Generative AI Powered by Intel

Akash Dhamasia, AI Software Solutions Engineer

March 07, 2024

intel®

Agenda

- Generative AI Introduction
- Intel SW for Generative AI
  - AI frameworks (PyTorch, TensorFlow)
  - Recipe for Intel® Optimizations with IPEX
  - Profiling techniques
  - Distributed Training
    - Intel Extension for Transformers and DeepSpeed
- Demo
- Conclusion

Let's play a little game: which image is real?

ALL images fully generated by Stable Diffusion SDXL

AI has made incredible progress in the last years

Typical Domains of AI (for the last 10 years)

COMPUTER VISION

- Ability to understand the visual world

Classification

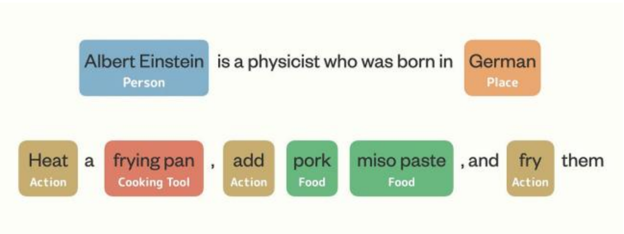
Object Detection

Instance or Semantic segmentation

NATURAL LANGUAGE PROCESSING (NLP)

- Ability to understand the written world

Translation



Entity Name Recognition

Generative AI

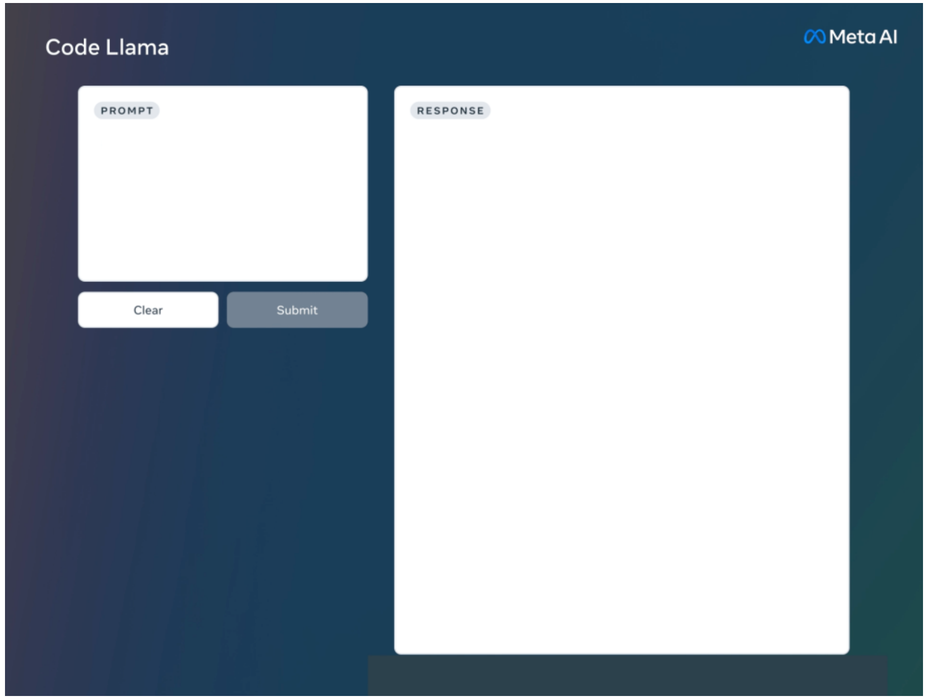
- End of 2022, ChatGPT was released, and the generative AI (genAI) craze started!
- Generative AI is the ability for the AI model to create contents (text, image , music, code ...)

Image fully generated by Stable Diffusion SDXL, a text-to-image AI

The new trend – generative AI

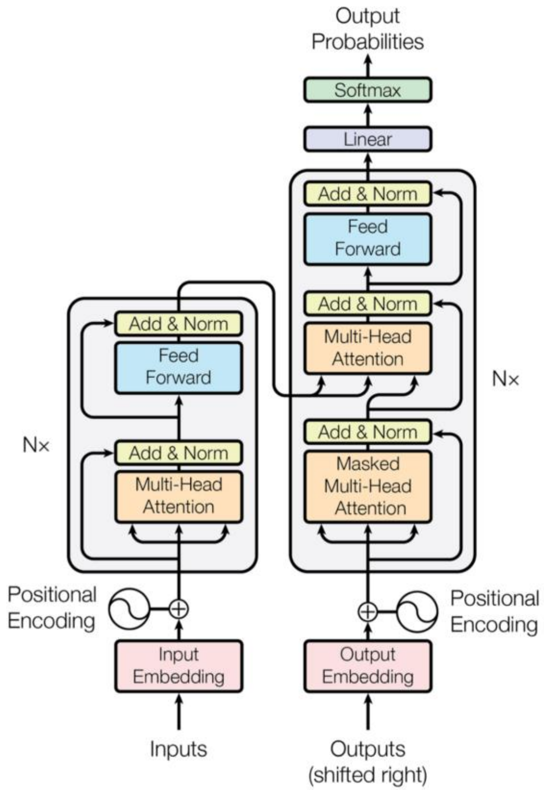
Art generation

Code generation



Transformers

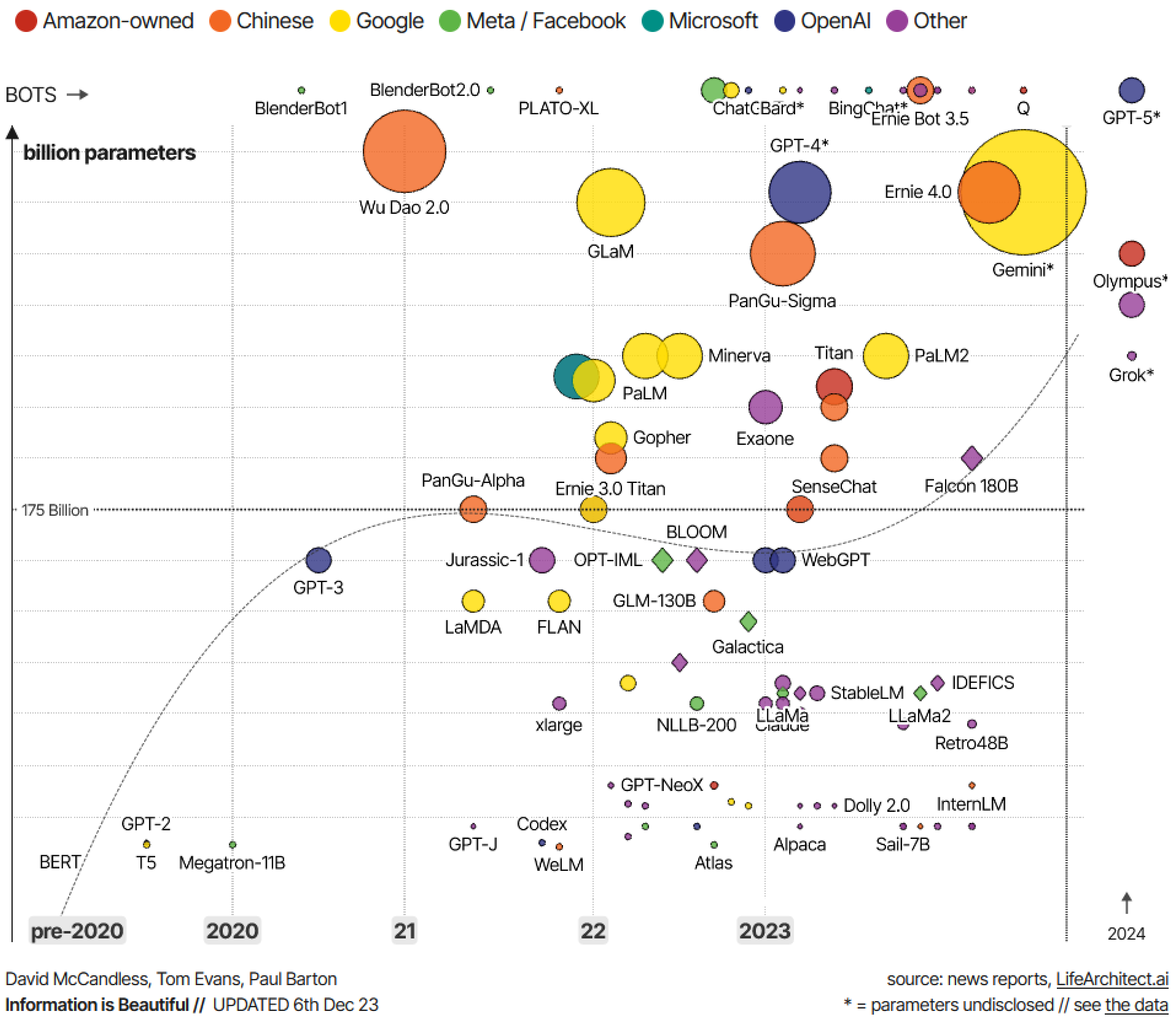
- Transformer architecture is the base for NLP and genAI (e.g., BERT, LLM, …)
  - Composed of 2 building blocks: encoder and decoder



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

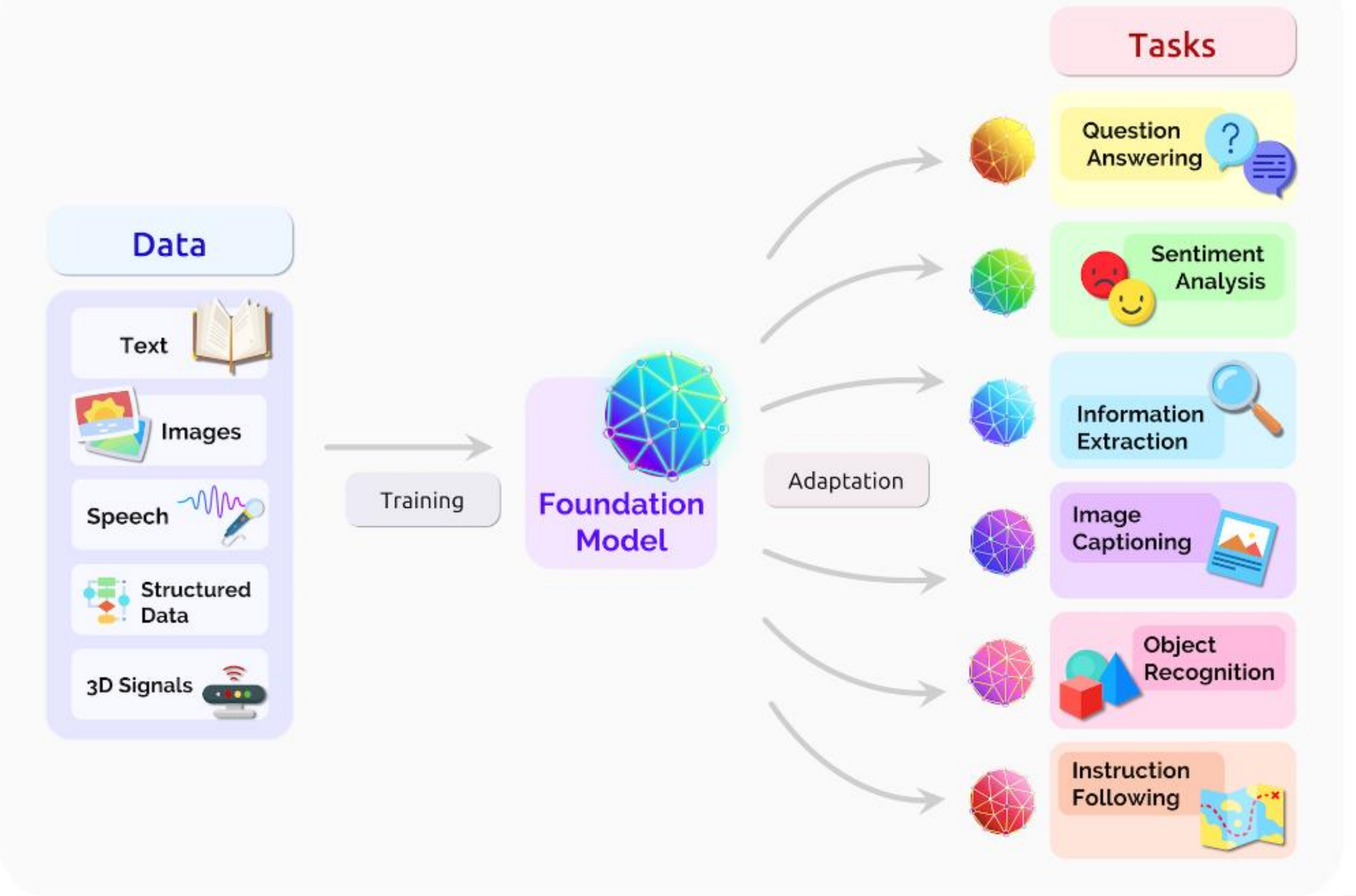
Model Size, way up

- Model keep growing in size
- Trained in self-supervised manner on web-scale quantities of unlabeled data
- Starting with Bert was 0.3B in 2018
- This is logarithmic scale!



Foundational Model

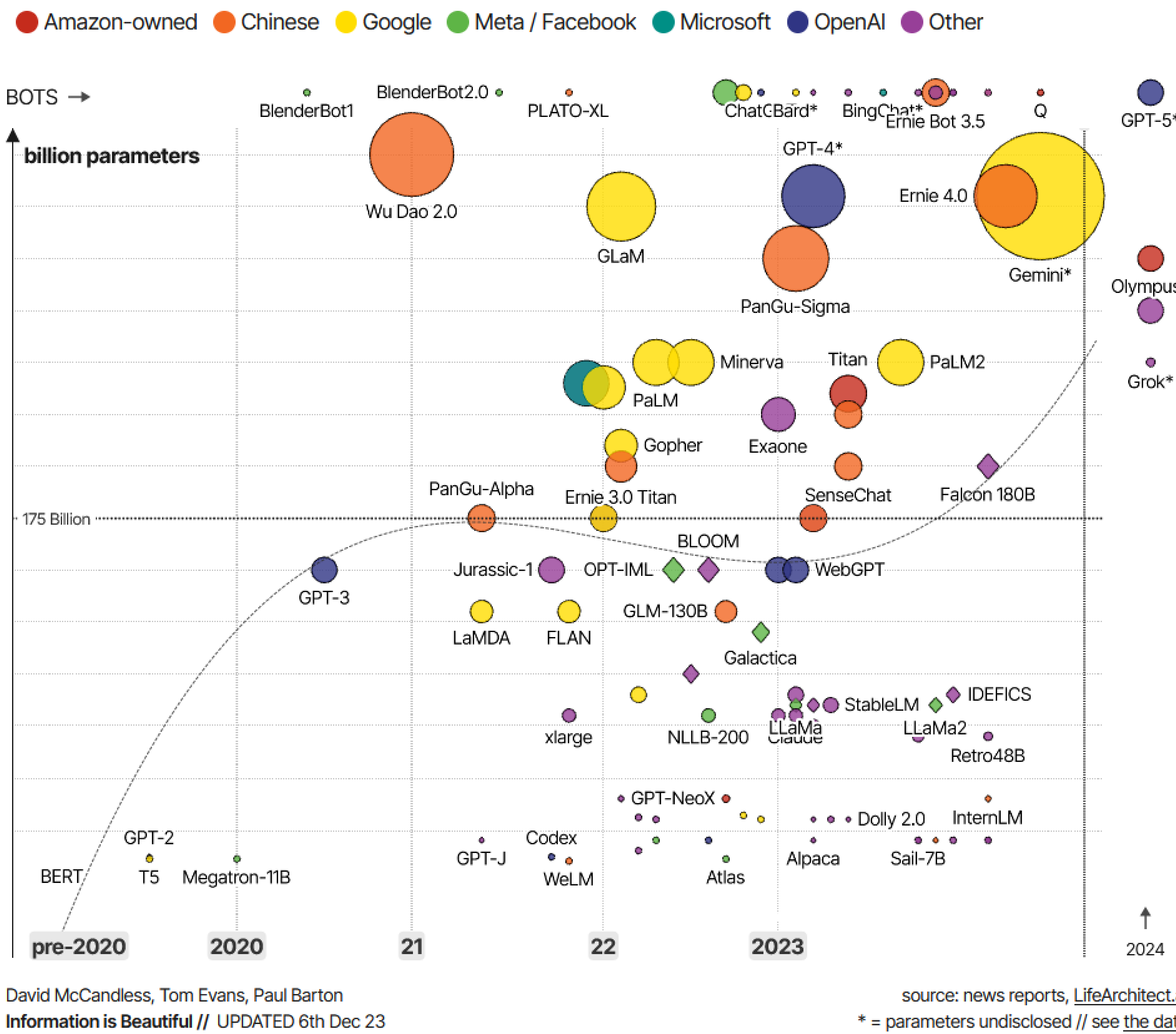
- From the first iterations of Large Language Model such as GPT, model started to be large enough to abstract concepts and language
- They are coined as Foundational Models
- This ONE model can then be adapted to a wide range of downstream tasks



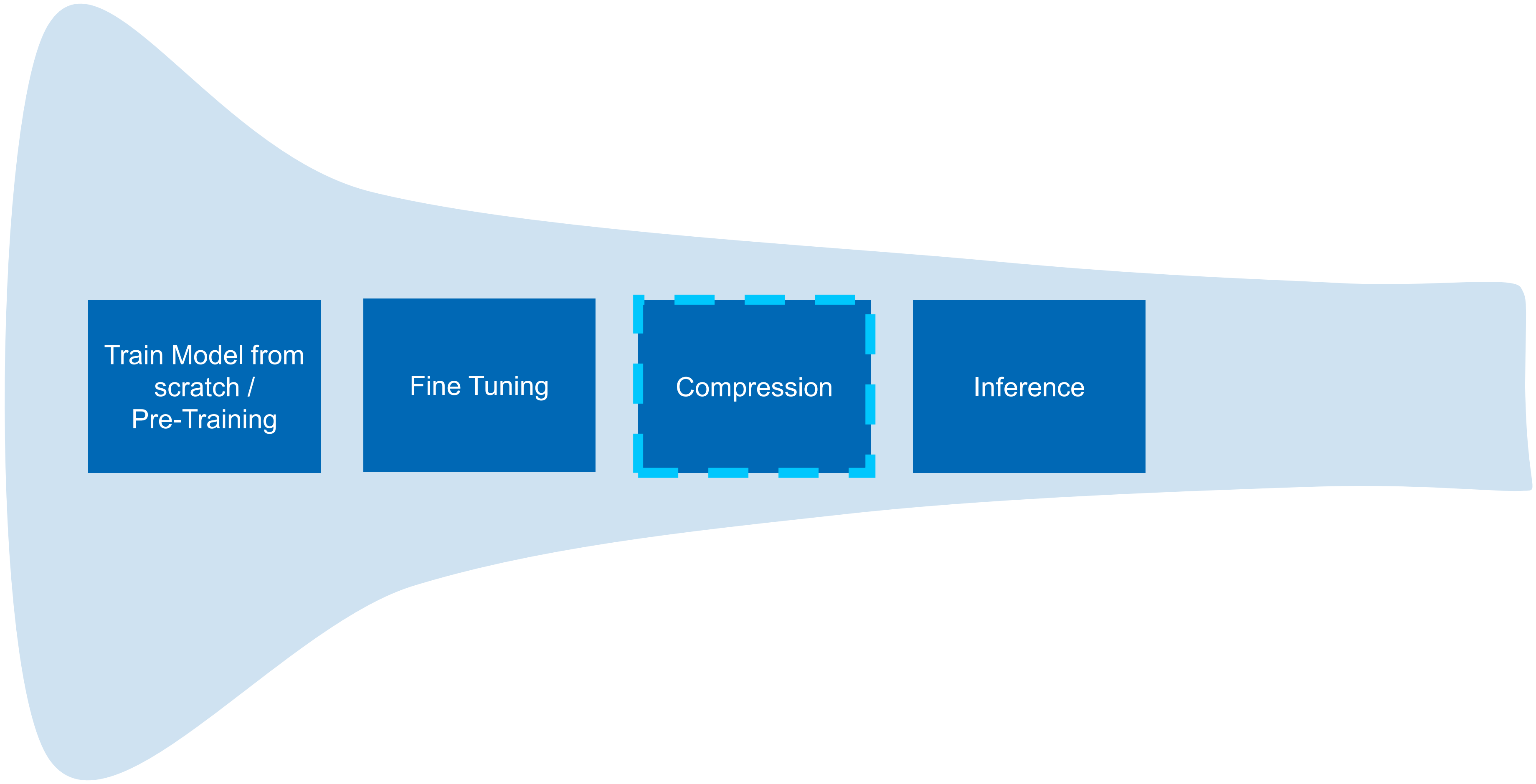
Bommasani, Rishi, et al. "On the opportunities and risks of foundation models." *arXiv preprint arXiv:2108.07258* (2021).

Recent trend is to scale down

- Push from the open-source community
- Models are trained on better quality (and smaller) dataset
- Trained on “smaller” infrastructures
- Mixture of Experts are also the trend



Deep Learning Funnel Pipeline



Intel Generative AI Products

intel. DATA CENTER GPU MAX SERIES

LFM pre-training and batch inferencing

<20b parameters (training)

Fine tune on Gaudi 2

intel. DATA CENTER GPU MAX SERIES

Fine tune on GPU Max

<10b parameters (fine tuning in hours)

Fine tune on single or dual socket workstation (4-8TB) and Servers (8TB per node, multi-node)

DL inference on Gaudi 2

intel. DATA CENTER GPU MAX SERIES

Fine tune on GPU Max

intel. xeon intel. core intel. ARC GRAPHICS

Inference on Xeon, Core, ARC

Entry Points to the Intel AI Platform

Bring your own data

Bring your own Model

Select Inference Model

Bring your prompts

API

Models, and Services

Ecosystem Services, Hubs & Framework

Intel Developer Catalogue & AI Reference KIts

Customer Workflows

Build efficient, SOTA production models

Pre-train models → Fine-Tune Models → Optimize Inference Models

intel. Tools and Frameworks

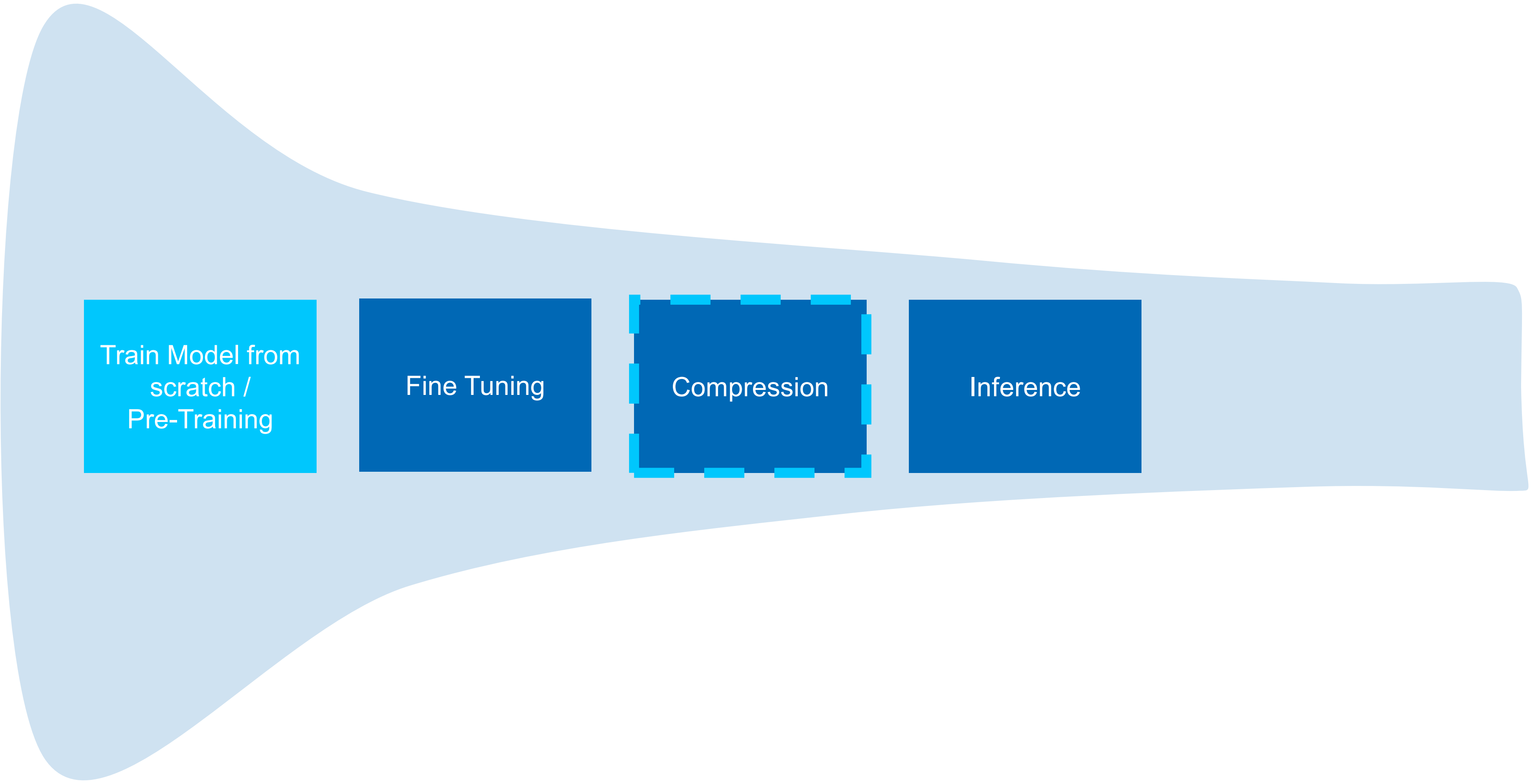
Up-streamed optimizations

- Intel® Extension for Transformers (ITREX1.1)
- Intel® Extension for PyTorch (IPEX)
- Intel® Extension for DeepSpeed (IDEX)
- Intel® Neural Compressor
- Retrieval Framework: fastRAG, haystack by deepset

E2E AI FW and Libs.

TensorFlow | PyTorch | **Intel AI Software Distribution** | ONNX RUNTIME | OpenVINO

Deep Learning Funnel Pipeline



Pre-training

- Training in self-supervised manner on web-scale quantities of unlabeled data
- Common foundational models are trained on web-scale corpus of text extracted from the Internet

Infrastructure to Train those Foundational Models

- Only possible through supercomputers

European example:

- BLOOM (176B) was trained on Jean-Zay supercomputer (CNRS, France)
  - 384 80GB A100 GPUs for 3.5 months
  - On 366B tokens (1.6TB of pre-processed text)

Pretraining for domain adaptation

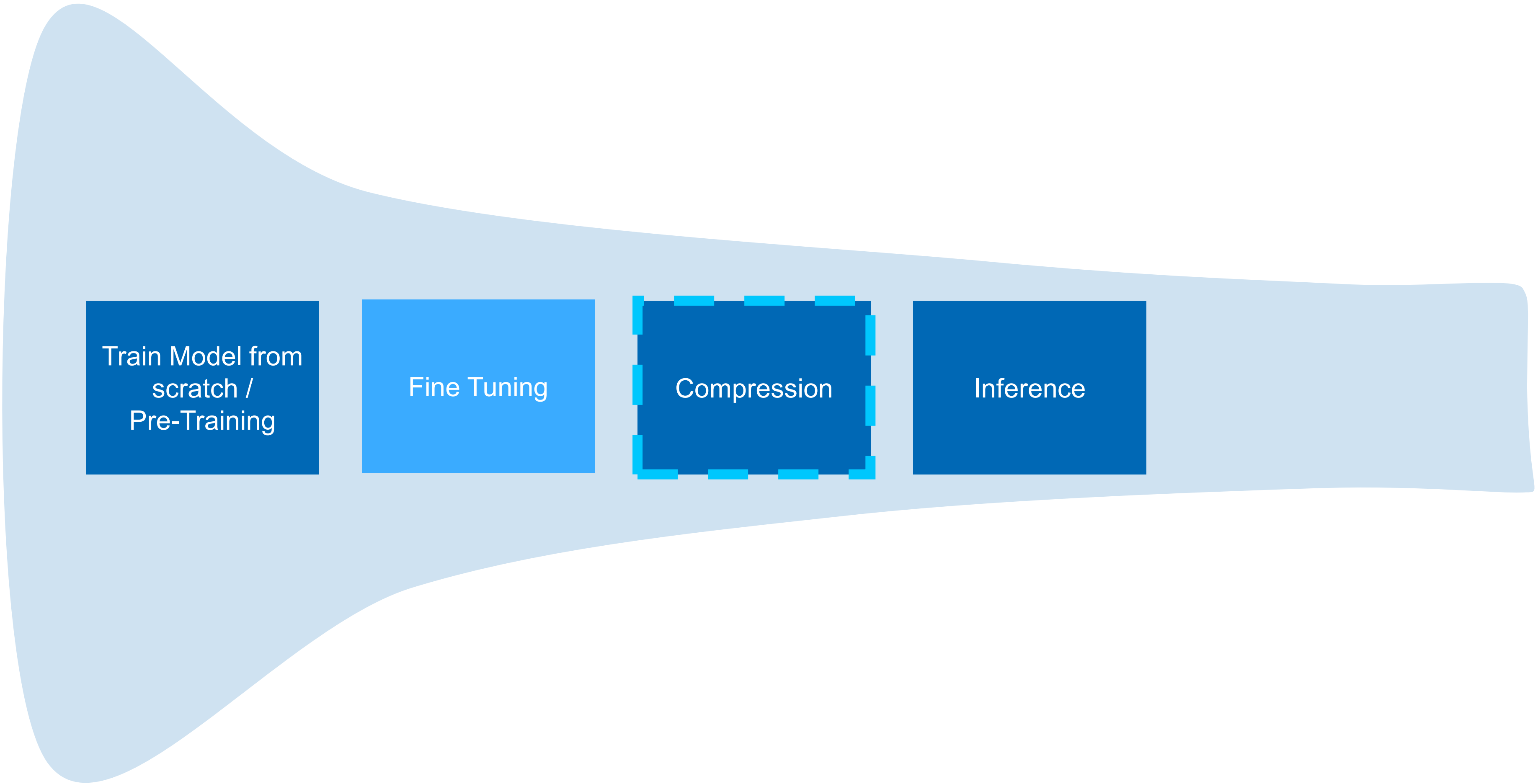
- Specialized vocabulary:
  - Domain with specialized vocabulary: medical, legal, finance
  - E.g. BloombergGPT: trained for financial data. (51% financial data / 49% general data)
- Applied on non-text data - unstructured sequential data
  - Genomics
  - Protein generation
  - Generative chemistry
- Where few or no foundational model can be found for those applications yet

Intel tools necessary for it

Intel Extension for PyTorch

Intel Extension for Deepspeed

Deep Learning Funnel Pipeline



“Fine tuning is the new training “

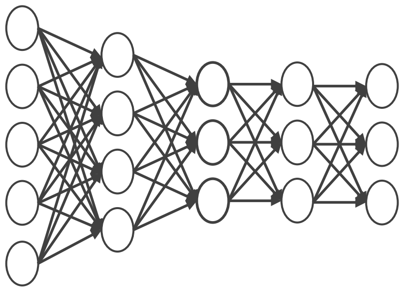
Fine-tuning

- Adapt the pre-trained model to a specific task or domain using a smaller, task-specific dataset.
- Makes the model more specialized and improves its performance on that particular task.

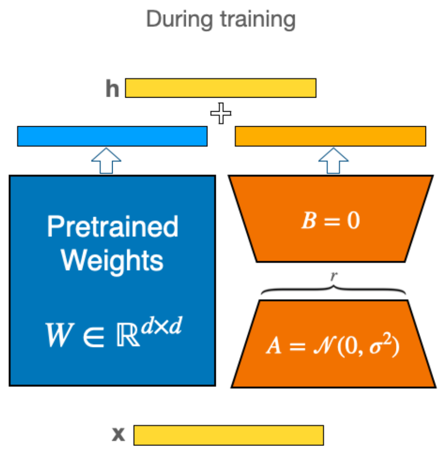
Fine-tuning techniques

Supervised fine-tuning

Small dataset with annotation



Parameter-Efficient Fine-Tuning.



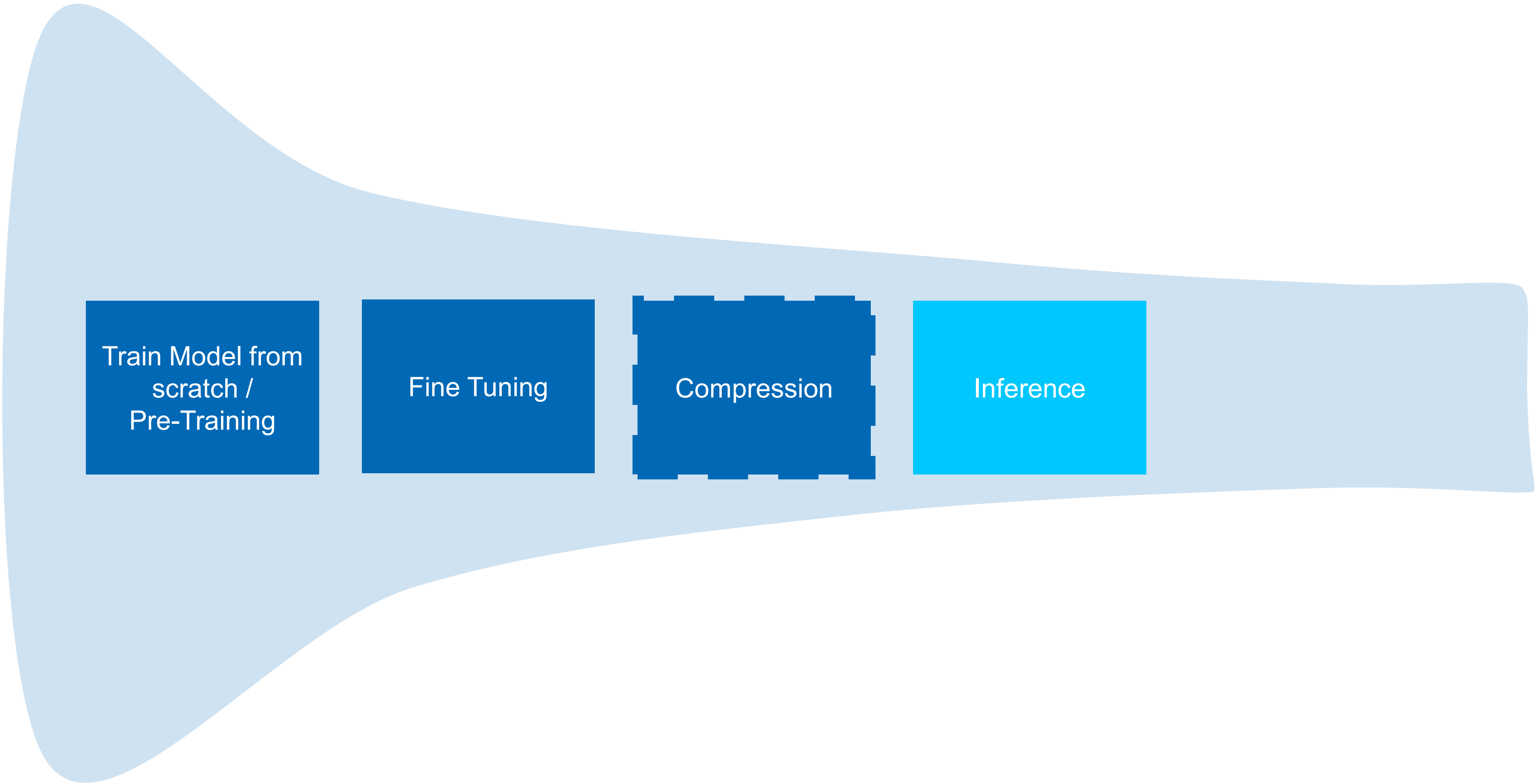
Intel Extension for PyTorch

Intel Extension for Deepspeed

Intel Extension for PyTorch

Intel Extension for Transformers

Deep Learning Funnel Pipeline



Intel tools necessary for it

Intel Extension for PyTorch

Intel Extension for Deepspeed

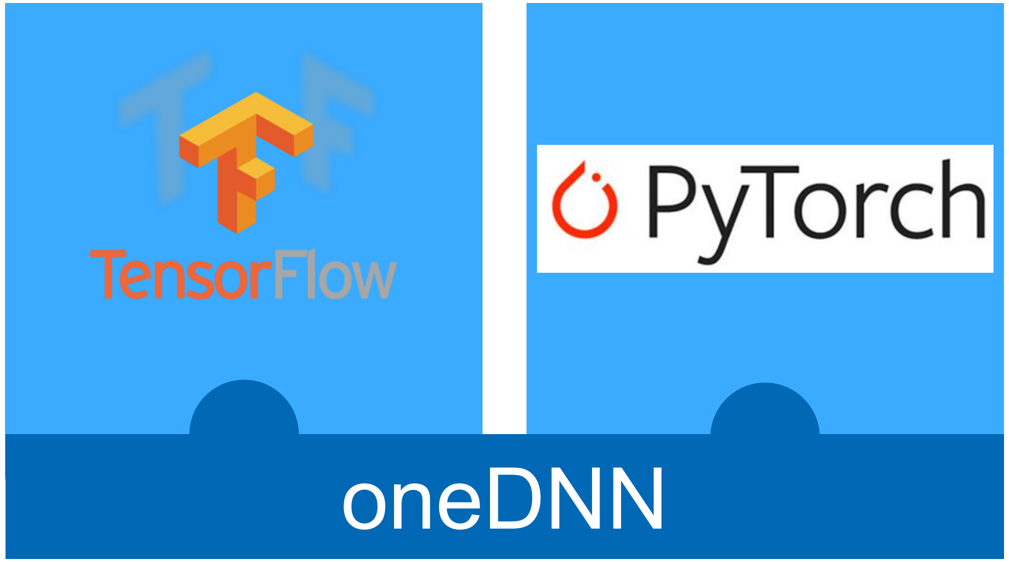
Intel Extension for Tensorflow

Essential building blocks – AI frameworks

Intel®-Optimized Deep Learning Frameworks – Introduction

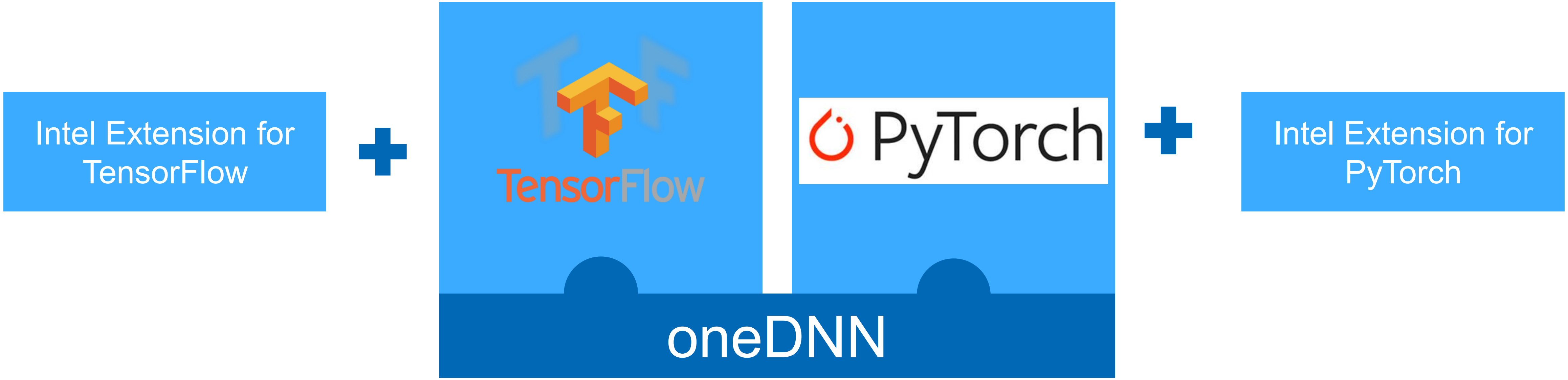
Intel®-Optimized Deep Learning Frameworks

- Intel®-optimized DL frameworks are drop-in replacement,
  - **No front code change for the user**
- Optimizations are up-streamed automatically (TF) or on a regular basis (PyTorch) to stock frameworks

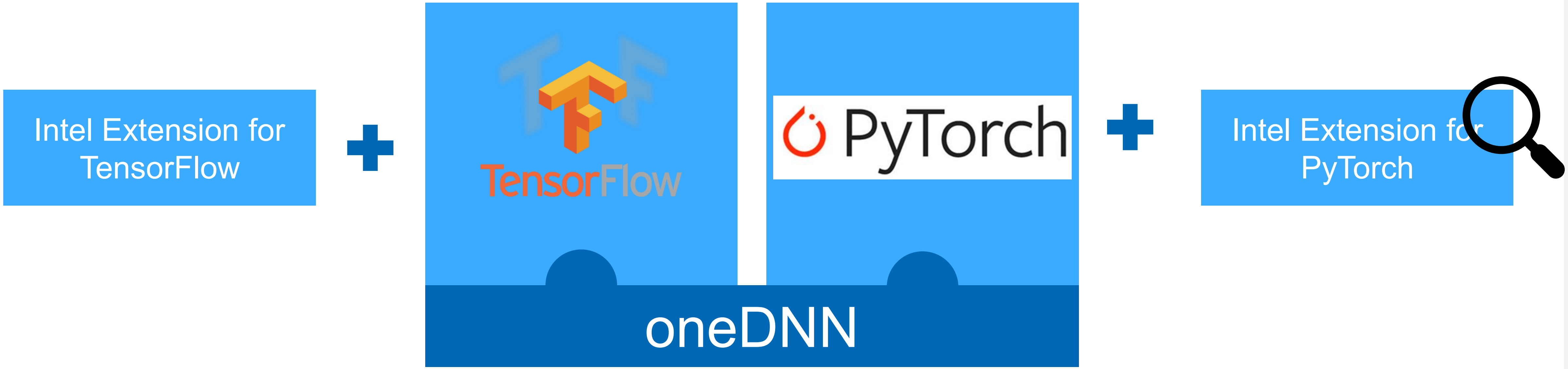


Intel®-Optimized Deep Learning Frameworks

- Intel® Extension for PyTorch and TensorFlow are additional modules for functions not supported in standard frameworks (such as mixed precision and dGPU support).
- As they offer more aggressive optimizations, they offer bigger speed-ups for training and inference.



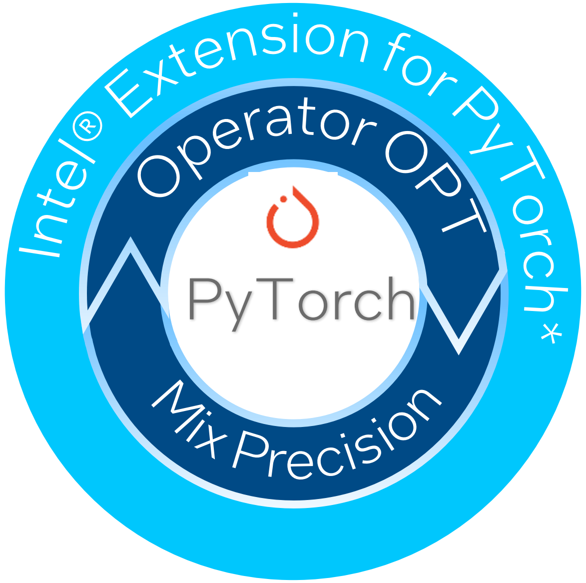
Intel®-Optimized Deep Learning Frameworks



Intel® Extension for PyTorch

Intel® Extension for PyTorch* (IPEX)

- Buffer the PRs for stock PyTorch
- Provide users with the up-to-date Intel software/hardware features
- Streamline the work to integrate oneDNN
- Unify user experiences on Intel CPU and GPU



Intel® Optimization for PyTorch

ECOSYSTEM

torchvision | TorchServe | Hugging Face | PyTorch Lightning | …

FRAMEWORKS

PyTorch | Intel® Extension for PyTorch*

LIBRARIES

oneAPI oneDNN | oneAPI oneCCL

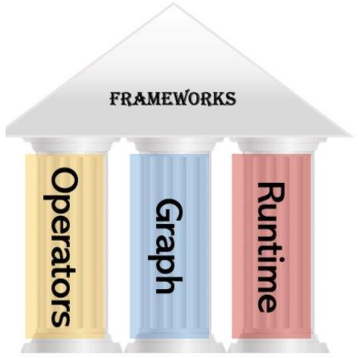
33

intel XEON inside

Other names and brands may be claimed as the property of others

Major Optimization Methodologies

3-Pillar Framework Optimization Techniques



Op

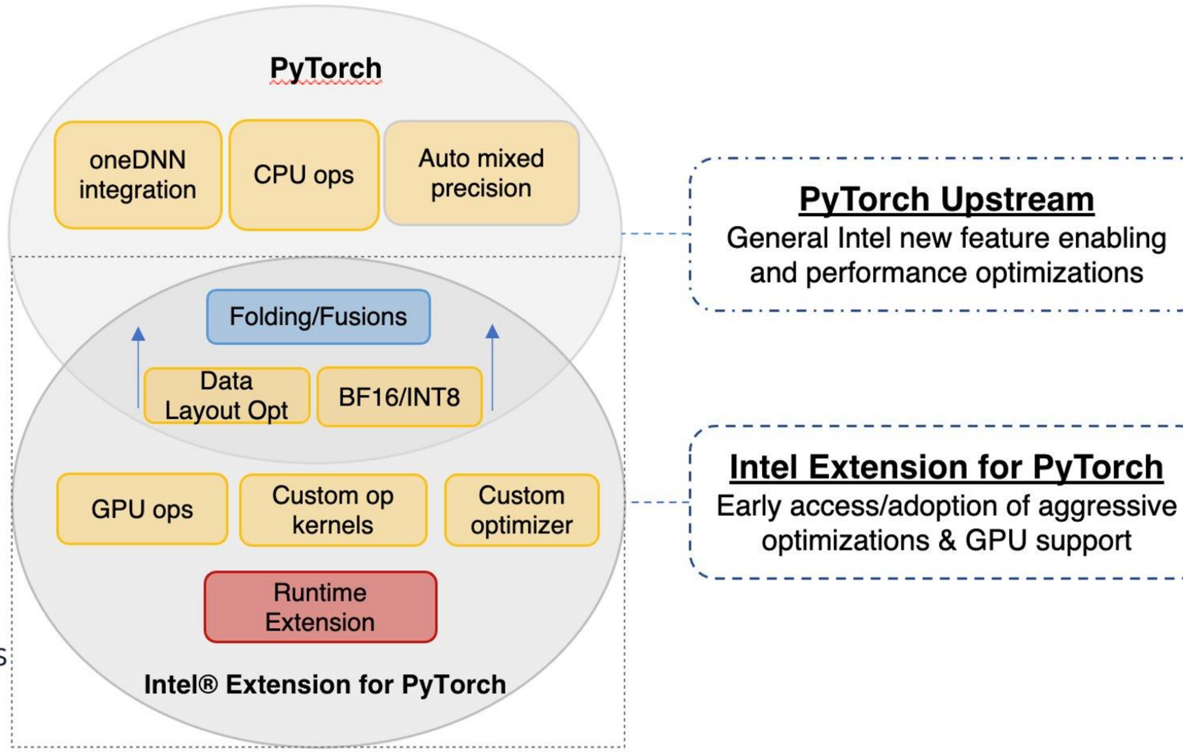
- Vectorization and Multi-threading
- Low-precision BF16/INT8 compute
- Ease-of-use BF16 compute with Auto-Mixed-Precision (AMP)
- Data layout optimization for better cache locality

Graph

- Constant folding to reduce compute
- Op fusion for better cache locality

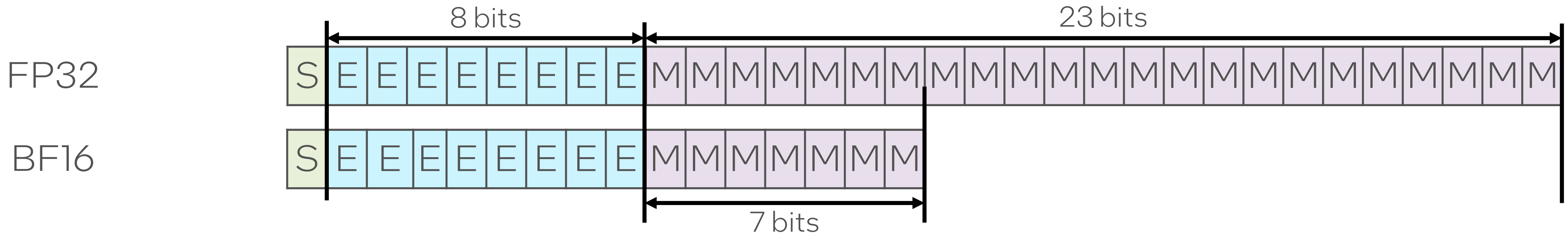
Runtime

- Thread affinitization and multi-streams
- Memory buffer pooling
- GPU runtime
- Launcher



Building and Deploying with BF16

Low-precision Optimization – BF16



BF16 has the same range as FP32 but less precision due to 16 less mantissa bits.
Running with 16 bits can give significant performance speedup.

https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf

Inference with Intel® Extension for PyTorch
Usage Example

*The .to("xpu") is needed for GPU only
**Use torch.cpu.amp.autocast() for CPU
***Channels last format is automatic

Resnet50

```python
import torch
import torchvision.models as models
############# code changes ################
import intel_extension_for_pytorch as ipex
############# code changes ################

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

#################### code changes #################
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)
#################### code changes #################

with torch.no_grad():
    d = torch.rand(1, 3, 224, 224)
    ############################# code changes #####################
    d = d.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
    ############################# code changes #####################
        model = torch.jit.trace(model, d)
        model = torch.jit.freeze(model)
        model(data)
```

BERT

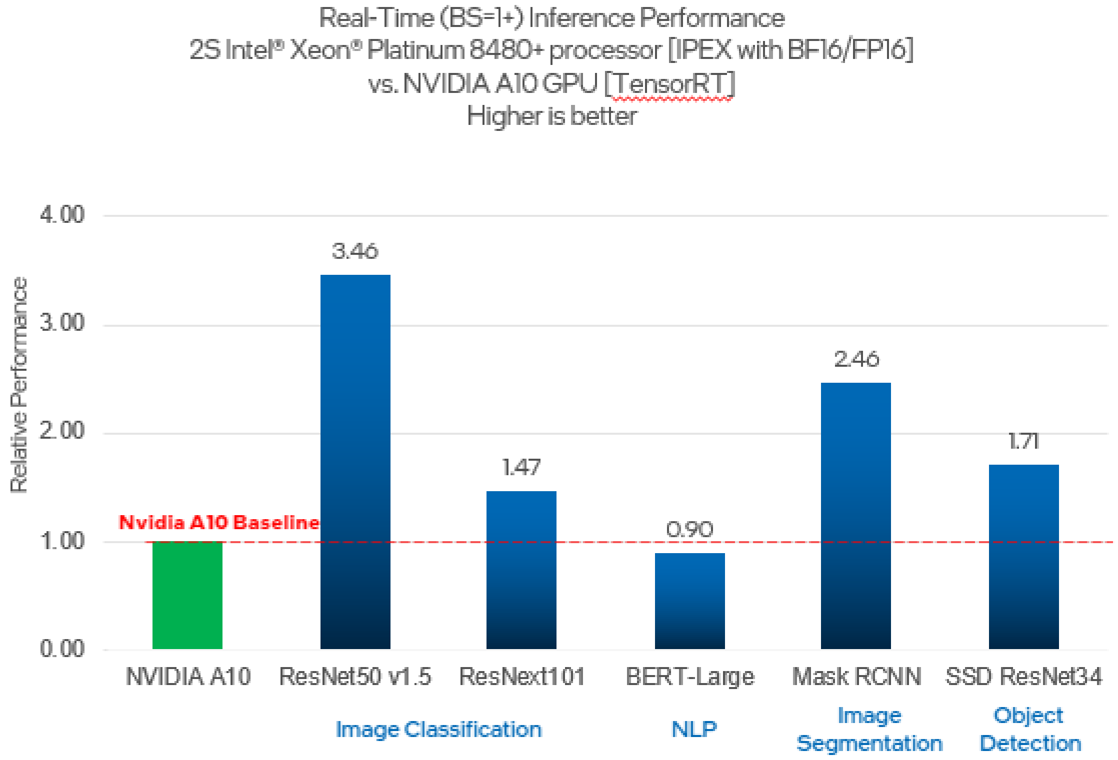
```python
import torch
from transformers import BertModel
############# code changes ###############
import intel_extension_for_pytorch as ipex
############# code changes ###############

model = BertModel.from_pretrained(args.model_name)
model.eval()

vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

#################### code changes #################
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)
#################### code changes #################

with torch.no_grad():
    d = torch.randint(vocab_size, size=[batch_size, seq_length])
    ############################# code changes #####################
    d = d.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
    ############################# code changes #####################
        model = torch.jit.trace(model, (d,), strict=False)
        model = torch.jit.freeze(model)

        model(data)
```

Training with Intel Extension for PyTorch Usage Example

```python
import torch
import torchvision
import intel_extension_for_pytorch as ipex

LR = 0.001
DOWNLOAD = True
DATA = 'datasets/cifar10/'

transform = torchvision.transforms.Compose([
  torchvision.transforms.Resize((224, 224)),
  torchvision.transforms.ToTensor(),
  torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
train_dataset = torchvision.datasets.CIFAR10(
  root=DATA,
  train=True,
  transform=transform,
  download=DOWNLOAD,
)
train_loader = torch.utils.data.DataLoader(
  dataset=train_dataset,
  batch_size=128
)
```

```python
model = torchvision.models.resnet50()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = LR, momentum=0.9)
model.train()
model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)

for batch_idx, (data, target) in enumerate(train_loader):
  optimizer.zero_grad()
  with torch.cpu.amp.autocast():
    output = model(data)
    loss = criterion(output, target)
    loss.backward()
  optimizer.step()
  print(batch_idx)
torch.save({
  'model_state_dict': model.state_dict(),
  'optimizer_state_dict': optimizer.state_dict(),
  }, 'checkpoint.pth')
```

Intel Extension for PyTorch Performance



1.8x higher average* BF16/FP16 inference performance vs Nvidia A10 GPU[3]

Benchmark data for the Intel® 4th Gen Xeon Scalable Processors can be found here.

Ref: https://venturebeat.com/ai/unlocking-generative-ai-with-ubiquitous-hardware-and-open-software/

Deploying with INT8

Low-precision Optimization – INT8

What is Quantization?

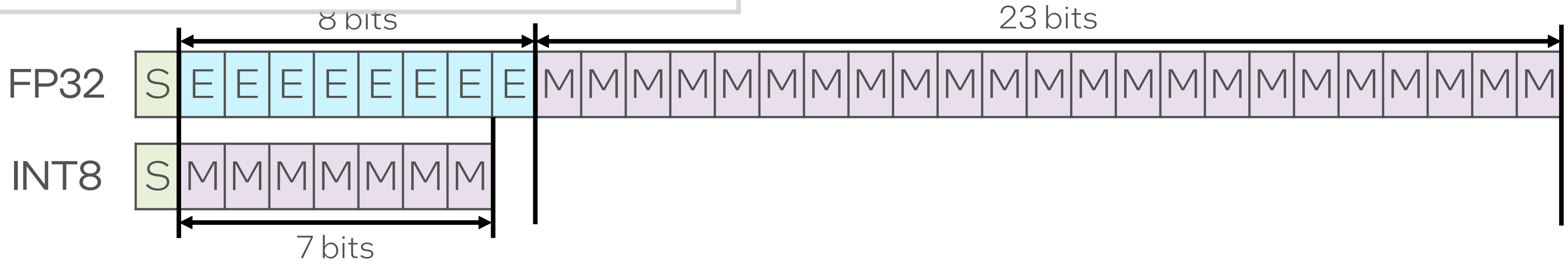
- An approximation method
- The process of mapping values from a large set (e.g., continuous, FP64/FP32) to those with smaller set (e.g., countable, BF16, INT8)

How to Quantize?

- PyTorch quantization
- **IPEX quantization (with or w/o INC integration)**
- Inter Neural Compressor (INC)

Why Quantization?

- Significant performance increase with similar accuracy



Quantization Workflow and API

Static Quantization

1. Import `intel_extension_for_pytorch` as `ipex`.
2. Import `prepare` and `convert` from `intel_extension_for_pytorch.quantization`.
3. Instantiate a config object from `torch.ao.quantization.QConfig` to save configuration data during calibration.
4. Prepare model for calibration.
5. Perform calibration against dataset.
6. Invoke `ipex.quantization.convert` function to apply the calibration configure object to the fp32 model object to get an INT8 model.
7. Save the INT8 model into a `pt` file.

```
import os
import torch
#################### code changes ####################
import intel_extension_for_pytorch as ipex
from intel_extension_for_pytorch.quantization import prepare, convert
######################################################

model = Model()
model.eval()
data = torch.rand(<shape>)

qconfig = ipex.quantization.default_static_qconfig
# Alternatively, define your own qconfig:
#from torch.ao.quantization import MinMaxObserver, PerChannelMinMaxObserver, QConfig
#qconfig = QConfig(activation=MinMaxObserver.with_args(qscheme=torch.per_tensor_affine, dtype=torch.quint8),
#        weight=PerChannelMinMaxObserver.with_args(dtype=torch.qint8, qscheme=torch.per_channel_symmetric))
prepared_model = prepare(model, qconfig, example_inputs=data, inplace=False)

for d in calibration_data_loader():
  prepared_model(d)

converted_model = convert(prepared_model)
with torch.no_grad():
  traced_model = torch.jit.trace(converted_model, data)
  traced_model = torch.jit.freeze(traced_model)

traced_model.save("quantized_model.pt")
```

Dynamic Quantization

1. Import `intel_extension_for_pytorch` as `ipex`.
2. Import `prepare` and `convert` from `intel_extension_for_pytorch.quantization`.
3. Instantiate a config object from `torch.ao.quantization.QConfig` to save configuration data during calibration.
4. Prepare model for quantization.
5. Convert the model.
6. Run inference to perform dynamic quantization.
7. Save the INT8 model into a `pt` file.

```
import os
import torch
#################### code changes ####################
import intel_extension_for_pytorch as ipex
from intel_extension_for_pytorch.quantization import prepare, convert
######################################################

model = Model()
model.eval()
data = torch.rand(<shape>)

dynamic_qconfig = ipex.quantization.default_dynamic_qconfig
# Alternatively, define your own qconfig:
#from torch.ao.quantization import MinMaxObserver, PlaceholderObserver, QConfig
#qconfig = QConfig(
#        activation = PlaceholderObserver.with_args(dtype=torch.float, compute_dtype=torch.quint8),
#        weight = PerChannelMinMaxObserver.with_args(dtype=torch.qint8, qscheme=torch.per_channel_symmetric))
prepared_model = prepare(model, qconfig, example_inputs=data)

converted_model = convert(prepared_model)
with torch.no_grad():
  traced_model = torch.jit.trace(converted_model, data)
  traced_model = torch.jit.freeze(traced_model)

traced_model.save("quantized_model.pt")
```

TorchScript and torch.compile()

TorchScript

- Converts PyTorch model into a graph for faster execution
- torch.jit.trace() traces and records all operations in the computational graph; requires a sample input
- torch.jit.script() parses the Python source code of the model and compiles the code into a graph; sample input not required

Resnet50

```
import torch
import torchvision.models as models

model = models.resnet50(weights='ResNet50_Weights.DEFAULT')
model.eval()
data = torch.rand(1, 3, 224, 224)

#################### code changes ####################
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)
######################################################

with torch.no_grad(), torch.cpu.amp.autocast():
  model = torch.jit.trace(model, torch.rand(1, 3, 224, 224))
  model = torch.jit.freeze(model)

  model(data)
```

torch.compile() – in BETA

- Makes PyTorch code run faster by just-in-time (JIT)-compiling PyTorch code into optimized kernels

Verifying That AMX Is Used

How to Check If AMX Is Enabled

- On bash terminal, enter the following command:
  - *cat /proc/cpuinfo*
- Check the “flags” section for amx_bf16, amx_int8
- Alternatively, you can use:
  - *lscpu | grep amx*
- If you do not see them, upgrade to Linux kernel 5.17 and above

```
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse s
se2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpu
id aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 s
se4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 cat_
l2 cdp_l3 invpcid_single intel_ppin cdp_l2 ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_
ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a avx512f avx512dq rdseed adx smap avx512ifma clflus
hopt clwb intel_pt avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_
mbm_local split_lock_detect avx_vnni avx512_bf16 wbnoinvd dtherm ida arat pln pts hwp hwp_act_window hwp_epp hwp_pkg_req hfi
 avx512vbmi umip pku ospke waitpkg avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg tme avx512_vpopcntdq la57 rdp
id bus_lock_detect cldemote movdiri movdir64b enqcmd fsrm uintr avx512_vp2intersect md_clear serialize tsxldtrk pconfig arch
_lbr amx_bf16 avx512_fp16 amx_tile amx_int8 flush_l1d arch_capabilities
```

How to Check AMX Is Actually Used

- Generate oneDNN Verbose logs using guide and parser
- To enable verbosity, set environment variables:
  - export DNNL_VERBOSE=1
  - export DNNL_VERBOSE_TIMESTAMP=1
- Set a Python breakpoint RIGHT AFTER one iteration of training/inference

oneDNN Verbose Sample Output

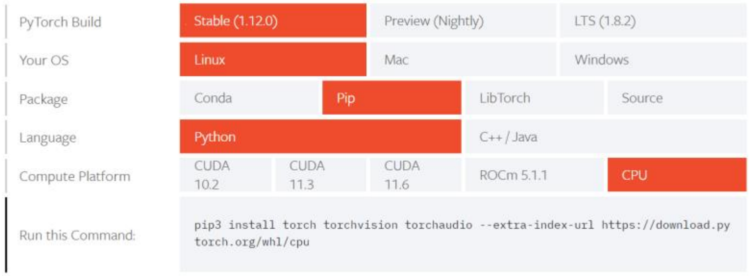
Sample oneDNN Verbose Output

```
onednn_verbose,info,oneDNN v2.6.0 (commit 52b5f107dd9cf10910aaa19cb47f3abf9b349815)
onednn_verbose,info,cpu,runtime:OpenMP,nthr:32
onednn_verbose,info,cpu,isa:Intel AVX-512 with Intel DL Boost
onednn_verbose,info,gpu,runtime:none
onednn_verbose,info,prim_template:timestamp,operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,problem_desc,exec_time
onednn_verbose,1678917979730.501953,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abcd:f0 dst_f32:p:blocked:Acdb16a:f0,attr-scratchpad:user ,,1x1x1x37,0.00292969
onednn_verbose,1678917979730.888916,exec,cpu,convolution,jit:avx512_core,forward_training,src_f32::blocked:abcd:f0 wei_f32:p:blocked:Acdb16a:f0 bia_undef::undef::f0 dst_f3
onednn_verbose,1678917979732.105957,exec,cpu,reorder,jit:uni,undef,src_f32:p:blocked:aBcd16b:f0 dst_f32::blocked:abcd:f0,attr-scratchpad:user ,,1x1x1x48000,0.0649414
onednn_verbose,1678917980009.694092,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abc:f0 dst_f32::blocked:acb:f0,attr-scratchpad:user ,,1x60x305,0.00878906
onednn_verbose,1678917980011.387939,exec,cpu,convolution,brgconv:avx512_core,forward_training,src_f32::blocked:acb:f0 wei_f32::blocked:Acb32a:f0 bia_f32::blocked:a:f0 dst_
onednn_verbose,1678917980012.134033,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abc:f0 dst_f32::blocked:acb:f0,attr-scratchpad:user ,,1x1024x301,0.278076
onednn_verbose,1678917980012.912109,exec,cpu,reorder,simple:any,undef,src_f32:p:blocked:Acb48a:f0 dst_f32::blocked:Acb64a:f0,attr-scratchpad:user ,,1024x1024x1,3.31201
```

- Note the ISA. For AMX, you should see the following:
  - Intel AMX with bfloat16 and 8-bit integer support
- Check for AMX in the primitive implementation:

```
onednn_verbose,1673049613345.454102,exec,cpu,convolution,brgconv:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0 wei_
onednn_verbose,1673049613348.691895,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
onednn_verbose,1673049613353.259033,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
onednn_verbose,1673049613364.104980,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
```

How to get the Intel Extension for PyTorch



Note: Intel® Extension for PyTorch* has PyTorch version requirement. Check the mapping table here.

- pip wheel - CPU:

```
python -m pip install intel_extension_for_pytorch
```

- pip wheel – GPU:

```
# General Python*
python -m pip install torch==1.13.0a0 torchvision==0.14.1a0 intel_extension_for_pytorch==1.13.10+xpu -f https://developer.intel.com/ipex-whl-stable-xpu

# Intel® Distribution for Python*
python -m pip install torch==1.13.0a0 torchvision==0.14.1a0 intel_extension_for_pytorch==1.13.10+xpu -f https://developer.intel.com/ipex-whl-stable-xpu-idp
```

More info: https://intel.github.io/intel-extension-for-pytorch/xpu/latest/

PyTorch AMX Training/Inference Code Samples

Training

GitHub: https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Features-and-Functionality/IntelPyTorch_TrainingOptimizations_AMX_BF16

Trains a ResNet50 model with Intel Extension for PyTorch and shows performance speedup with AMX BF16

Inference

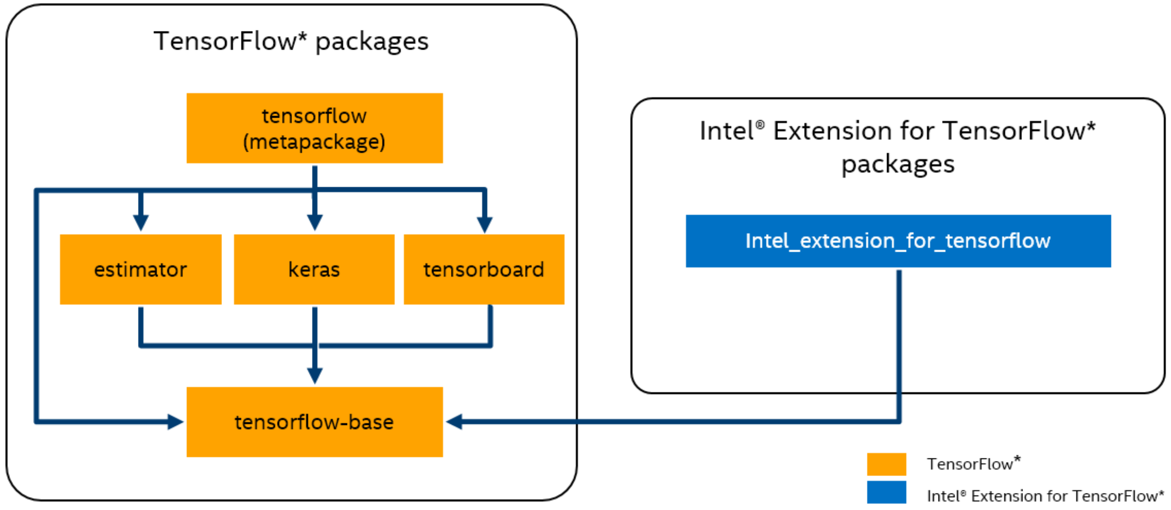
GitHub: https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Features-and-Functionality/IntelPyTorch_InferenceOptimizations_AMX_BF16_INT8

Performs inference on ResNet50 and BERT with Intel Extension for PyTorch and shows performance speedup with AMX BF16 and INT8 over VNNI INT8

Intel® Extension for TensorFlow

Intel® Extension for TensorFlow* (ITEX)

- Provide users with the up-to-date Intel software/hardware features
- Streamline the work to integrate oneDNN
- Unify user experiences on Intel CPU and GPU



Intel® Extension for TensorFlow* PyPI packages and dependencies

How to get the Intel® Extension for TensorFlow*

- pip wheel - GPU:

pip install --upgrade intel-extension-for-tensorflow[gpu]

- pip wheel - CPU (experimental)

pip install --upgrade intel-extension-for-tensorflow[cpu]

How to use Intel® Extension for TensorFlow* - FP32

No code changes, the default backend will be Intel GPU after installing intel-extension-for-tensorflow[gpu]

OR

```
import intel_extension_for_tensorflow as itex

#CPU, GPU or AUTO
backend = "GPU"
itex.set_backend(backend)
```

Mixed precision (FP16 and BF16)

- 2 choices:

- Use Keras mixed precision API in Stock TensorFlow
  - ITEX is compatible

mixed_precision.set_global_policy('mixed_float16')

OR

mixed_precision.set_global_policy('mixed_bfloat16')

- Use Advanced Auto Mixed Precision provided by ITEX for better performance
  - 2 modes of activation
  - Can be run from frozen graph

	FP16	BF16
Intel CPU	No	Yes
Intel GPU	Yes	Yes

Advanced Auto Mixed Precision - Python API

- import intel_extension_for_tensorflow as itex

- auto_mixed_precision_options = itex.AutoMixedPrecisionOptions()
- auto_mixed_precision_options.data_type = itex.BFLOAT16 (or itex.FLOAT16)

- graph_options = itex.GraphOptions()
- graph_options.auto_mixed_precision_options=auto_mixed_precision_options
- graph_options.auto_mixed_precision = itex.ON

- config = itex.ConfigProto(graph_options=graph_options)
- itex.set_backend("gpu", config) [in ITEX v1.0.0 and ITEX v1.1.0]
    - --> itex.set_config(config) [latest master branch]

Advanced Auto Mixed Precision - Environment Variable

- export ITEX_AUTO_MIXED_PRECISION=1
- export ITEX_AUTO_MIXED_PRECISION_DATA_TYPE="BFLOAT16" (or "FLOAT16“)

Optimizations under oneDNN, IPEX & ITEX

Operator optimizations

Memory/data layout optimizations

Graph optimizations

Mixed Precision

Operator Optimizations

- Replace default kernels by highly-optimized kernels (using Intel® oneDNN)
- Adapt to available instruction sets (AMX, AVX-512, AVX2, VNNI)
- Adapt to required precision:
  - **Training**: FP32, BF16
  - **Inference**: FP32, BF16, FP16, and INT8

	Intel® oneDNN
Convolution	2D/3D Direct Convolution/Deconvolution, Depthwise separable convolution 2D Winograd convolution
Inner Product	2D/3D Inner Production
Pooling	2D/3D Maximum 2D/3D Average (include/exclude padding)
Normalization	2D/3D LRN across/within channel, 2D/3D Batch normalization
Eltwise (Loss/activation)	ReLU(bounded/soft), ELU, Tanh; Softmax, Logistic, linear; square, sqrt, abs, exp, gelu, swish
Data manipulation	Reorder, sum, concat, View
RNN cell	RNN cell, LSTM cell, GRU cell
Fused primitive	Conv+ReLU+sum, BatchNorm+ReLU
Data type	f32, bfloat16, s8, u8

Operator optimizations

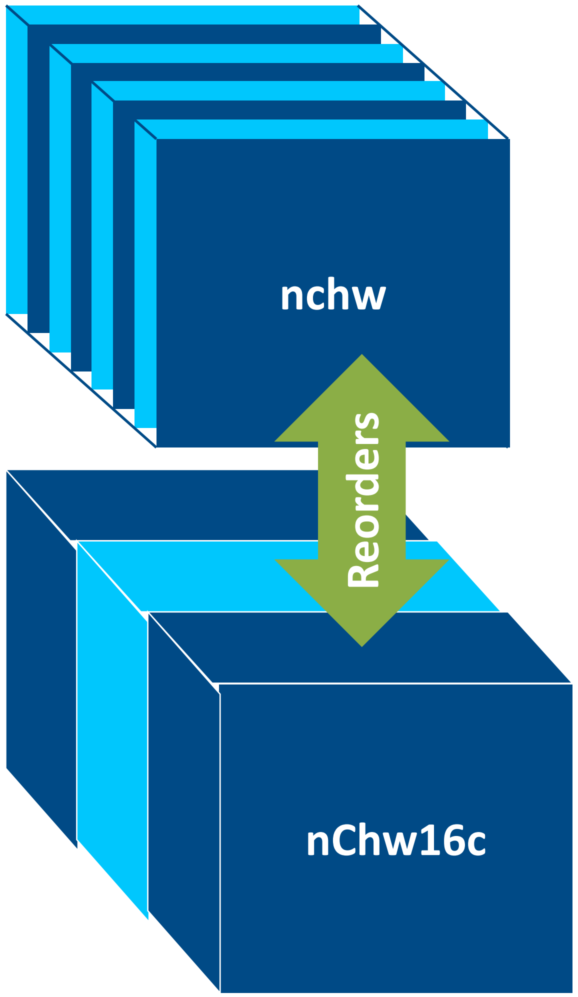
Memory/data layout optimizations

Graph optimizations

Mixed Precision

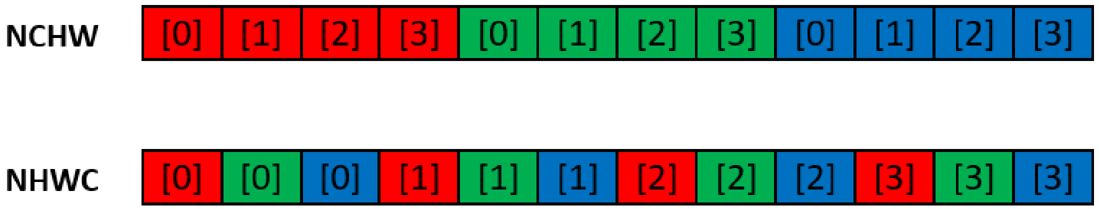
Memory Layouts Optimization

- Most popular memory layouts for image recognition are **NHWC** and **NCHW**
  - Challenging for Intel processors both for vectorization or for memory accesses
- Intel oneDNN convolutions use blocked layouts
  - Most popular oneDNN data format is nChw16c on AVX512+ systems and nChw8c on SSE4.1+ systems



More details: https://oneapi-src.github.io/oneDNN/dev_guide_understanding_memory_formats.html

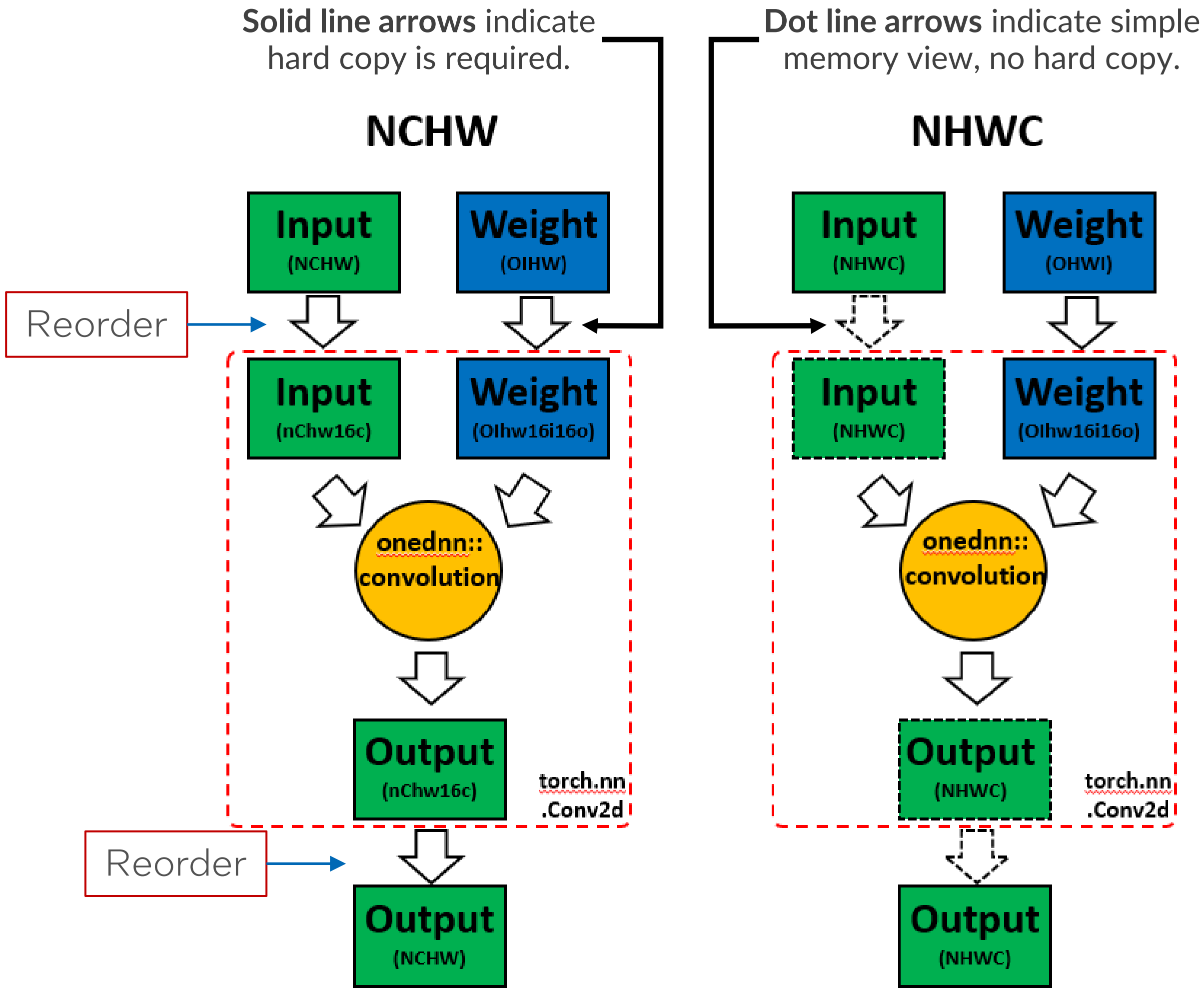
Data Layouts in PyTorch



- Used in Vision workloads
- NCHW
  - Default format
  - *torch.contiguous_format*
- NHWC
  - *torch.channels_last*
  - NHWC format yields higher performance with IPEX

Channels last conversion is now applied **automatically** with IPEX
Users do not have to explicitly convert input and weight for CV models.

Benefit of NHWC in IPEX



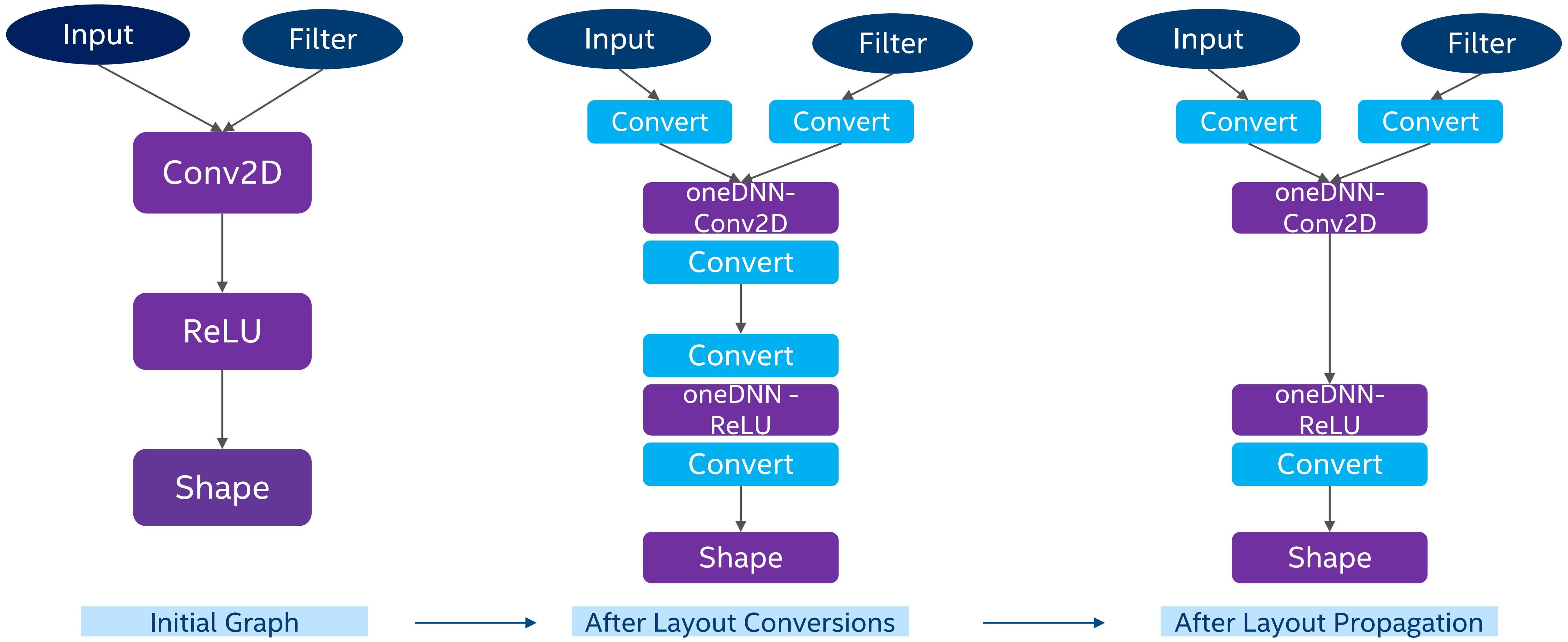
Operator optimizations

Memory/data layout optimizations

Graph optimizations

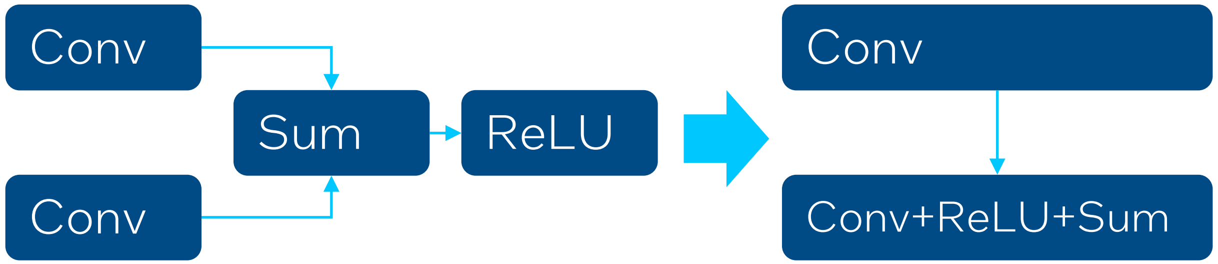
Mixed Precision

Graph Optimizations: Layout Propagation



Fusing Computations

- On Intel processors a high percentage of time is typically spent in bandwidth-limited ops such activation functions
  - ~40% of ResNet-50, even higher for inference
- The solution is to fuse BW-limited ops with convolutions or one with another to reduce the number of memory accesses
  - We fuse patterns: Conv+ReLU+Sum, BatchNorm+ReLU, etc…



Graph Optimizations: Fusion

Before Merge

After Merge

Fusing Computations in IPEX

- Intel® Extension for PyTorch in JIT/Torchscript mode can fuse:
  - Multi-head-attention fusion, Concat Linear, Linear+Add, Linear+Gelu, Add+LayerNorm fusion and etc.
- Hugging Face reports that ~70% of most popular NLP tasks in question-answering, text-classification, and token-classification can get performance benefits with such fusion patterns [1]
  - for both Float32 precision and BFloat16 Mixed precision

[1] https://huggingface.co/docs/transformers/perf_infer_cpu

Operator optimizations

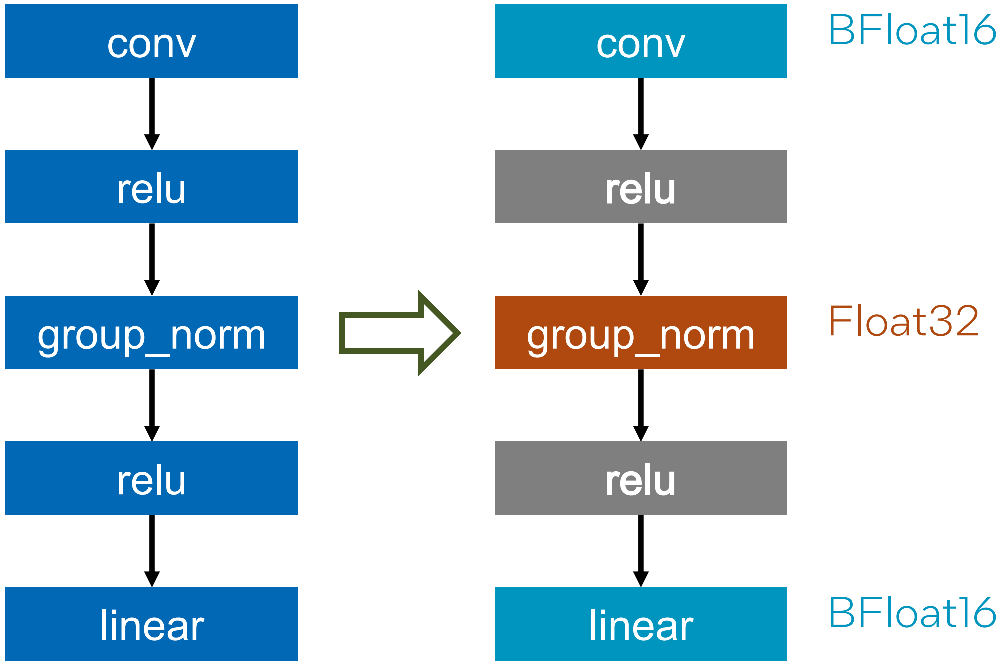
Memory/data layout optimizations

Graph optimizations

Mixed Precision

Auto Mixed Precision (AMP)

- 3 Categories of operators
  - lower_precision_fp
    - Computation bound operators that could get performance boost with BFloat16.
    - E.g.: conv, linear
  - Fallthrough
    - Operators that runs with both Float32 and BFloat16 but might not get performance boost with BFloat16.
    - E.g.: relu, max_pool2d
  - FP32
    - Operators that are not enabled with BFloat16 support yet. Inputs of them are casted into float32 before execution.
    - E.g.: max_pool3d, group_norm

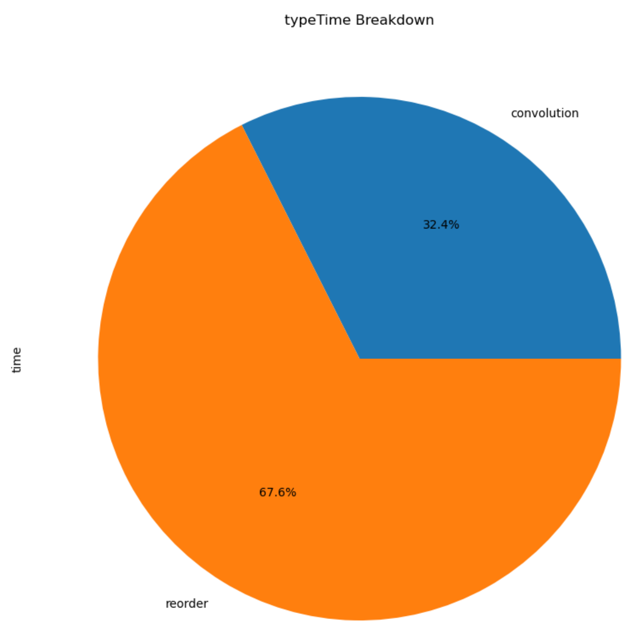


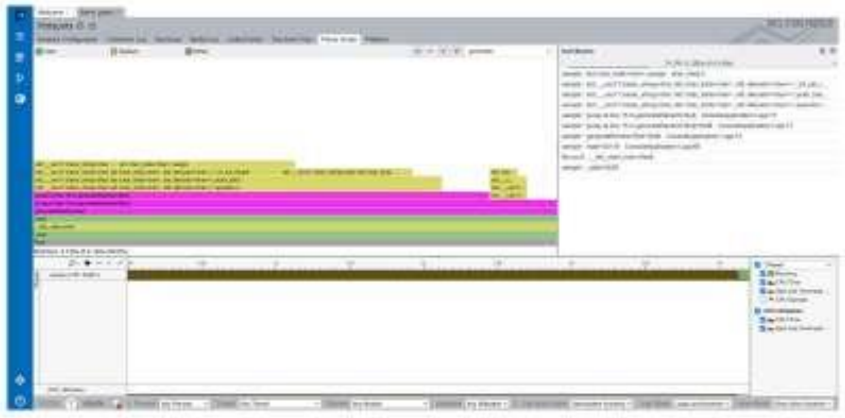
Profiling tools

Intel Profilers

- You can profile your application via oneDNN verbose logs.
  - DNN_VERBOSE=1 python application.py
  - You can also use profile_utils.py script to parse oneDNN verbose logs.
  - Code sample on oneDNN profiling can be found here: https://github.com/oneapi-src/oneAPI-samples/tree/master/Libraries/oneDNN/tutorials/profiling

- Another famous profiling tool is VTune from Intel which provides very deep hardware information and show them in easier way on how to **optimize the performance.** You can easily **find the hotspots** using VTune. (most costly functions)



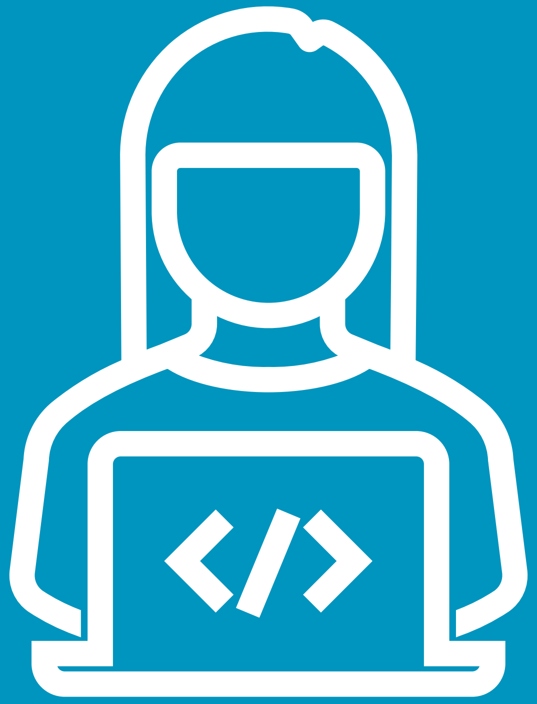


Recipe for Intel® Optimizations with IPEX

Easy Recipe for faster Intel® Optimizations with IPEX

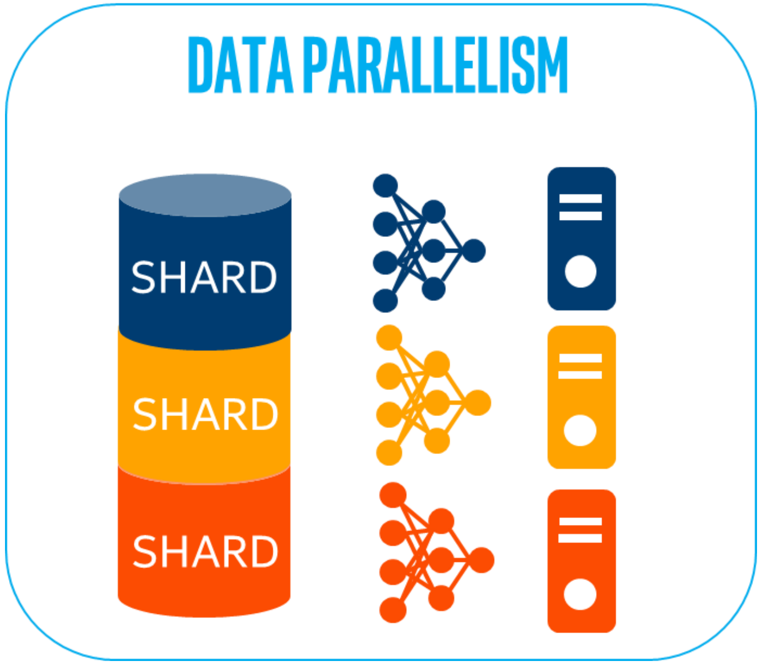
- Add IPEX
- Add some Warmup steps for oneDNN initialization
- Utilize AMX or XMX instruction sets with efficient bfloat16 data type
- Utilize graph mode with TorchScript
- Quantize model to INT8
- Runtime optimizations using ipexrun
- Distributed training with oneCCL/ DDP/Horovod
- Profile with oneDNN verbose / Pytorch Profiler / VTune for further analysis.

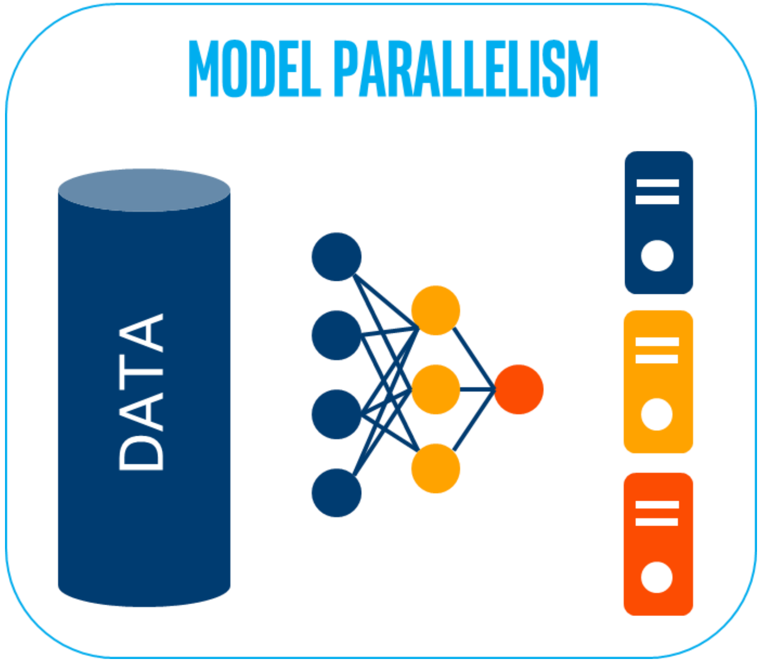
Demo: Trajectory Prediction of Vehicles



Distributed Training with Intel® oneCCL

Neural Network Parallelism





Data is processed in increments of N. Work on minibatch samples and distributed among the available resources.

The work is divided according to a split of the model. The sample minibatch is copied to all processors which compute part of the DNN.

source: https://arxiv.org/pdf/1802.09941.pdf

Intel® oneAPI Collective Communications Library (oneCCL)

- enables developers and researchers to quickly train DL models
- optimizes communication patterns to distribute model training across multiple nodes
- designed for easy integration into deep learning frameworks, whether they are implemented them from scratch or customizing existing ones
- [DistributedDataParallel (DDP) with Intel® oneCCL]
  - E.g mpirun -n 2 -l python Example_DDP.py
- [Horovod with Intel® oneCCL & PyTorch]
  - E.g horovodrun -np 2 python Example_horovod.py
  - Or e.g mpirun -np 2 python Example_horovod.py
- DeepSpeed (https://github.com/intel/intel-extension-for-deepspeed)
  - Deep learning optimization software suite that enables scale and speed for Deep Learning Training and inference of models with billions or trillions of parameters

Usage for Distributed Training with DDP

- 4 root devices, 4 GPUs
- 8 ranks and two ranks per GPU
- E.g mpirun -n 8 -l python Example_DDP.py

```
(base) ac.louie.tsai@florentia05:~> sycl-ls
Warning: SYCL_DEVICE_FILTER environment variable is set to level_zero.
To see the correct device id, please unset SYCL_DEVICE_FILTER.

[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) Graphics [0x0bd5] 1.3 [1.3.23937]
[ext_oneapi_level_zero:gpu:1] Intel(R) Level-Zero, Intel(R) Graphics [0x0bd5] 1.3 [1.3.23937]
[ext_oneapi_level_zero:gpu:2] Intel(R) Level-Zero, Intel(R) Graphics [0x0bd5] 1.3 [1.3.23937]
[ext_oneapi_level_zero:gpu:3] Intel(R) Level-Zero, Intel(R) Graphics [0x0bd5] 1.3 [1.3.23937]
```

- Monitor XPU usage using Intel® XPU manager:
  - https://www.intel.com/content/www/us/en/software/xpu
- xpumcli dump –d 0 –m 0,1,2,3,4,5

```
Timestamp, DeviceId, GPU Utilization (%), GPU Power (W), GPU Frequency (MHz), GPU Core Temperature
(Celsius Degree), GPU Memory Temperature (Celsius Degree), GPU Energy Consumed (J)

08:04:16.000,   0, 53.35, 234.08, 0.00,    ,    , 2018647.97
08:04:17.000,   0, 65.83, 341.15, 1600.00,    ,    , 2018956.02
08:04:18.000,   0, 92.52, 375.21, 900.00,    ,    , 2019332.25
08:04:19.000,   0, 92.54, 384.55, 1500.00,    ,    , 2019715.47
08:04:20.000,   0, 94.21, 387.95, 975.00,    ,    , 2020105.06
08:04:21.000,   0, 93.25, 386.10, 1600.00,    ,    , 2020491.66
08:04:22.000,   0, 94.21, 391.84, 800.00,    ,    , 2020881.66
```

```
[6] ZE_AFFINITY_MASK=0,1,2,3
[6] Iterations: 5. Warmup runs: 2
[6] Running on device: IntelGPU6
[6] Running on torch: 1.10.0a0+git90332b4
[6] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[6] Converting model to DDP & syncing...
[6] Starting warmup runs...
[6] Starting benchmark runs...
[6]       total:     459.63ms (458.7-460.4) +-1.15, 34.81 (imgs/s)
[6] csv,resnet50,16,0,34.81,1.10.0a0+git90332b4,IntelGPU6,2,5
[0] ZE_AFFINITY_MASK=0,1,2,3
[0] Iterations: 5. Warmup runs: 2
[0] Running on device: IntelGPU0
[0] Running on torch: 1.10.0a0+git90332b4
[0] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[0] Converting model to DDP & syncing...
[0] Starting warmup runs...
[0] Starting benchmark runs...
[0]       total:     459.24ms (458.3-460.4) +-1.49, 34.84 (imgs/s)
[0] csv,resnet50,16,0,34.84,1.10.0a0+git90332b4,IntelGPU0,2,5
[0] Total img/sec on 8 IntelGPU(s): 278.72133230304513
[3] ZE_AFFINITY_MASK=0,1,2,3
[3] Iterations: 5. Warmup runs: 2
[3] Running on device: IntelGPU3
[3] Running on torch: 1.10.0a0+git90332b4
[3] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[3] Converting model to DDP & syncing...
[3] Starting warmup runs...
[3] Starting benchmark runs...
[3]       total:     458.77ms (457.9-459.7) +-1.42, 34.88 (imgs/s)
[3] csv,resnet50,16,0,34.88,1.10.0a0+git90332b4,IntelGPU3,2,5
[4] ZE_AFFINITY_MASK=0,1,2,3
[4] Iterations: 5. Warmup runs: 2
[4] Running on device: IntelGPU4
[4] Running on torch: 1.10.0a0+git90332b4
[4] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[4] Converting model to DDP & syncing...
[4] Starting warmup runs...
[4] Starting benchmark runs...
[4]       total:     459.09ms (458.4-460.0) +-1.22, 34.85 (imgs/s)
[4] csv,resnet50,16,0,34.85,1.10.0a0+git90332b4,IntelGPU4,2,5
[2] ZE_AFFINITY_MASK=0,1,2,3
[2] Iterations: 5. Warmup runs: 2
[2] Running on device: IntelGPU2
[2] Running on torch: 1.10.0a0+git90332b4
[2] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[2] Converting model to DDP & syncing...
[2] Starting warmup runs...
[2] Starting benchmark runs...
[2]       total:     459.20ms (458.7-460.1) +-1.00, 34.84 (imgs/s)
[2] csv,resnet50,16,0,34.84,1.10.0a0+git90332b4,IntelGPU2,2,5
[1] ZE_AFFINITY_MASK=0,1,2,3
[1] Iterations: 5. Warmup runs: 2
[1] Running on device: IntelGPU1
[1] Running on torch: 1.10.0a0+git90332b4
[1] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[1] Converting model to DDP & syncing...
[1] Starting warmup runs...
[1] Starting benchmark runs...
[1]       total:     459.00ms (458.1-460.4) +-1.91, 34.86 (imgs/s)
[1] csv,resnet50,16,0,34.86,1.10.0a0+git90332b4,IntelGPU1,2,5
[5] ZE_AFFINITY_MASK=0,1,2,3
[5] Iterations: 5. Warmup runs: 2
[5] Running on device: IntelGPU5
[5] Running on torch: 1.10.0a0+git90332b4
[5] ModelType: resnet50, Kernels: DPCPP Input shape: 16x3x224x224
[5] Converting model to DDP & syncing...
[5] Starting warmup runs...
[5] Starting benchmark runs...
[5]       total:     458.69ms (457.9-460.5) +-1.88, 34.88 (imgs/s)
[5] csv,resnet50,16,0,34.88,1.10.0a0+git90332b4,IntelGPU5,2,5
[7] ZE_AFFINITY_MASK=0,1,2,3
[7] Iterations: 5. Warmup runs: 2
[7] Running on device: IntelGPU7
[7] Running on torch: 1.10.0a0+git90332b4
```

DeepSpeed – Introduction

- Deep learning optimization software that enables scale and speed for Deep Learning training and inference for large scale models

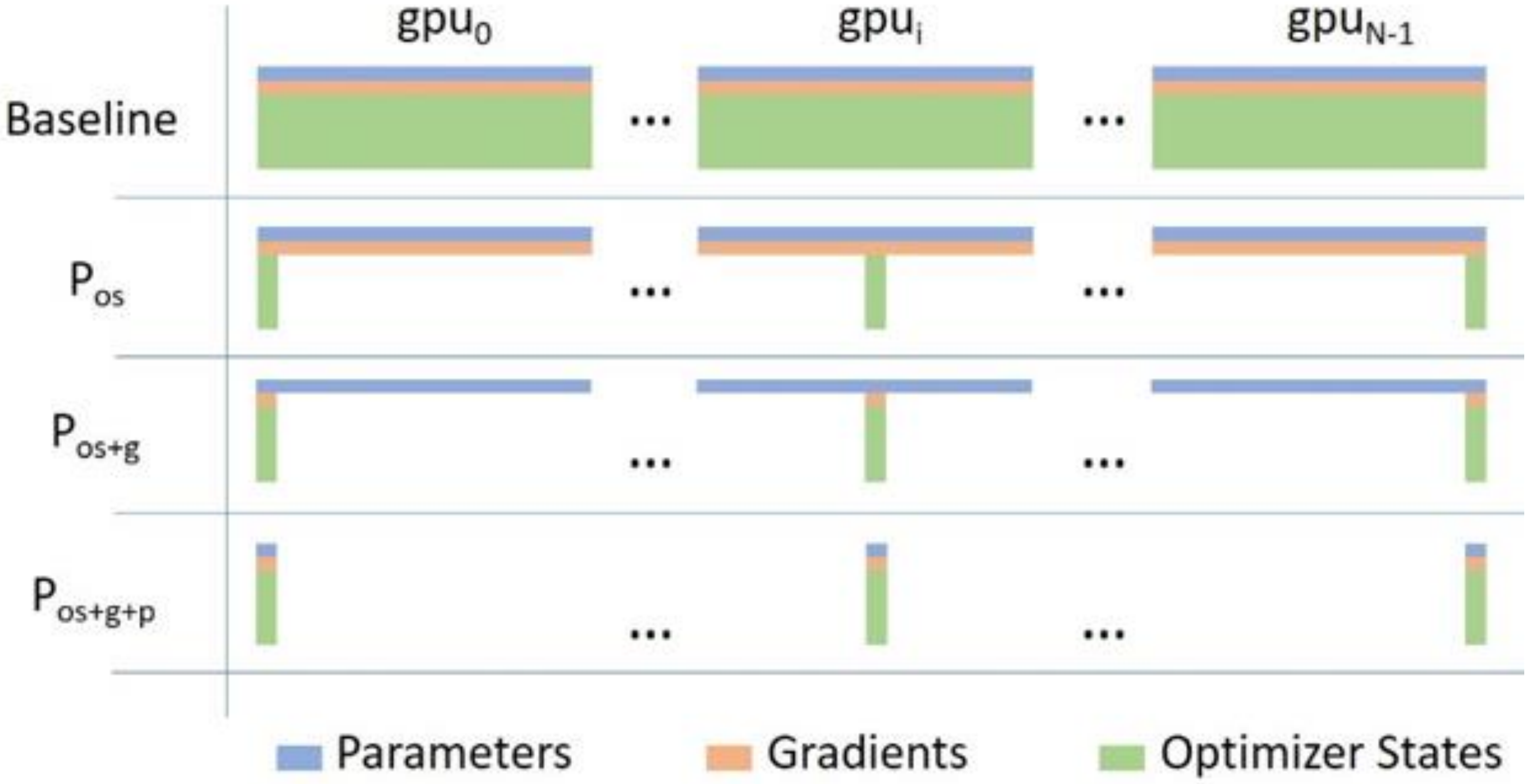
$\Rightarrow$ Train/inference models with billions or trillions of parameters

$\Rightarrow$ Efficiently scale to thousands of computing units

$\Rightarrow$ Train/inference on GPU system with limited GPU memory

$\Rightarrow$ Low latency and high throughput for inference

DeepSpeed training technology – ZeRO stage 1/2/3



Intel Extension for DeepSpeed

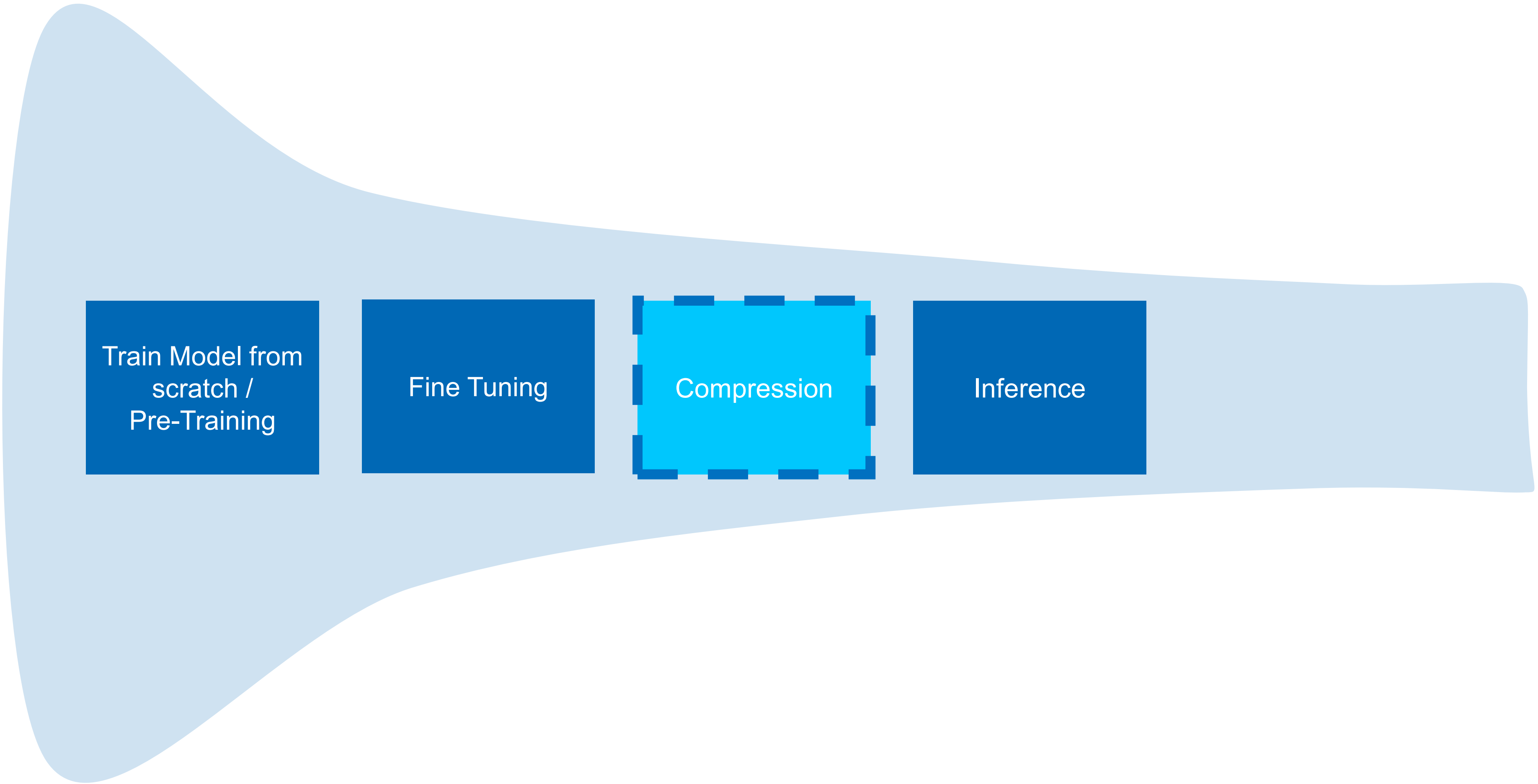
- Intel® Extension for DeepSpeed* is an extension that brings Intel GPU support to DeepSpeed
- Example to train GPT 3.6B, 20B, 175B

https://github.com/intel/intel-extension-for-deepspeed/tree/main/examples

A use-case: Aurora GPT

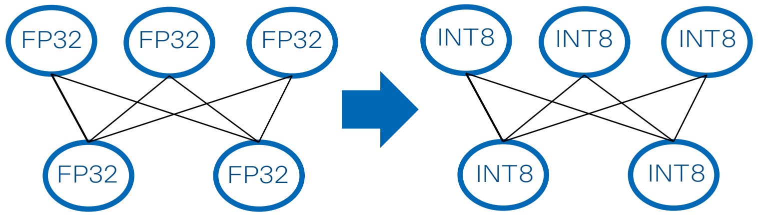
- Argonne National Labs and Intel are currently training a 1 trillion parameter GPT model for science, called AuroraGPT.
- AuroraGPT, also referred to as “ScienceGPT,” will have a chatbot interface for researchers to use for insights and answers.
- The AI model could be used in various scientific fields, including biology, cancer research, and climate change.
- The training process will take several months and will scale from 256 to 10,000 nodes using Intel Extension for DeepSpeed

Deep Learning Funnel Pipeline

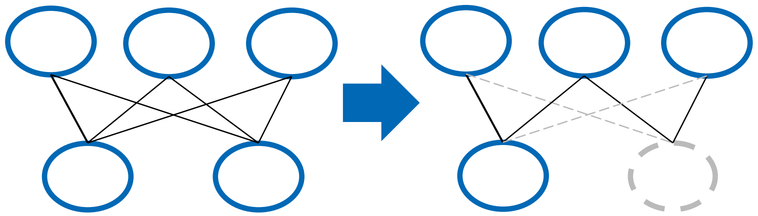


Deep Learning Inference Optimization

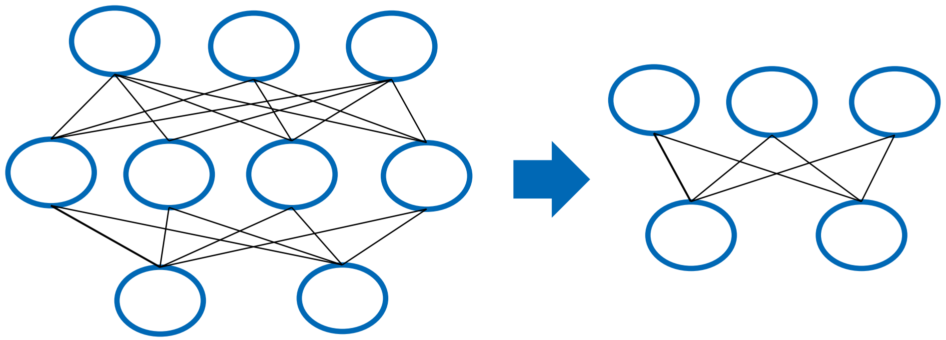
Quantization



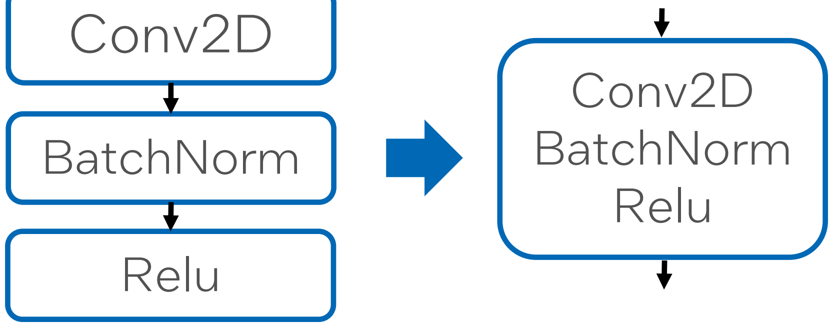
Pruning



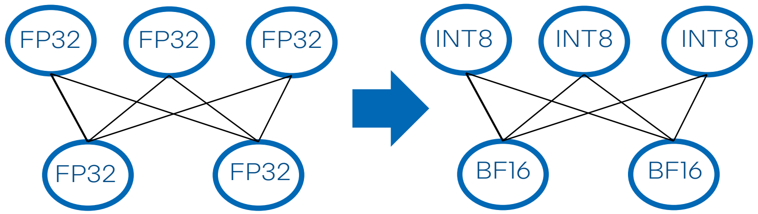
Knowledge Distillation



Graph Optimization



Mixed Precision Graph Optimization



INC use automatic accuracy-driven tuning strategies to help user **easily & quickly** find out the best optimization methods above.

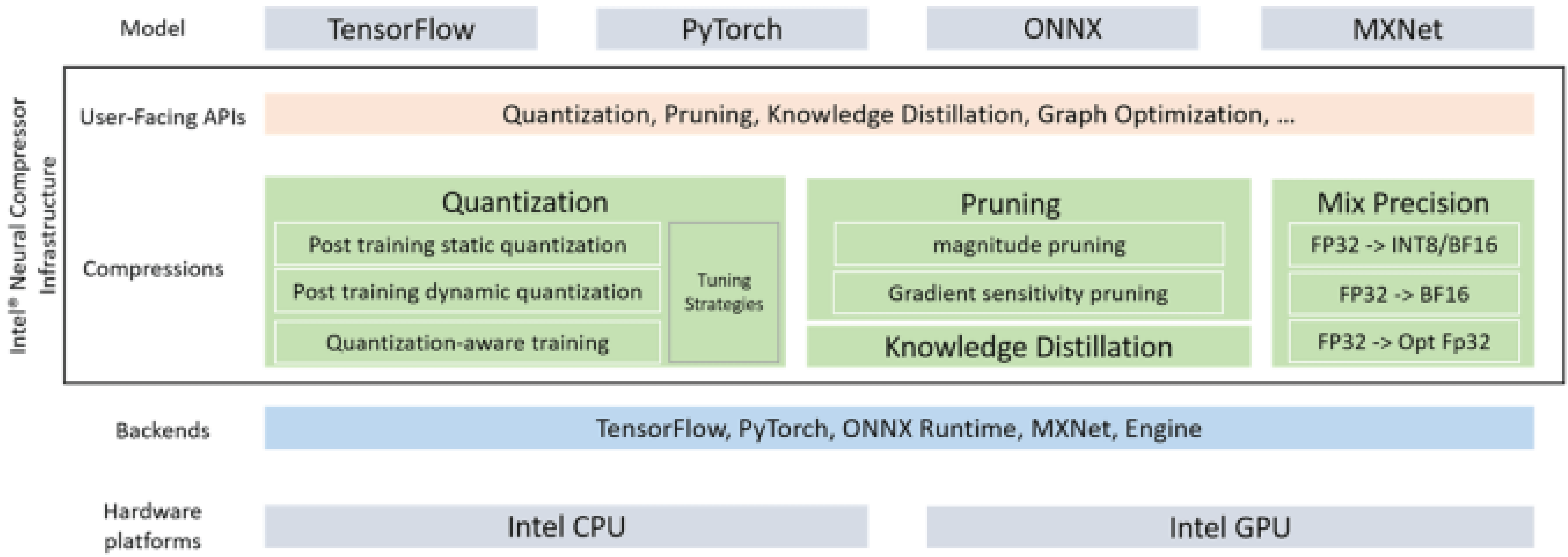
Intel tools necessary for it

Intel Neural Compressor

Intel Extension for Transformers

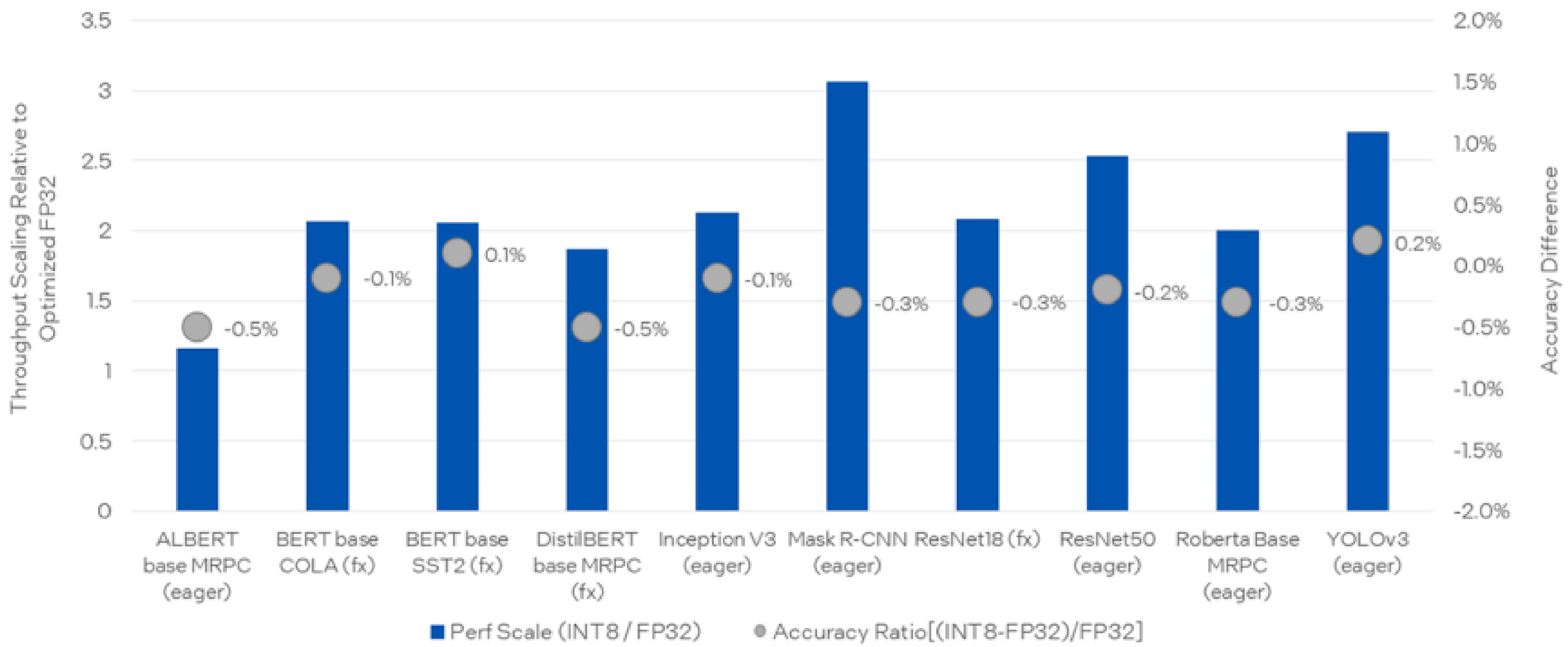
Includes more generic transformers optimizations

Intel® Neural Compressor Architecture



Performance vs Accuracy on INT8 Quantization with Intel® Neural Compressor

Post-Training Quantization and PyTorch* Inference



Testing Date: Performance results are **based on testing by Intel as of June 7, 2022** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: PyTorch* v1.11.0+cpu; Intel® Neural Compressor v1.12; Platform: Intel® Xeon® Platinum 8380 CPU @ 2.30GHz; 1 socket; 4 cores/instance; 10 instances; batch size = 1; Turbo: On; BIOS version: SE5C6200.86B.0022.D64.2105220049; System DDR Mem Config: 256GB (16x16GB DDR4 3200MT/s [3200MT/s]); OS: Ubuntu 20.04.1 LTS; Kernel: 5.4.0-42-generic. Full benchmark results available on Intel® Neural Compressor GitHub* (https://github.com/intel/neural-compressor/blob/master/docs/validated_model_list.md).

Getting Intel® Neural Compressor

- Intel® Neural Compressor is included in the **Intel® AI Analytics Toolkit (AI Kit):**
  - https://www.intel.com/content/www/us/en/developer/tools/oneapi/ai-analytics-toolkit-download.html?operatingsystem=linux
- Download the Stand-Alone Version:
  - https://intel.github.io/neural-compressor/latest/docs/source/Welcome.html#installation
- Use Intel® Developer Cloud:
  - https://www.intel.com/content/www/us/en/secure/forms/devcloud/enrollment.html?tgt=www.intel.com/content/www/us/en/secure/forms/devcloud-enrollment/account-provisioning.html

Intel Extension for Transformers – Overview

- Intel Extension for Transformers (ITREX): Built on top of INC ecosystem and Hugging Face
- Its target is the democratization of NLP and Transformers for both training/fine-tuning and inference
- Brings compression and model optimizations in a high-level HF – like API
- Staging area for all Intel's transformer feature enhancements:
  - Upstream to HF as much as possible (Transformers + Optimum)
  - Intel's differentiation remains, e.g., NAS, MoE, dynamic model, etc., and is ready for future upstream

Intel® Extension esp. for LLMs:

Intel® Extension for Transformers:

https://github.com/intel/intel-extension-for-transformers

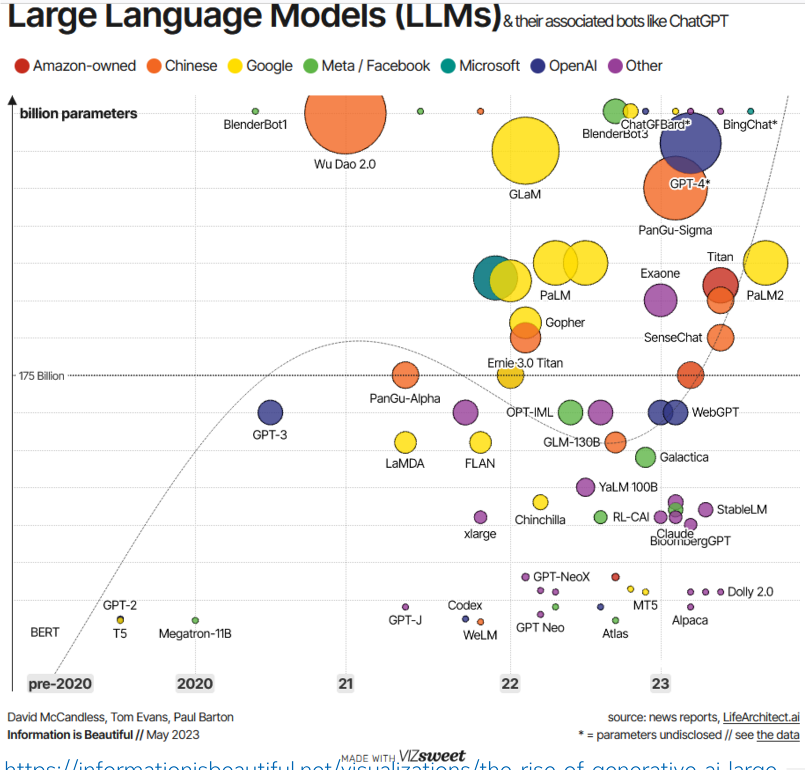
Intel® Extension for Transformers is an innovative toolkit to accelerate Transformer-based models on Intel platforms, in particular effective on 4th Intel Xeon Scalable processor Sapphire Rapids The toolkit provides many key features and examples such as **Stable Diffusion, GPT-J-6B, GPT-NEOX, BLOOM-176B, T5, Flan-T5 and end-to-end workflows such as SetFit-based text classification and document level sentiment analysis (DLSA)**

Get it through: pip install intel-extension-for-transformers

Intel® Extension for DeepSpeed:

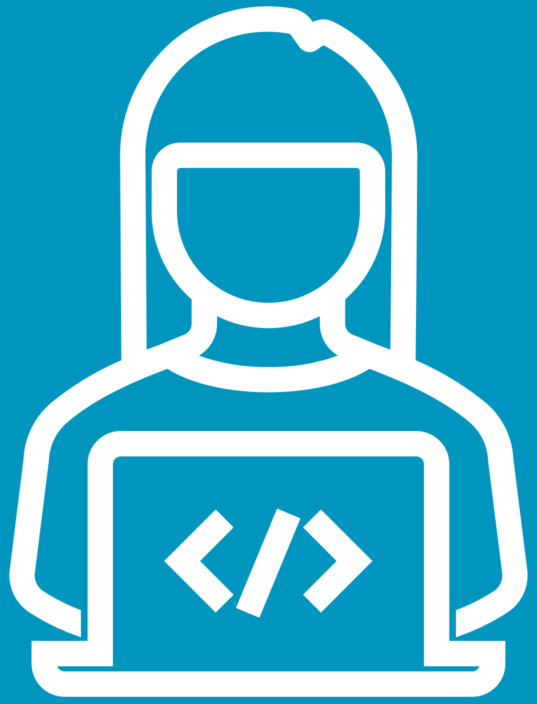
https://github.com/intel/intel-extension-for-deepspeed

- Deep learning optimization software suite that enables scale and speed for Deep Learning Training and inference
  - Train/inference models with billions or trillions of parameters
  - Efficiently scale to thousands of computing units
  - Train/inference on GPU system with limited GPU memory
  - Low latency and high throughput for inference
  - **Get it through: pip install intel-extension-for-deepspeed**



[Ref: https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/](https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/)

Demo Llama2 Inference on Multi-GPU



Engineer Data | Create Machine Learning & Deep Learning Models | Deploy

Container Repository **oneContainer**	oneAPI powered **AI Reference Kits**	MLOps **Cnvrg.io**	Developer Sandbox **Intel® Developer Cloud**	Annotation/Training/Optimization Intel® GETi

Connect AI to Big Data — Apache Spark — BigDL (previously "Analytics Zoo")

Accelerate End-to-End Data Science and AI — AI Analytics Toolkit

Data Analytics Scale

MODIN, SciPy, pandas, NumPy

Optimized Frameworks and Middleware

TensorFlow, PyTorch, mxnet, PaddlePaddle, scikit learn, ONNX, LightGBM, XGBoost, CatBoost

Optimize Models

Automate Model Tuning AutoML — Automate Low-Precision Optimization

SigOpt — **Intel Neural Compressor**

OpenVINO™ Toolkit

Write Once

Deploy Auto-Optimized

Anywhere

w/ Intel Optimizations

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM, intel CORE, intel XEON, intel ARC GRAPHICS, intel DATA CENTER GPU, intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL- Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

Conclusion

Key Takeaways & Call to Action

- Intel provides a plethora of AI software tools
- 100% Python
- No to very minimal code changes necessary
- The new Intel® AMX & XMX instruction set accelerates training and inference workloads in BF16 and INT8 for 4th Gen Intel® Xeon® Scalable Processor & Intel® Data Center GPU Max Series respectively
- Multiple Intel® extensions for running your LLM models on XPU/CPU.
- "Low-hanging fruit" to run AI workloads efficiently on Intel hardware
- Code samples are available to get started.

GITHUB Repo

Document

Download the tools:

Intel® oneAPI Toolkits

Intel Extension for PyTorch

Intel Extension for DeepSpeed

Intel Extension for Transformers

VTune Profiler

Getting Started Samples

Model Zoo for Intel® Architecture GitHub

PyTorch Benchmarking Configurations

<u>4th Generation Intel® Xeon® Scalable Processors</u>

Hardware and software configuration (measured October 24, 2022):

- **Deep Learning config:**
  - Hardware configuration for Intel® Xeon® Platinum 8480+ processor (formerly code named Sapphire Rapids): 2 sockets, 56 cores, 350 watts, 16 x 64 GB DDR5 4800 memory, BIOS version EGSDCRB1.SYS.8901.P01.2209200243, operating system: CentOS* Stream 8, using Intel® Advanced Matrix Extensions (Intel® AMX) int8 and bf16 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.7 optimized kernels integrated into Intel® Extension for PyTorch* v1.13, Intel® Extension for TensorFlow* v2.12, and Intel® Distribution of OpenVINO™ toolkit v2022.3. Measurements may vary.
  - Wall power refers to platform power consumption.
  - If the dataset is not listed, a synthetic dataset was used to measure performance. Accuracy (if listed) was validated with the specified dataset.
- **Transfer Learning config:**
  - Hardware configuration for Intel® Xeon® Platinum 8480+ processor (formerly code named Sapphire Rapids): Use DLSA single node fine tuning, Vision Transfer Learning using single node, 56 cores, 350 watts, 16 x 64 GB DDR5 4800 memory, BIOS version EGSDREL1.SYS.8612.P03.2208120629, operating system: Ubuntu 22.04.1 LT, using Intel® Advanced Matrix Extensions (Intel® AMX) int8 and bf16 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.6 optimized kernels integrated into Intel® Extension for PyTorch* v1.12, and Intel® oneAPI Collective Communications Library v2021.5.2. Measurements and some software configurations may vary.

<u>3rd Generation Intel® Xeon® Scalable Processors</u>

Hardware and software configuration (measured October 24, 2022):

- Hardware configuration for Intel® Xeon® Platinum 8380 processor (formerly code named Ice Lake): 2 sockets, 40 cores, 270 watts, 16 x 64 GB DDR5 3200 memory, BIOS version SE5C620.86B.01.01.0005.2202160810, operating system: Ubuntu 22.04.1 LTS, int8 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.6.0 optimized kernels integrated into Intel® Extension for PyTorch* v1.12, Intel® Extension for TensorFlow* v2.10, and Intel® oneAPI Data Analytics Library (oneDAL) 2021.2 optimized kernels integrated into Intel® Extension for Scikit-learn* v2021.2. XGBoost v1.6.2, Intel® Distribution of Modin* v0.16.2, Intel oneAPI Math Kernel Library (oneMKL) v2022.2, and Intel® Distribution of OpenVINO™ toolkit v2022.3. Measurements may vary.
- If the dataset is not listed, a synthetic dataset was used to measure performance. Accuracy (if listed) was validated with the specified dataset.

*All performance numbers are acquired running with 1 instance of 4 cores per socket

Thank you for your attention!

Questions?

Complete this Short Survey

Give us feedback…

- Tell us what you thought of this webinar.
- Give us feedback on what topics you'd like to see in future webinars.

Webinar Survey

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.
No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

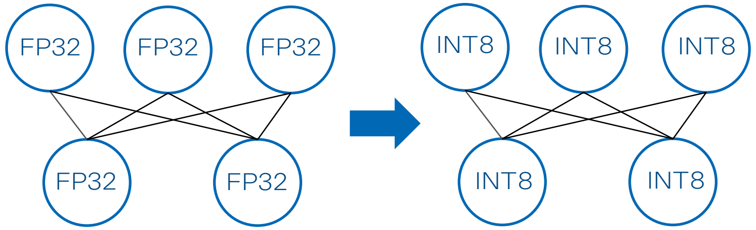
intel®

Appendix

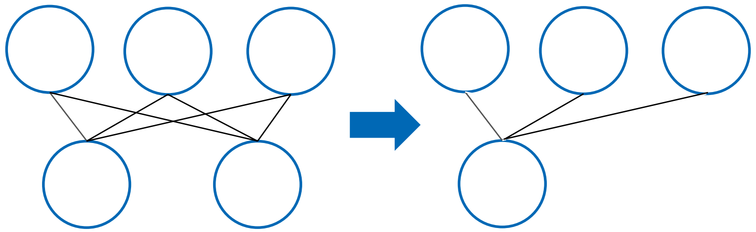
Intel® Neural Compressor

Deep Learning Inference Optimization

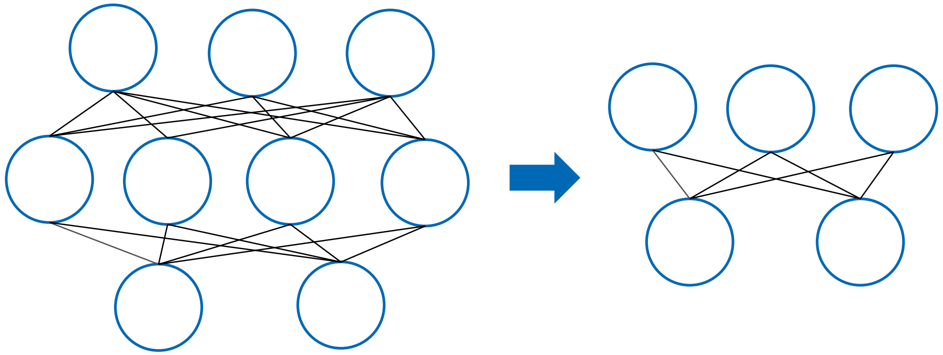
Quantization



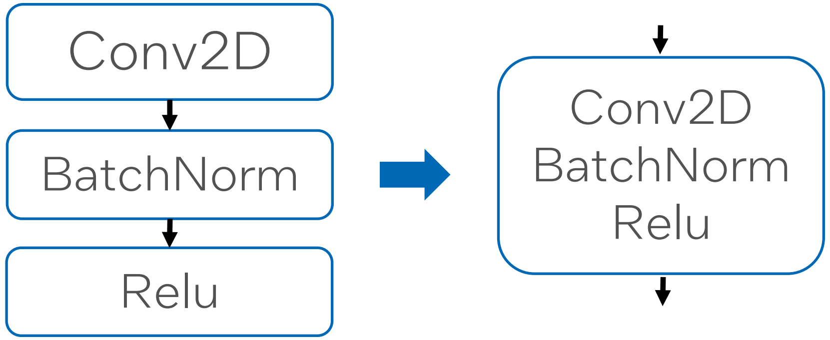
Pruning



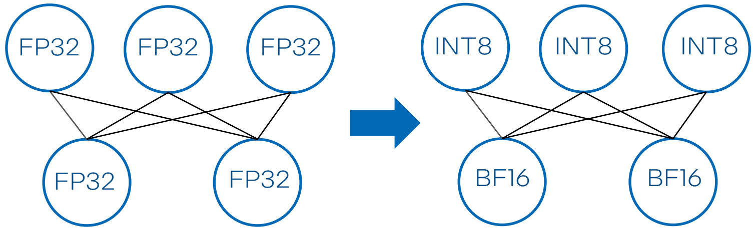
Knowledge Distillation



Graph Optimization

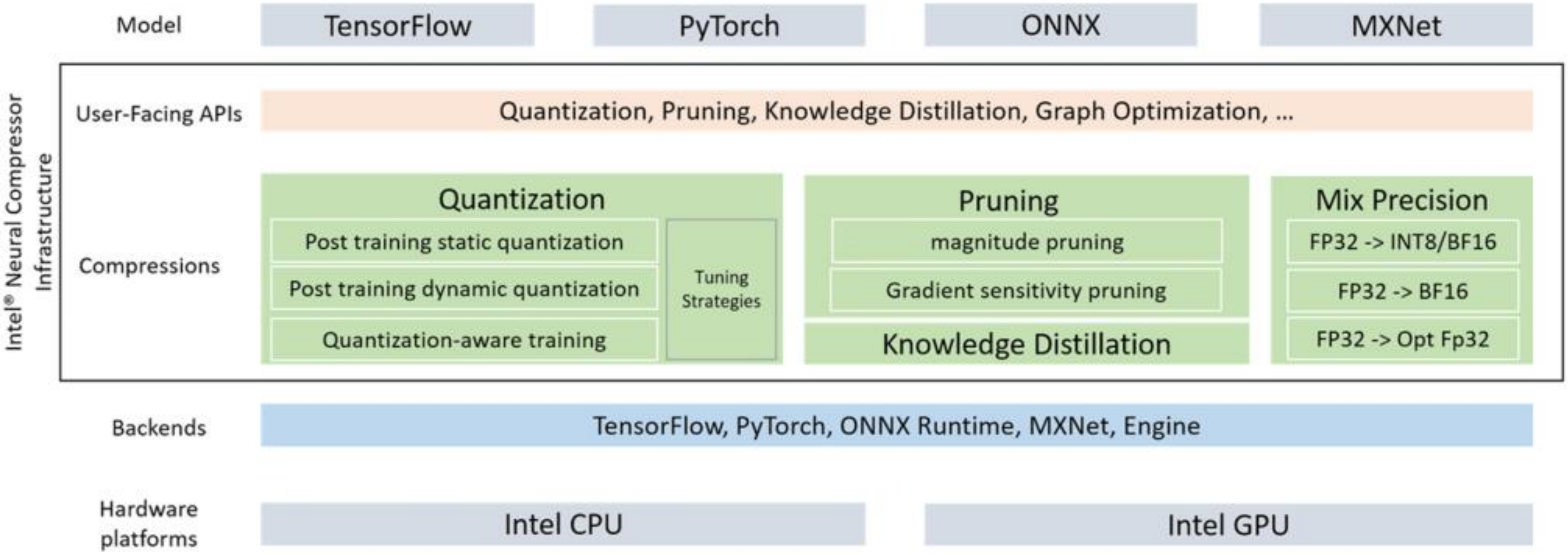


Mixed Precision Graph Optimization



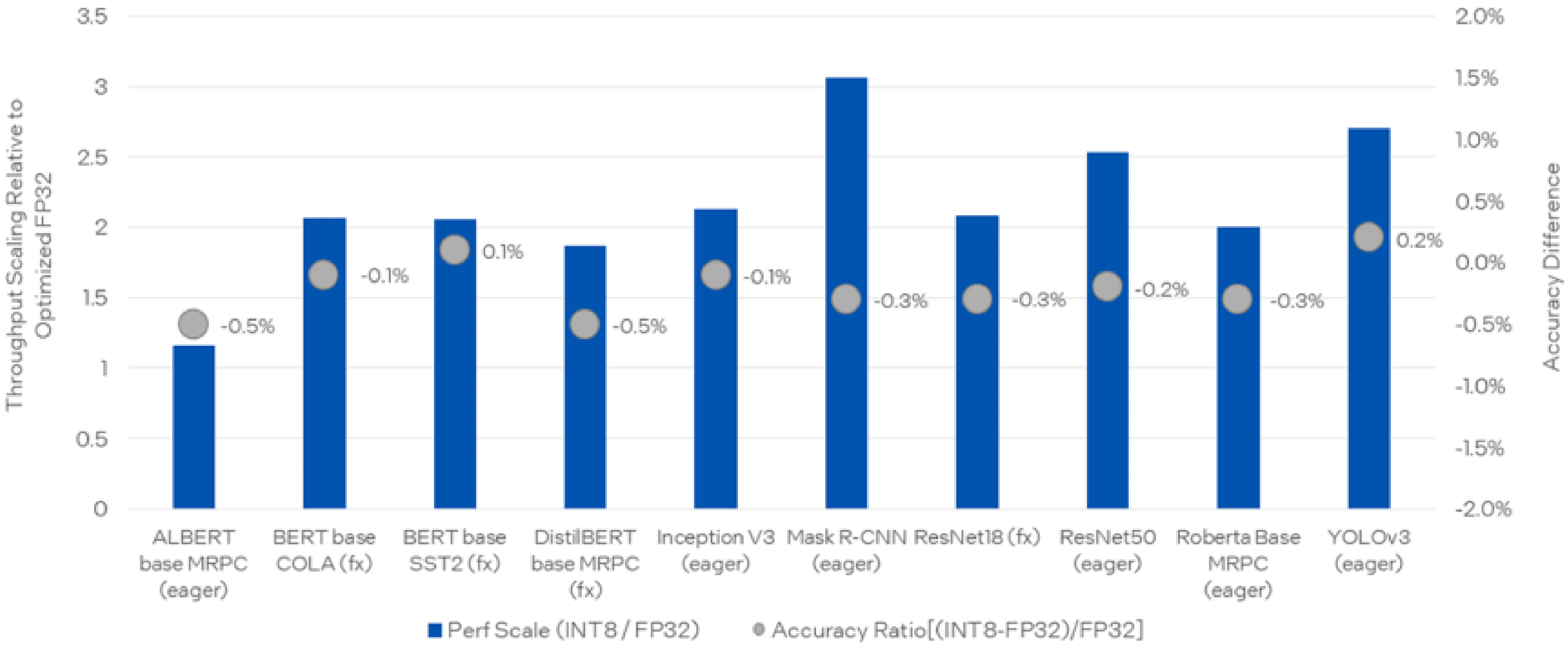
INC use automatic accuracy-driven tuning strategies to help user **easily & quickly** find out the best optimization methods above.

Intel® Neural Compressor Architecture



Performance vs Accuracy on INT8 Quantization with Intel® Neural Compressor

Post-Training Quantization and PyTorch* Inference



Testing Date: Performance results are **based on testing by Intel as of June 7, 2022** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: PyTorch* v1.11.0+cpu; Intel® Neural Compressor v1.12; Platform: Intel® Xeon® Platinum 8380 CPU @ 2.30GHz; 1 socket; 4 cores/instance; 10 instances; batch size = 1; Turbo: On; BIOS version: SE5C6200.86B.0022.D64.2105220049; System DDR Mem Config: 256GB (16x16GB DDR4 3200MT/s [3200MT/s]); OS: Ubuntu 20.04.1 LTS; Kernel: 5.4.0-42-generic. Full benchmark results available on Intel® Neural Compressor GitHub* (https://github.com/intel/neural-compressor/blob/master/docs/validated_model_list.md).

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex. Your costs and results may vary.

Getting Intel® Neural Compressor

- Intel® Neural Compressor is included in the **Intel® AI Analytics Toolkit (AI Kit):**
  - https://www.intel.com/content/www/us/en/developer/tools/oneapi/ai-analytics-toolkit-download.html?operatingsystem=linux
- Download the Stand-Alone Version:
  - https://intel.github.io/neural-compressor/latest/docs/source/Welcome.html#installation
- Use Intel® Developer Cloud:
  - https://www.intel.com/content/www/us/en/secure/forms/devcloud/enrollment.html?tgt=www.intel.com/content/www/us/en/secure/forms/devcloud-enrollment/account-provisioning.html

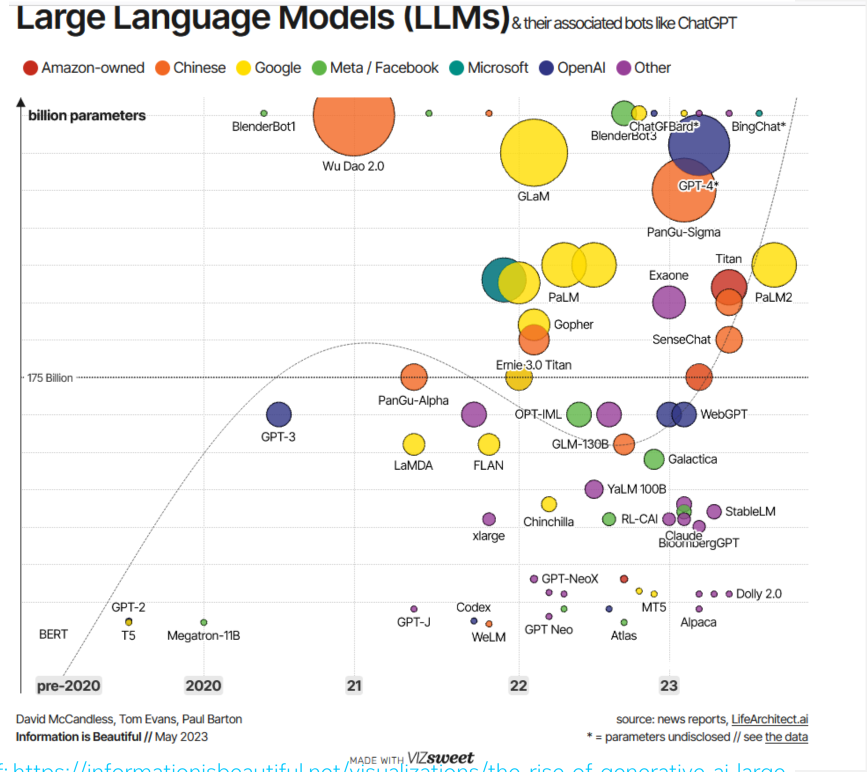
Intel® Optimization for LLMs

Intel® Extension esp. for LLMs:

Intel® Extension for Transformers:

https://github.com/intel/intel-extension-for-transformers

Intel® Extension for Transformers is an innovative toolkit to accelerate Transformer-based models on Intel platforms, in particular effective on 4th Intel Xeon Scalable processor Sapphire Rapids The toolkit provides many key features and examples such as **Stable Diffusion, GPT-J-6B, GPT-NEOX, BLOOM-176B, T5, Flan-T5 and end-to-end workflows such as SetFit-based text classification and document level sentiment analysis (DLSA)**



[Ref: https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/](https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/)

Intel® Extension for DeepSpeed:

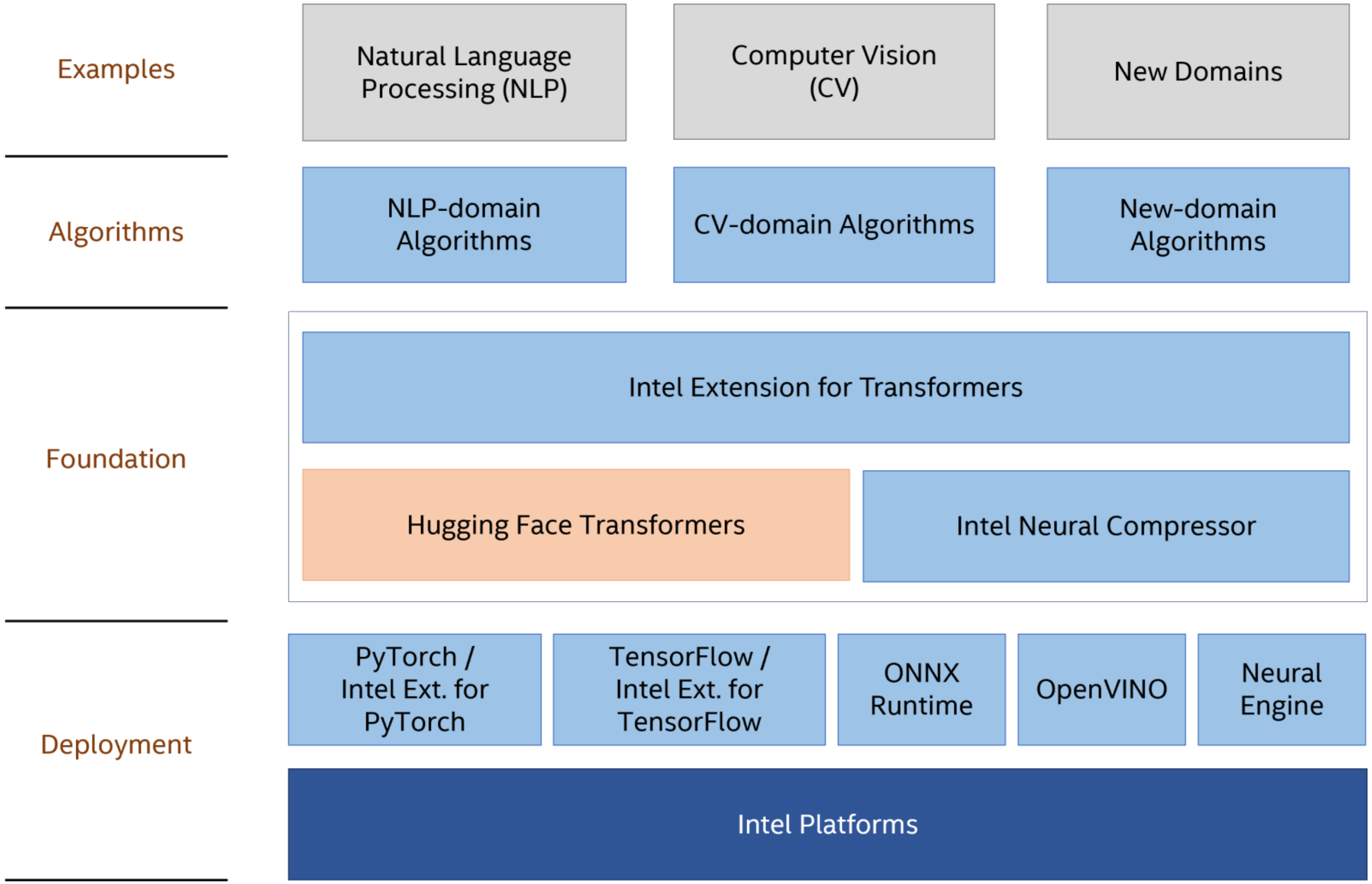
https://github.com/intel/intel-extension-for-deepspeed

- Deep learning optimization software suite that enables scale and speed for Deep Learning Training and inference
  - Train/inference models with billions or trillions of parameters
  - Efficiently scale to thousands of computing units
  - Train/inference on GPU system with limited GPU memory
  - Low latency and high throughput for inference

Intel® Extension for Transformers – Overview

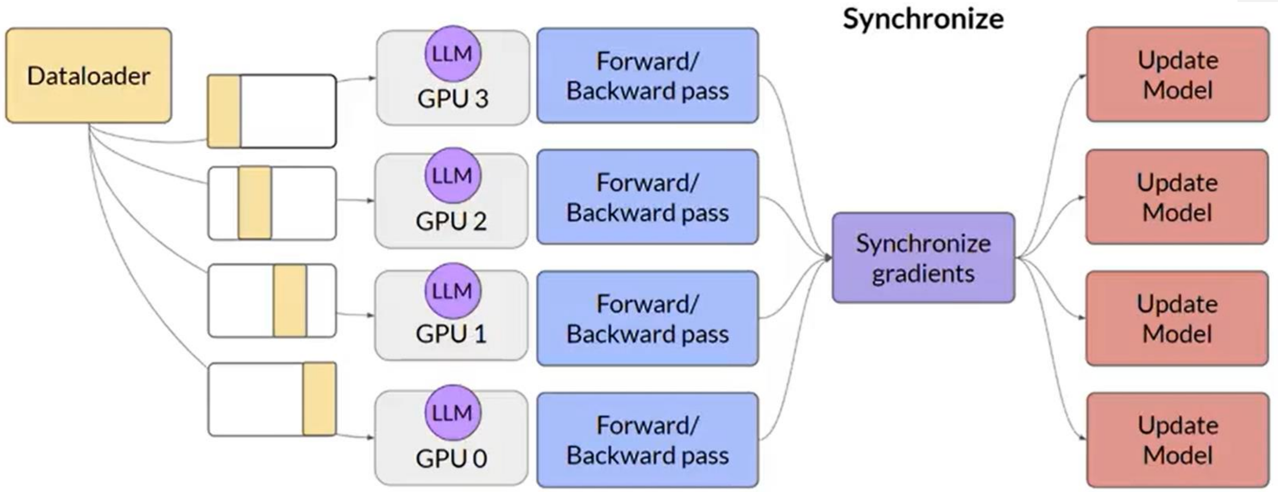
- Intel® Extension for Transformers (ITREX): Built on top of INC ecosystem and HF
- Its target is the democratization of NLP and Transformers for both training/fine-tuning and inference
- Extension to HF’s Transformers and Optimum
- Staging area for all Intel’s transformer feature enhancements:
  - Upstream to HF as much as possible (Transformers + Optimum)
  - Intel’s differentiation remains, e.g., NAS, MoE, dynamic model, etc., and is ready for future upstream

Intel® Extension for Transformers – Architecture



DeepSpeed Training technology

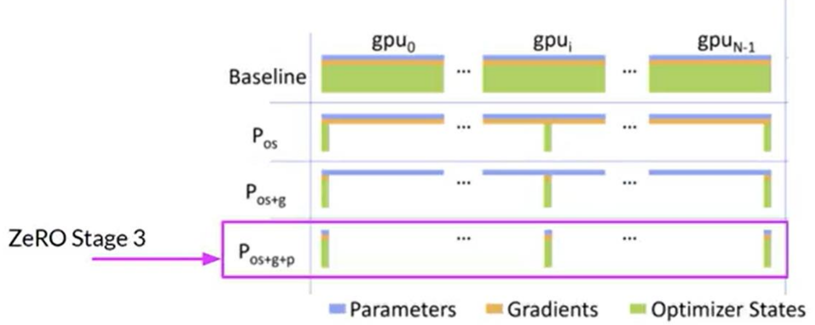
- When to use distributed compute
  - Model too big for single GPU
  - Model fits on GPU, but for faster training, train data in parallel
- DDP (Distributed Data Parallel) (Pytorch)
  - It requires all your training params should fit into 1 GPU
- If your model is too bigger to fit on single GPU:
  - Model Sharding
    - Zero (Zero data overlap between GPUs)
    - Zero (Memory optimizations toward training trillion parameter models)



Zero Redundancy Optimizer (ZeRO)

- Reduces memory by distributing (sharding) the model parameters, gradients, and optimizer states across GPUs

Sharding with stage 3 could reduce ur memory usage by the factor of num of GPUS used.

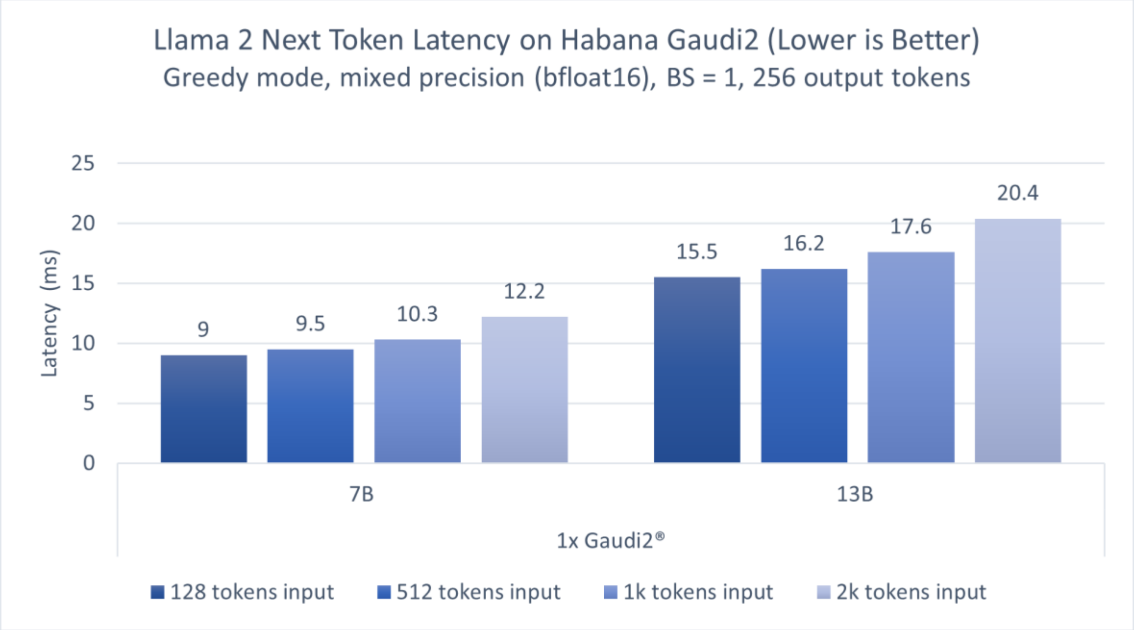


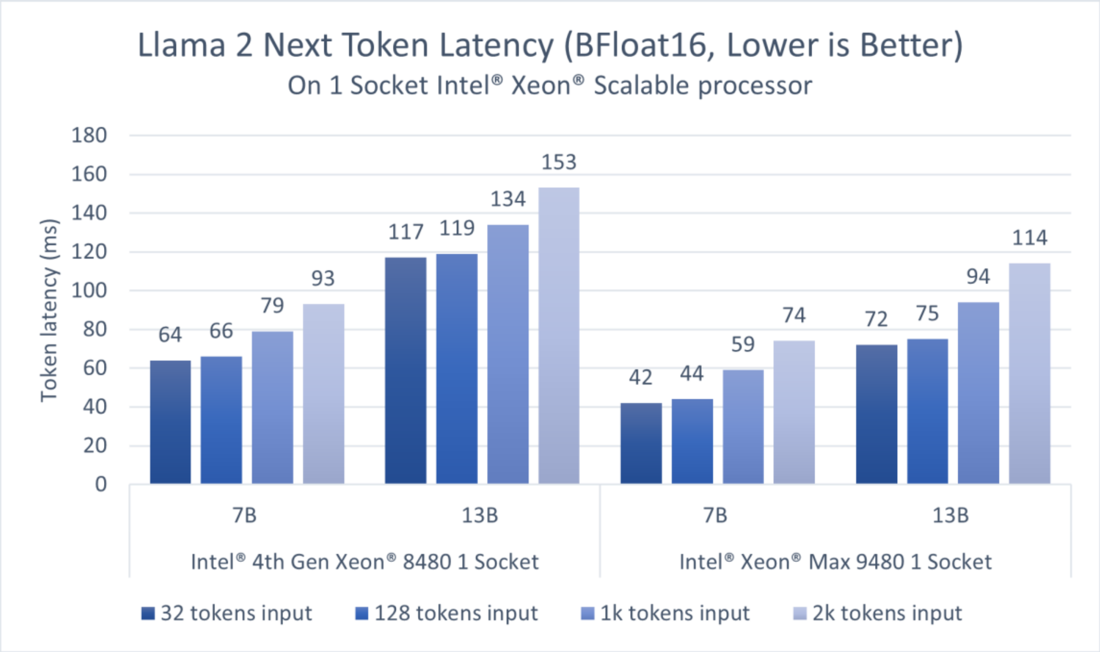
Ref: https://www.coursera.org/learn/generative-ai-with-llms/

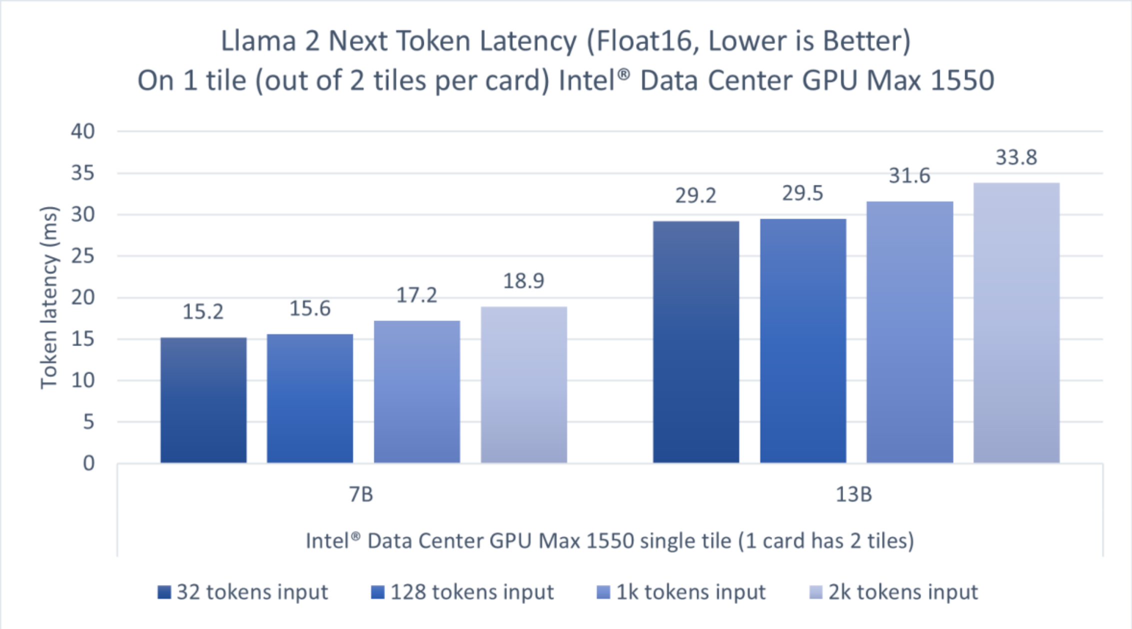
Building LLM Solutions

- Use dev branch before optimizations are provided in release branches.
  - https://intel.github.io/intel-extension-for-pytorch/llm/cpu/
  - https://intel.github.io/intel-extension-for-pytorch/llm/xpu/
- Models fitting one tile: PyTorch/ IPEX, similar experience as PyTorch CUDA
- Models not fitting one tile: PyTorch/IPEX, DeepSpeed/IDEX, similar experience as PyTorch CUDA + DeepSpeed
- Inference (16bit, 8bit, 4bit), fine-tune, pre-train

Llama 2 Inference Performance







Gaudi2 has demonstrated excellent training performance on large language models on the recently published MLPerf benchmark for training the 175 billion parameter GPT-3 model on 384 Gaudi2 accelerators

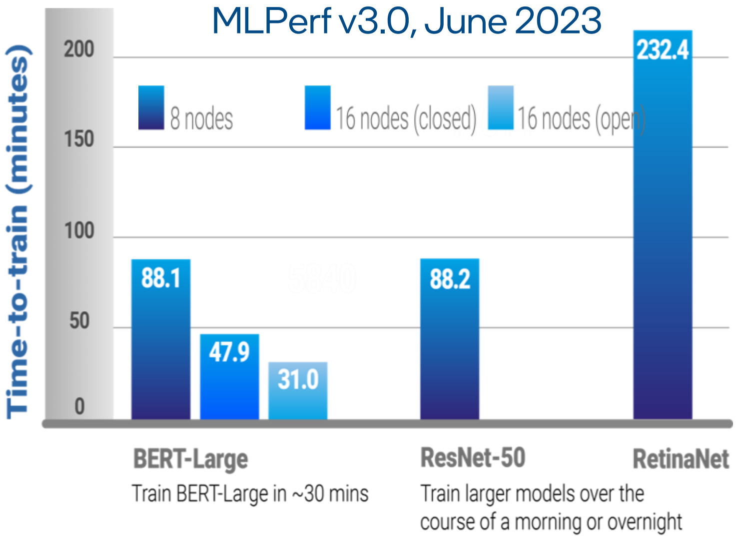
One 4th Gen Xeon socket delivers latencies under 100ms with 7 billon parameter and 13 billon parameter size of models. Users can run 2 parallel instances, one on each socket, for higher throughput and to serve clients independently

Intel Data Center GPU Max: Users can run 2 parallel instances, one on each tile, for higher throughput and to serve clients independently.

Ref: https://www.intel.com/content/www/us/en/developer/articles/news/llama2.html

XeonAI Performance

Training: TTT



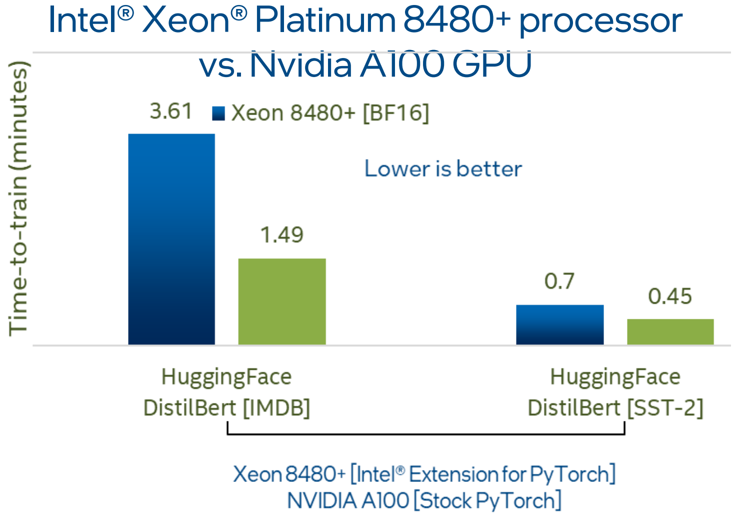
Only CPU submitted

Train models from scratch in hours

Multi-node Intel Ethernet scaling at > 97%

Great for intermittent training using standard industry frameworks

Fine Turning: TTT



Fine tune in <1 day for $100's

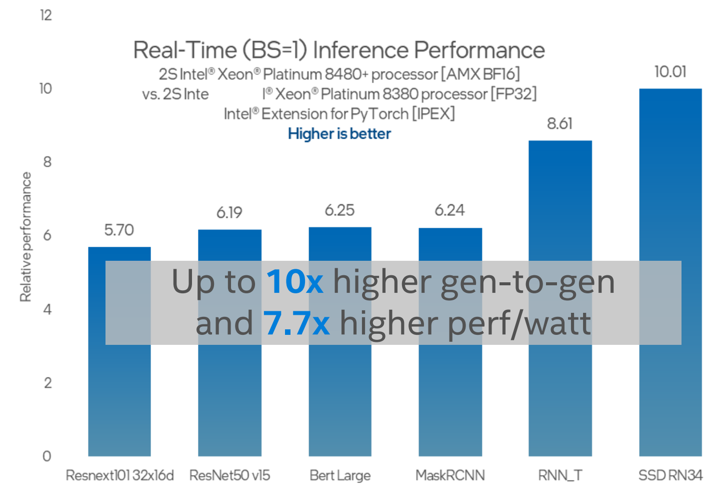
Small models: <1 minutes

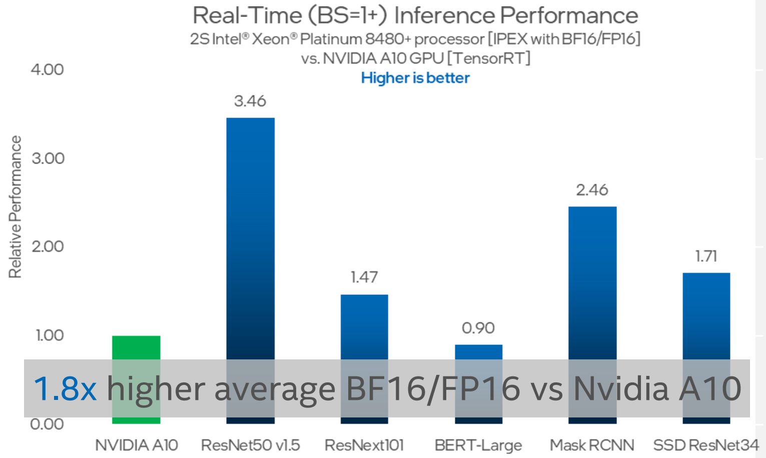
Medium sized models: <5minutes

Larger models: ~5m across multi nodes

- Stable Diffusion Few-Shot fine tuning across 4 XeonSP nodes: 4minutes
- All Intel SW optimizations default and automated in HuggingFace Framework

Inference



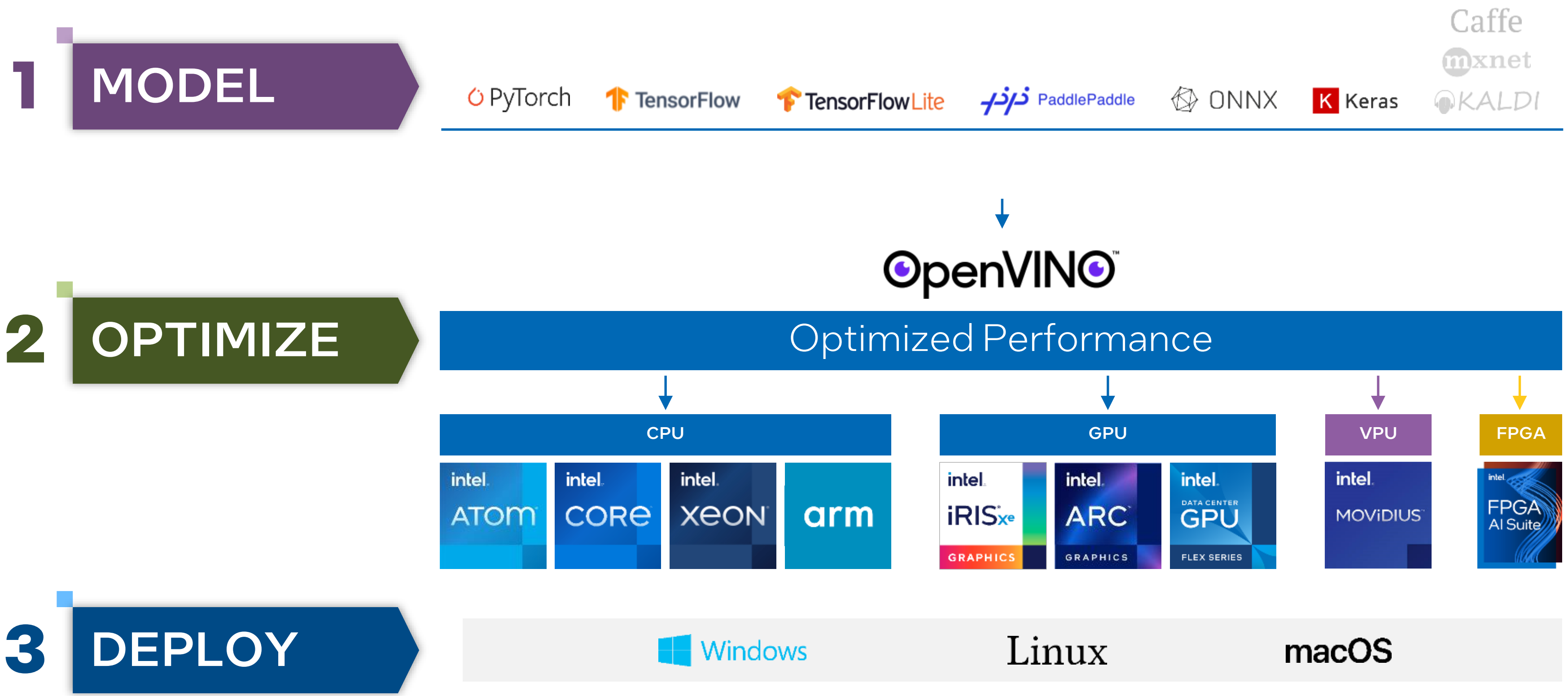


https://huggingface.co/spaces/Intel/Stable-Diffusion-Side-by-Side

OpenVINO™

OpenVINO™ Toolkit Overview

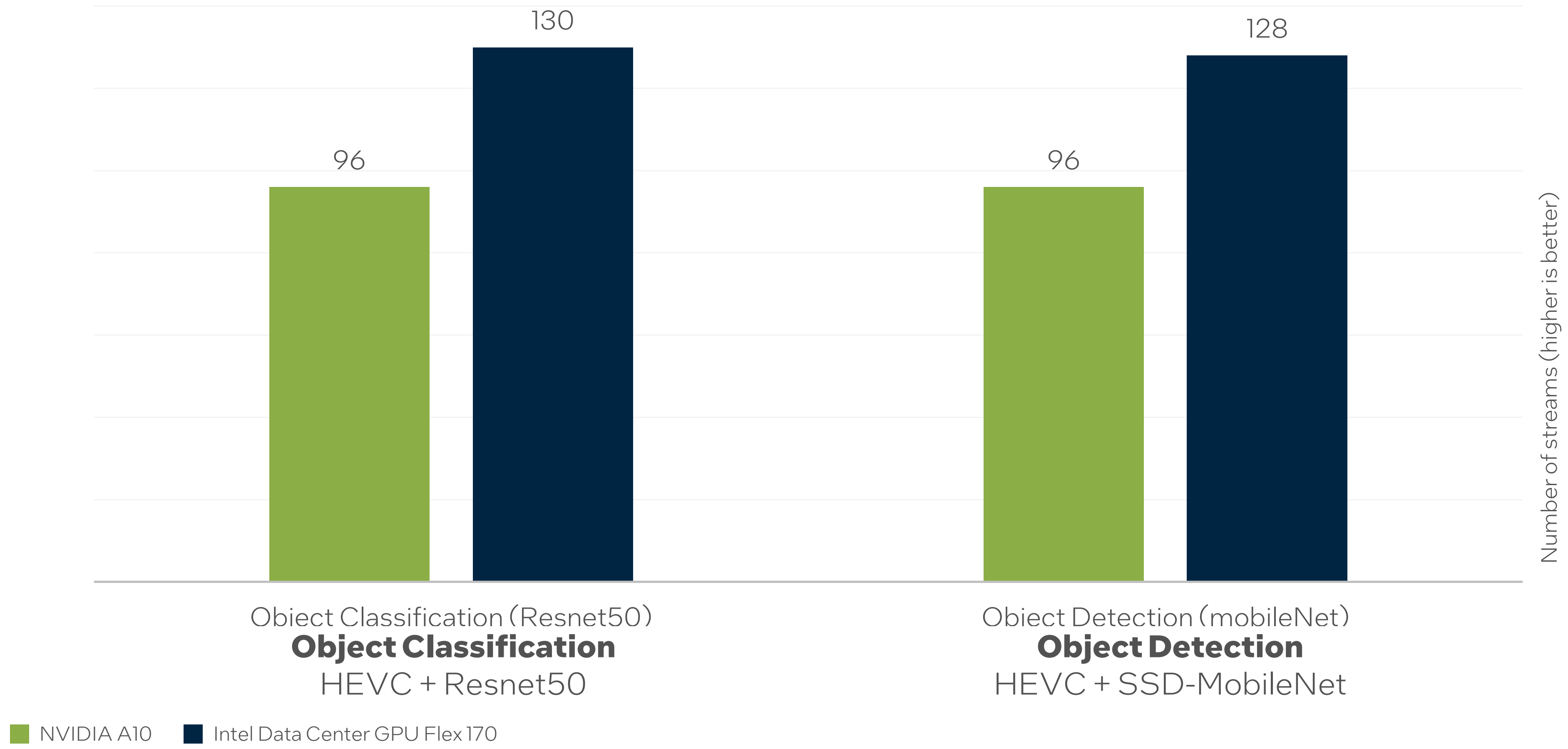
Convert and optimize models, and deploy across a mix of hardware and environments, on-premises and on-device, in the browser or in the cloud



Powered by oneAPI

The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary alternatives.

AI Visual Inference (Media + AI)



Based on OV 2022.2 For workloads and configurations visit www.Intel.com/PerformanceIndex. Click on the Events tab and Intel® Innovation 2022. Results may vary.
For workloads and configurations see this slide 7.

HEVC 1080p30 8 bit 3Mbps
Resnet50: 224x224; BS 64
SSD-MobileNet: 300x300; BS 64

Ready to get OpenVINO?

Choose and download free directly from Intel

Intel® Distribution of OpenVINO™ Toolkit

Also available from these sources:

Intel® Developer Cloud PIP Docker Hub
Dockerfile Anaconda Cloud YUM APT

Build from source:

GitHub | Gitee (for China)