0913ms    volta_fp16_s884gemm_fp16_256x64_ldg8_f2f_nn
1.27us    36.767us    1.4038ms    volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nt
3465ms    42.560us    4.1843ms    volta_fp16_sgemm_fp16_128x64_nt
```

Quantization
TF32, FP16

NVIDIA NeMo Framework



TensorCore Performance Guidance

- **Requirements to trigger TensorCore operations:**
  - Convolutions:
    - Number of input channels a multiple of 8
    - Number of output channels a multiple of 8
  - Matrix Multiplies:
    - M, N, K sizes should be multiples of 8
    - Larger K sizes make multiplications more efficient (amortize the write overhead)
    - Makes wider recurrent cells more practical (*K* is input layer width)
- **If you're designing models**
  - Make sure to choose layer widths that are multiples of 8
  - Pad input/output dictionaries to multiples of 8
    - Speeds up embedding/projection operations
- **If you're developing new cells**
  - Concatenate cell matrix ops into a single call

https://youtu.be/i8-Jw48Cp8w

Quantization
TF32, FP16

NVIDIA NeMo
Framework

Download Docker Containers Framework compatible with TF32

Download Latest TensorFlow

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow

Download Latest PyTorch

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch

Thanks

Any Question ?