Introduction to Neural Network Compression Techniques

LRZ AI Workshop

Dr. Nikolai Solmsdorf – AI Software Solutions Engineer
13.10.2022

intel®

Notices and 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.
- You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
- The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that code included in this document is licensed subject to the Zero-Clause BSD open source license (OBSD), http://opensource.org/licenses/0BSD.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.
-

Agenda

- Introduction
- Neural Network Compression Techniques
- Intel® Neural Compressor

Introduction

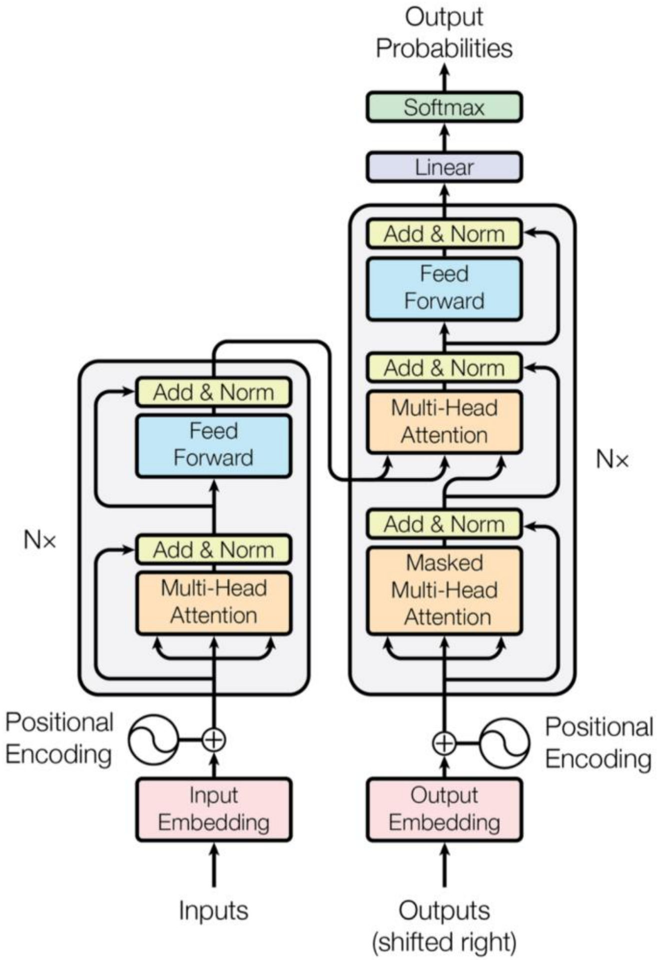
Introduction

? Need for neural network compression techniques?

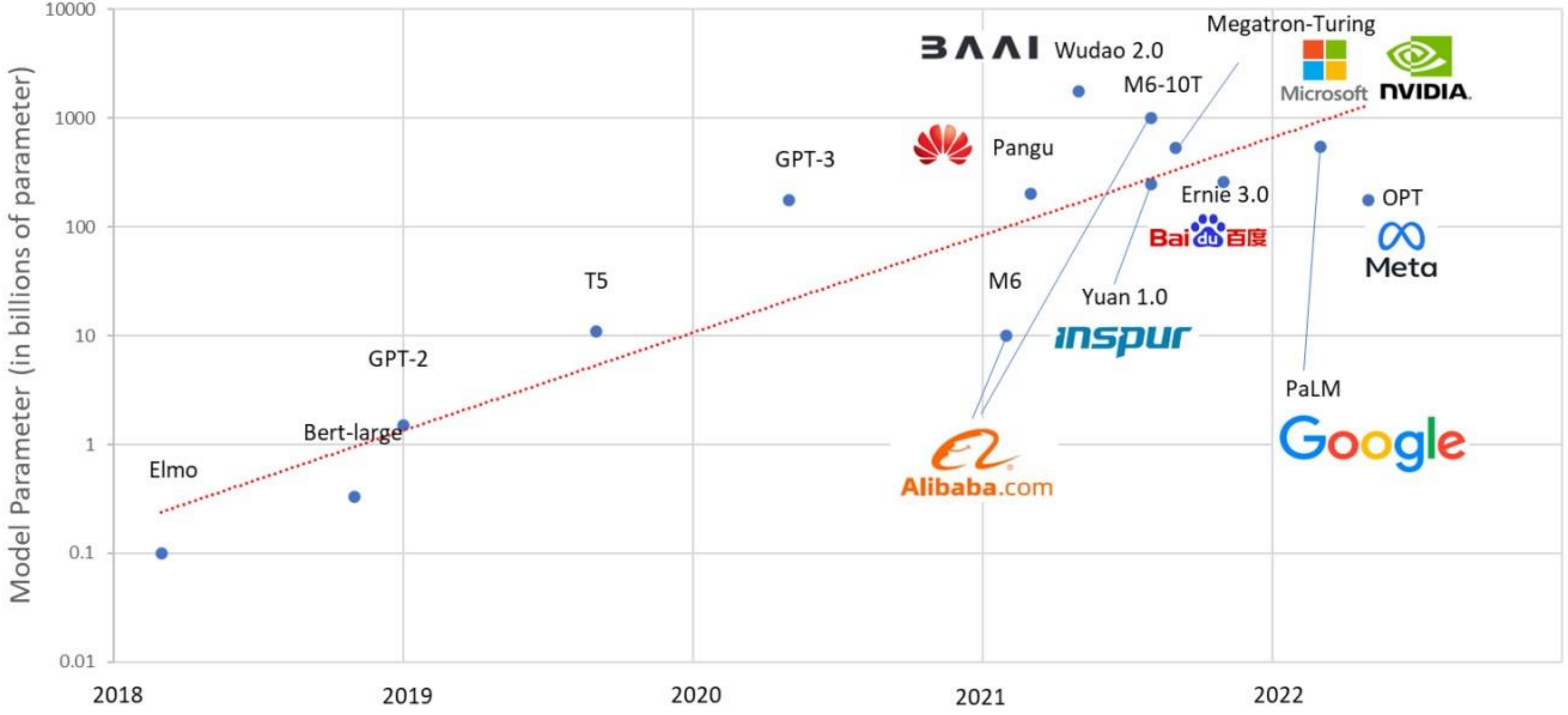
! Showcase: Transformer architectures!

Introduction – Transformers

- Transformer architecture originates from machine translation tasks (NLP)
- Core-component: Multi-head (self-) attention
- Parallelization
- Most commonly known: BERT
- Huge increase of model parameters over the years



Introduction – Transformers

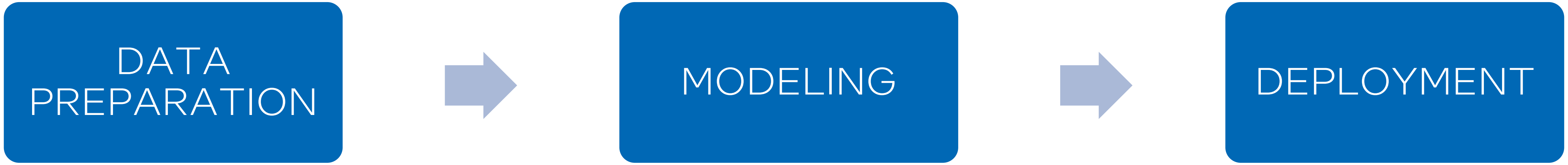


Introduction – Transformers (GPT-3)

- Large Language Model (LLM) by OpenAI (backed by Microsoft)
- 175B parameters (cf. BERT base: 110M)
- 45TB of training data
- Cf. GPT-NeoX:
  - Trained on 96 [!] Nvidia A100s
  - For 3 months
- Estimated cost for training: 10—20 million dollars

Introduction – Transformers

- Dominate NLP (via Hugging Face)
- BERT models (still) prevail
- **Optimization opportunities:**
  - Training/Finetuning
  - Inference
  - On a variety of hardware (CPU & GPU)
  - Aimed at speed-ups, memory reductions, sustainability.



Neural Network Compression Techniques

Neural Network Compression – Overview

- Precision
- Quantization
- Pruning
- Knowledge Distillation
- Graph Optimization
- Mixed Precision

Neural Network Compression Techniques

Precision

Floating Point – Precision

- Data precision:

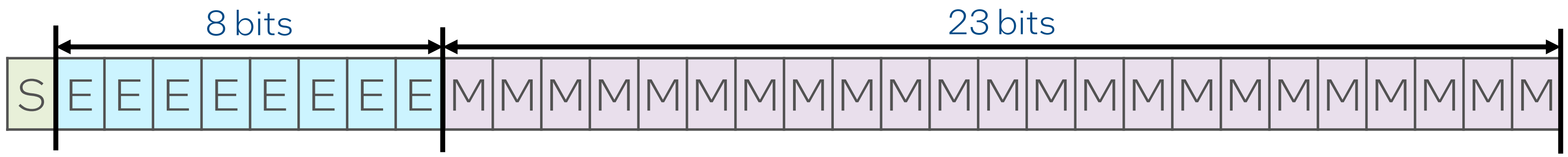
  Number of bits used to store numerical values in memory

- Commonly found types of precision in Deep Learning:

INT8	BF16 - FP16	TF32	FP32
↓	↓	↓	↓
8 bits	16 bits	19 bits	32 bits

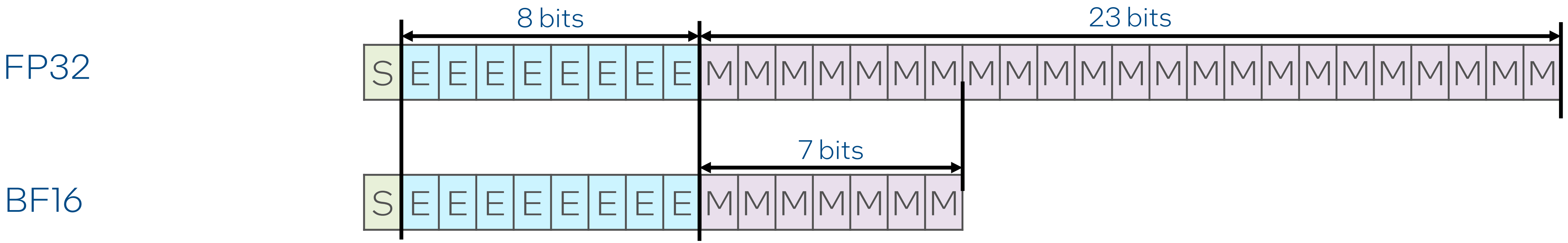
Floating Point – Precision

FP32



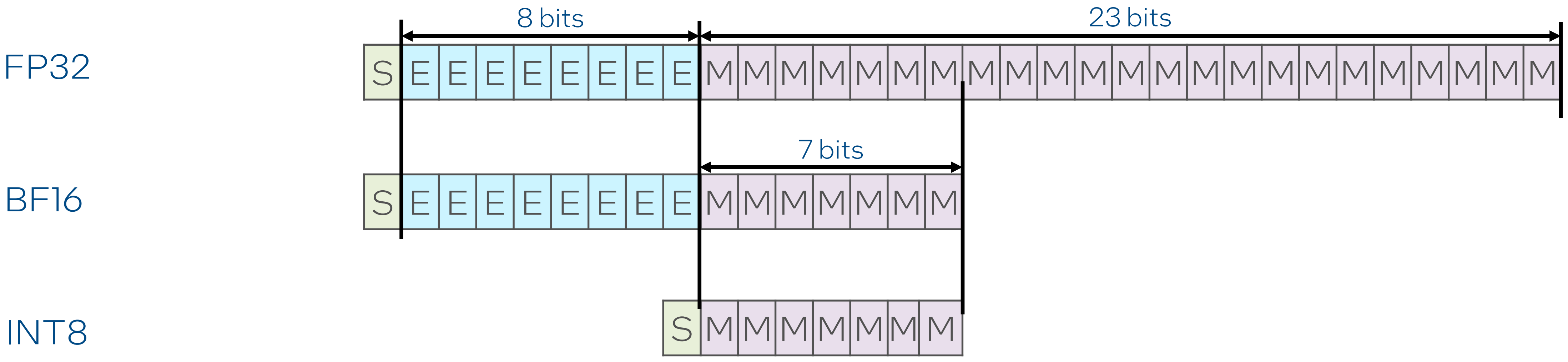
- FP32: The standard type for all neural network computations

Floating Point – Precision



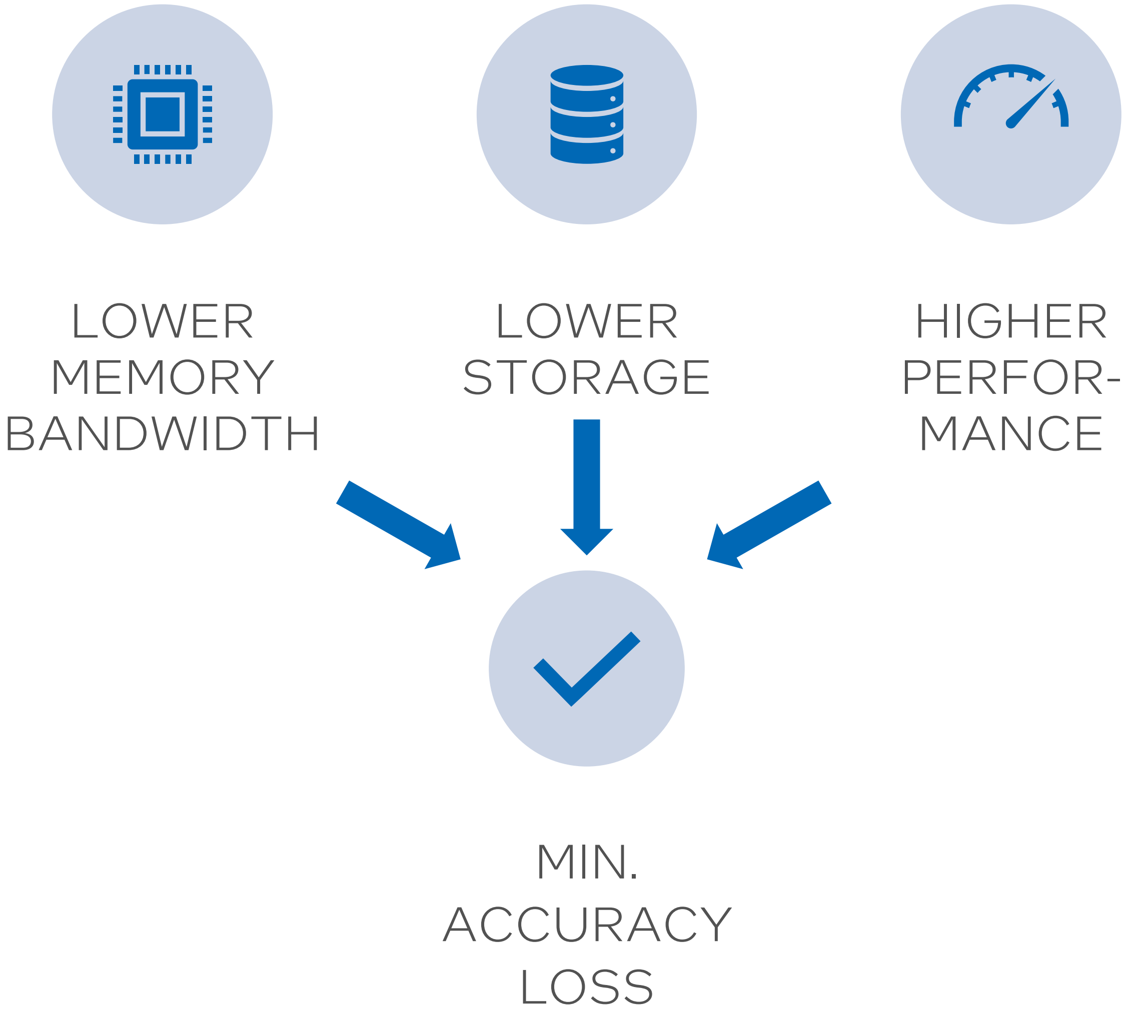
- FP32: The standard type for all neural network computations
- BF16: Efficient replacement for FP32 in training and inference

Floating Point – Precision



- FP32: The standard type for all neural network computations
- BF16: Efficient replacement for FP32 in training and inference
- INT8: Significant speed-up in inference with small loss in accuracy

Lower Precision – Summary

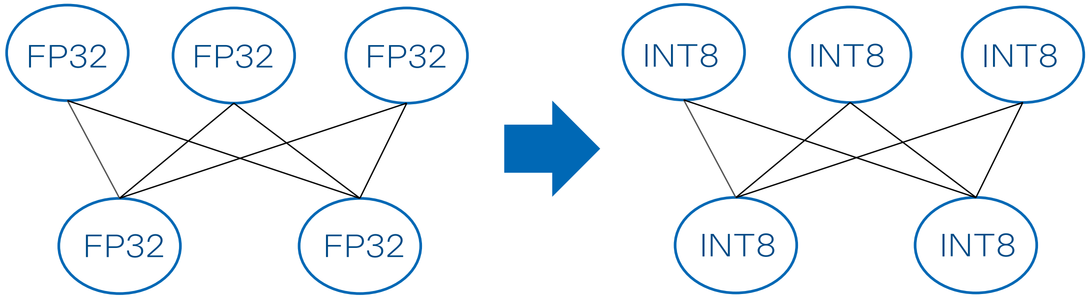


Neural Network Compression Techniques

Quantization

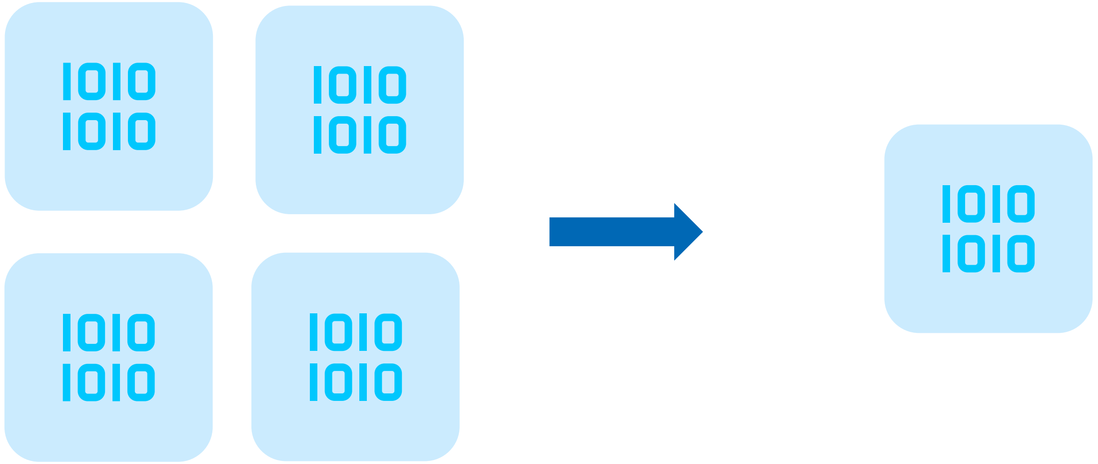
Quantization

- Systematic reduction of the precision of all or several layers within the model.
- Effectively, reduces the model size.
- Allows for faster inference.



Quantization

FP32	→	INT8
96.1924		96
32 bits		8 bits



Quantization – Techniques (PTQ)

- **Post-Training Static Quantization (PTQ static):**
  - Performs quantization on already trained models.
  - It requires an additional pass over a dataset to work, only activations do calibration.
- **Post-Training Dynamic Quantization (PTQ dynamic):**
  - Multiplies input values by the scale factor, then rounds the result to the nearest.
  - It determines the scale factor for activations dynamically based on the data range observed at runtime.
  - Weights are quantized ahead of time, but the activations are dynamically quantized during inference.

Quantization – Techniques (QAT)

- **Quantization-Aware Training (QAT):** Simulates low-precision inference-time computation in the forward pass of the training process.
  - All weights and activations are “fake quantized” during both the training forward and backward passes
  - FP32 values are rounded to mimic INT8 values, but all computations are still done with floating point numbers.
  - Weight are trained while being “aware” of the model will be quantized in the end.
  - Leads to higher accuracy than previous 2 quantization techniques but may take more time to deployment.

Quantization – Summary

Systematic reduction of the model precision

Common procedure: FP32 => INT8

Common techniques: PTQ (static, dynamic) & QAT

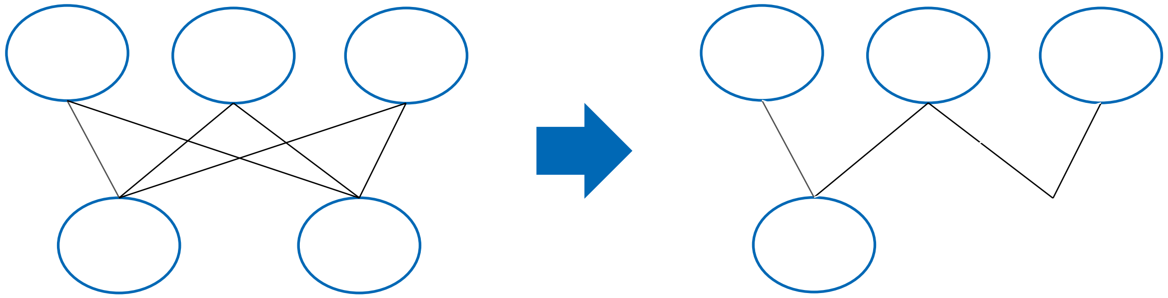
Faster inference & possible loss in accuracy

Neural Network Compression Techniques

Pruning

Pruning

- **Neural Network Pruning:** Reduces the size of a network by removing superfluous parts to achieve compact architectures with a minimal drop in accuracy.



Pruning – Techniques

- Unstructured Pruning
  - Finding and removing the least salient connections in the model where the nonzero patterns are irregular and could be anywhere in the matrix.
  - Superfluous weight values are reduced to zero.
  - Network architecture is unchanged, but induces sparsity in the network.
- Structured Pruning
  - Finding parameters in groups, deleting entire blocks, filters, or channels according to some pruning criteria.
  - Possibly, reduces width of layers and network.

Pruning – Techniques

Pruning Type	Pruning Granularity	Pruning Algorithm
Unstructured Pruning	Element-wise	Magnitude
		Pattern Lock
Structured Pruning	Filter/Channel-wise	Gradient Sensitivity
	Block-wise	Group Lasso
	Element-wise	Pattern Lock

Pruning – Summary

Remove superfluous parts in NN

Unstructured Pruning

Structured Pruning

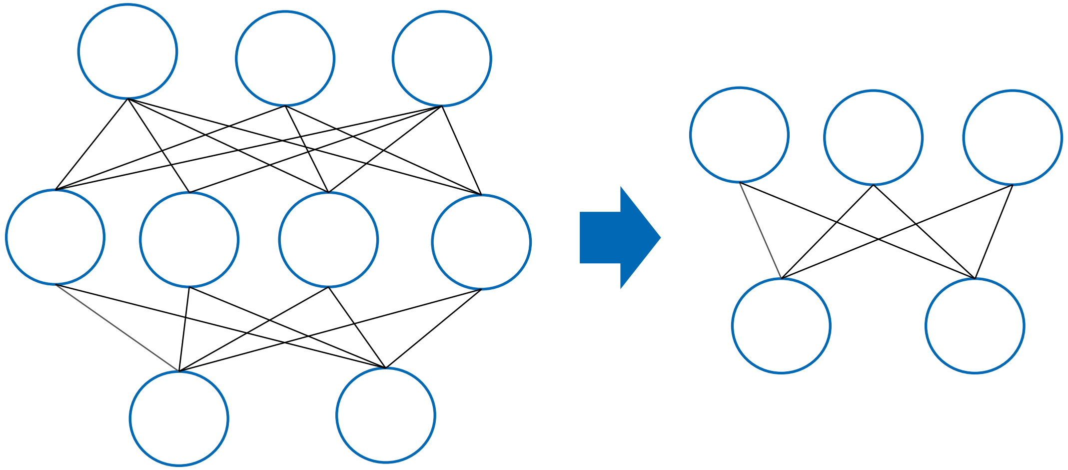
Faster inference & possible loss in accuracy

Neural Network Compression Techniques

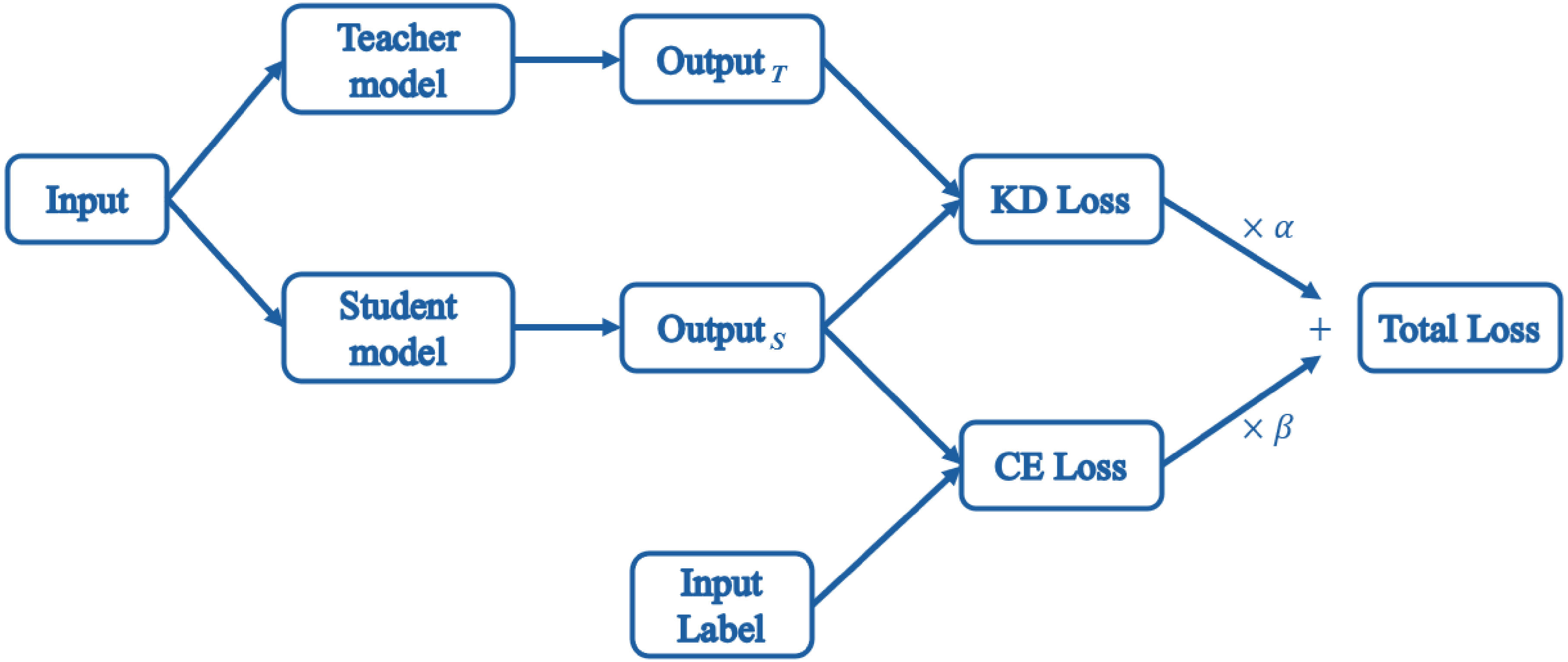
Knowledge Distillation

Knowledge Distillation

- A model compression method in which a small model is trained to mimic a pre-trained, larger model.
- The training setting is sometimes referred to as "teacher-student”.
- Transfers knowledge from the teacher to the student without loss of validity.



Knowledge Distillation



Knowledge Distillation

- **Student model:**
  - Shallow version of the teacher with fewer layers and fewer neurons per layer.
  - Quantized version of the teacher.
  - Pruned version of the teacher.
  - Smaller, optimized network through neural architecture search (NAS).
- **E.g., DistilBERT:**
  - 40% smaller than the teacher model, i.e., BERT base.
  - 97% of performance retained in GLUE benchmark.
  - 60% faster at inference.

BERT BASE → DISTILBERT

Knowledge Distillation – Summary

Student NN mimics teacher NN

Student: Shallower version of teacher

Knowledge transfer through combined loss

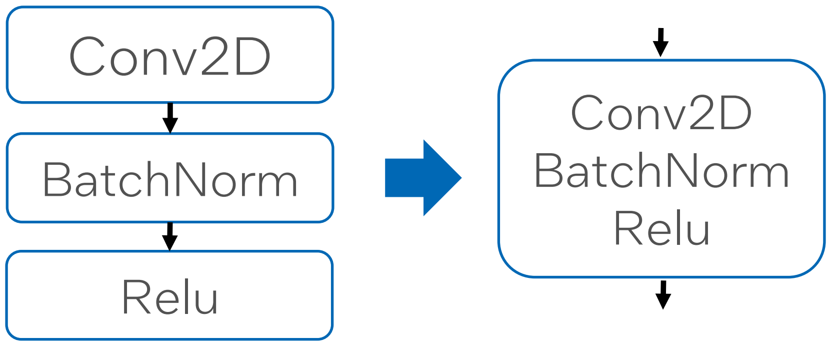
Faster fine-tuning & inference

Neural Network Compression Techniques

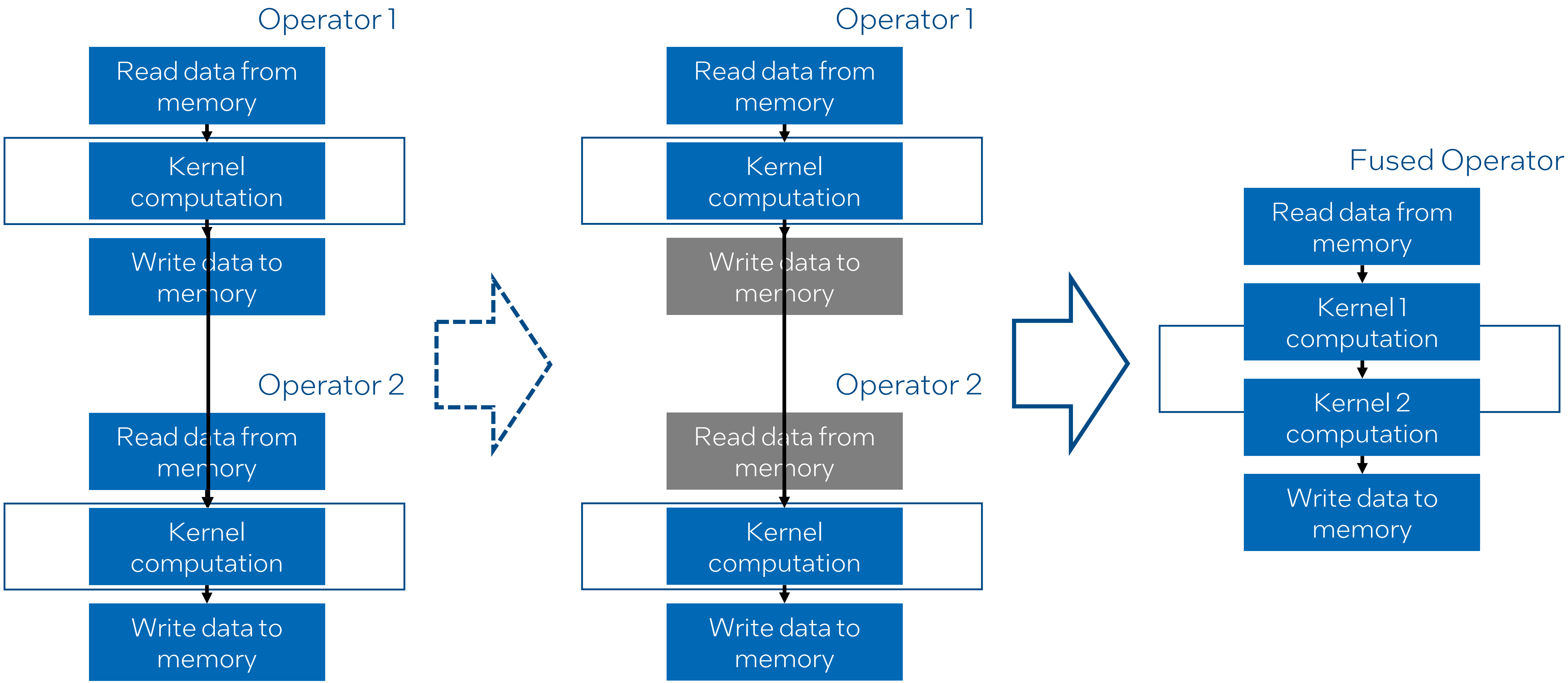
Graph Optimization

Graph Optimization

- Optimizes for inference of the model.
- Simplifies and fuses certain layers together.
  - Reduces complexity of the model.
- Optimizes graph (e.g., operators) and runtime (e.g., memory management).



Graph Optimization – Operator Fusion



Graph Optimization – FP32 & BF16 Fusion Patterns

- Conv2D + ReLU
- Conv2D + SUM
- Conv2D + SUM + ReLU
- Conv2D + Sigmoid
- Conv2D + Sigmoid + MUL
- Conv2D + HardTanh
- Conv2D + SiLU
- Conv2D + ELU
- Conv3D + ReLU
- Conv3D + SUM
- Conv3D + SUM + ReLU
- Conv3D + SiLU
- Linear + ReLU
- Linear + GELU
- Add + LayerNorm
- Div + Add + Softmax
- Linear + Linear + Linear
- View + Transpose + Contiguous + View

Graph Optimization – INT8 Fusion Patterns

- INT8 IN -> FP32 OUT
  - dequant -> conv
  - dequant -> linear
  - dequant -> conv -> relu
  - dequant -> conv -> sum
  - dequant -> conv -> sum -> relu
  - dequant -> linear -> relu
  - dequant -> linear -> gelu
  - dequant -> linear -> sigmoid
  - dequant -> linear -> sum
  - dequant -> bmm
  - dequant -> bmm -> div
- INT8 IN -> INT8 OUT
  - dequant -> conv -> quant
  - dequant -> linear -> quant
  - dequant -> conv -> relu -> quant
  - dequant -> conv -> sum -> dequant
  - dequant -> conv -> sum -> relu -> quant
  - dequant -> linear -> relu -> quant
  - dequant -> linear -> gelu -> quant
  - dequant -> linear -> sigmoid -> quant
  - dequant -> linear -> sum -> quant
  - dequant -> bmm -> quant
  - dequant -> bmm -> div -> quant
  - dequant -> max_pool2d -> quant

Graph Optimization – Summary

Neural network as DAG

Operator& memory optimization

E.g., operator fusion

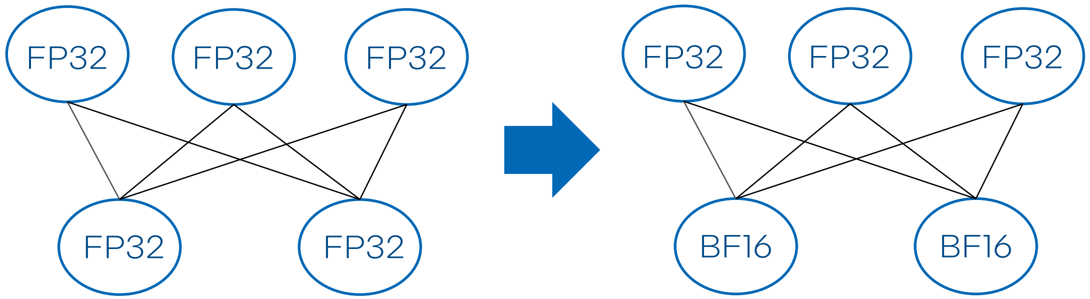
Faster inference

Neural Network Compression Techniques

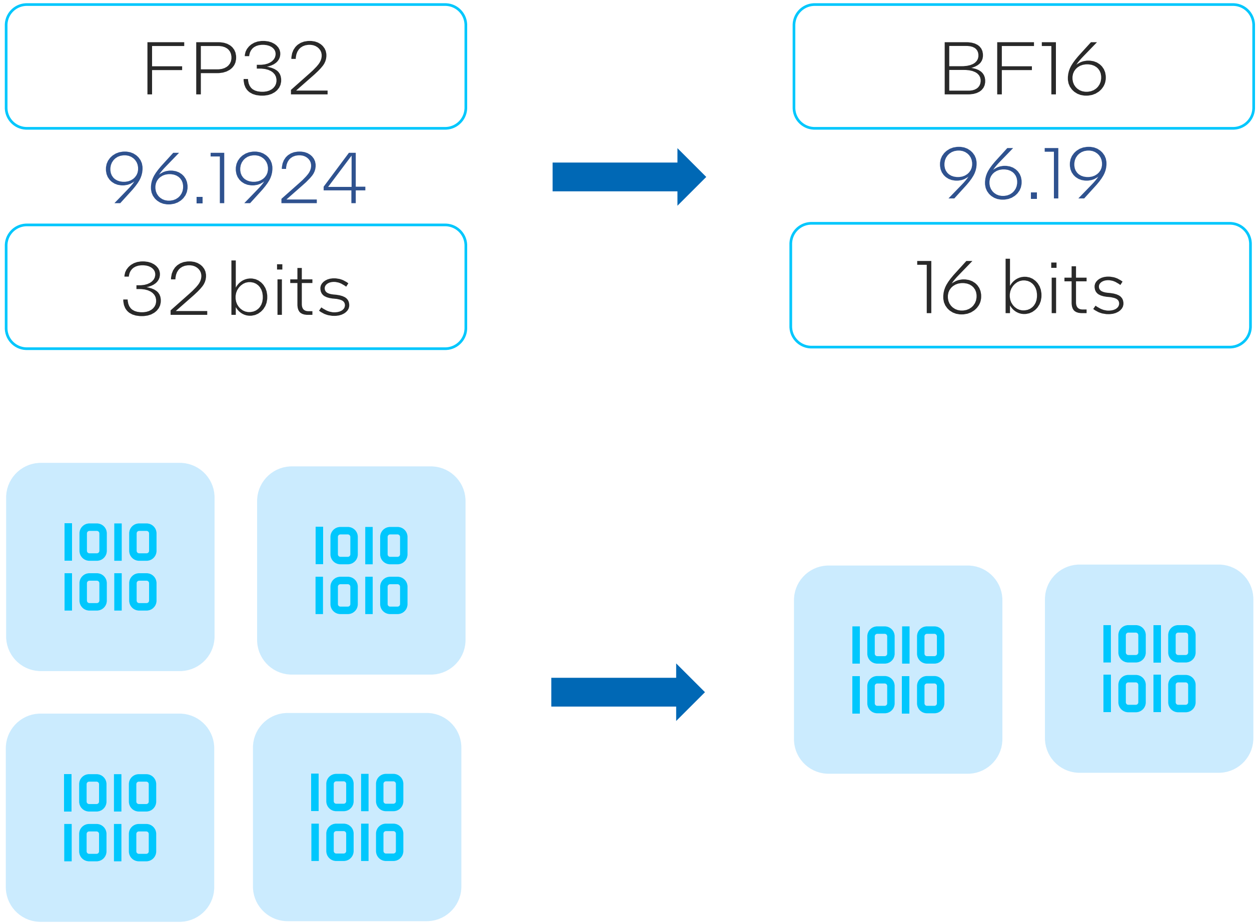
Mixed Precision

Mixed Precision

- Combination of different numerical formats in one computational workload.
- Requires less memory.
- Allows for faster training and inference.

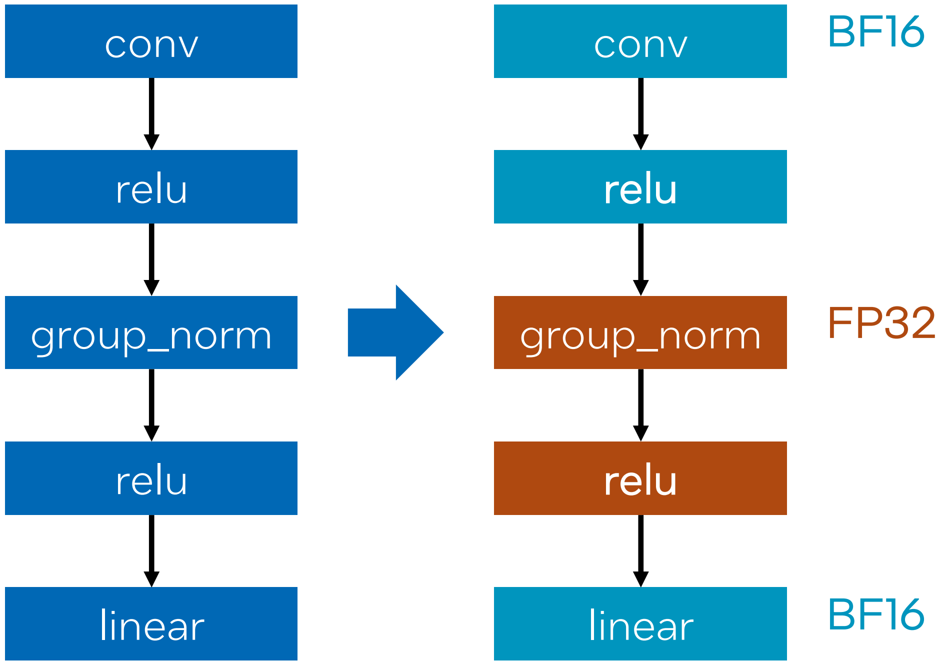


Mixed Precision



Auto Mixed Precision (AMP)

- 3 Categories of operators
  - Lower precision
    - Computation bound operators that could get performance boost with BF16.
    - E.g.: conv, linear
  - Fallthrough
    - Operators that runs with both FP32 and BF16 but might not get performance boost with BF16.
    - E.g.: relu, max_pool2d
  - FP32
    - Operators that are not enabled with BF16 support yet. Inputs of them are casted into FP32 before execution.
    - E.g.: max_pool3d, group_norm



Mixed Precision – Summary

Combination of precision types

Common: FP32 & BF16

E.g., AMP

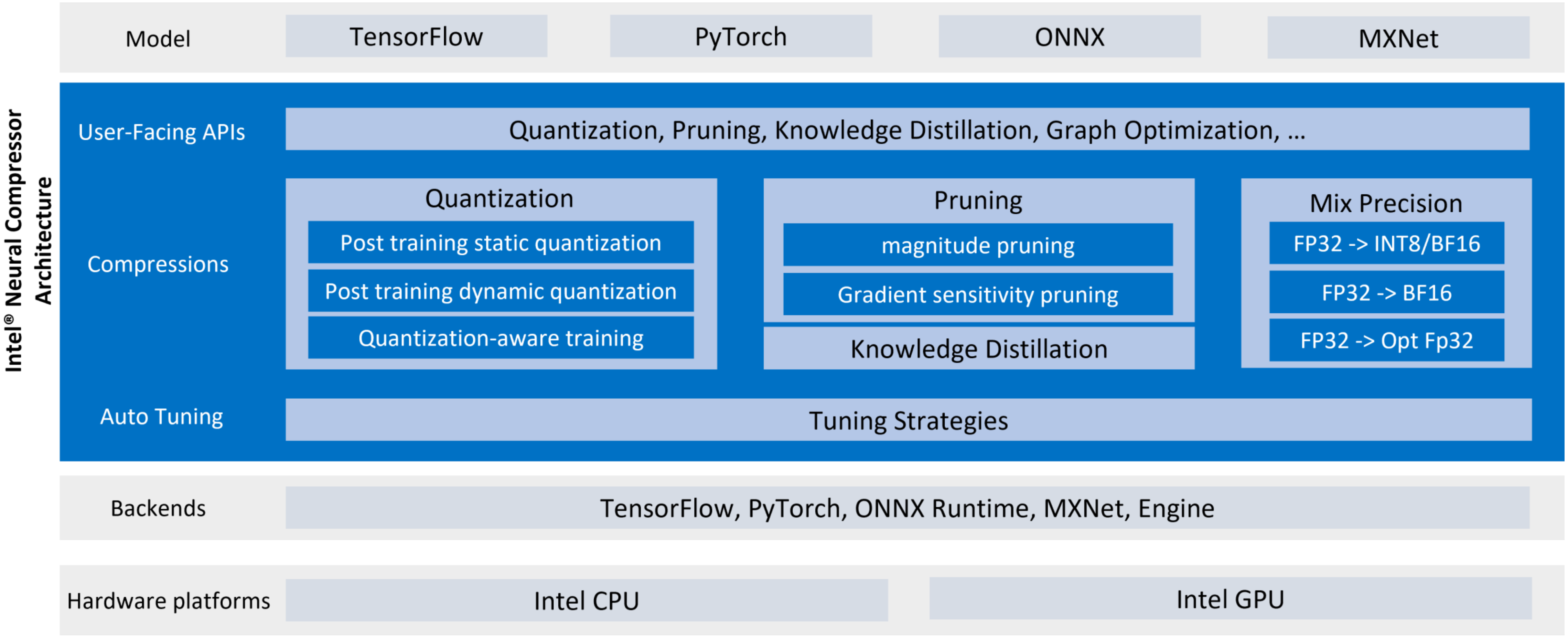
Faster training & inference

Intel Neural Compressor (INC)

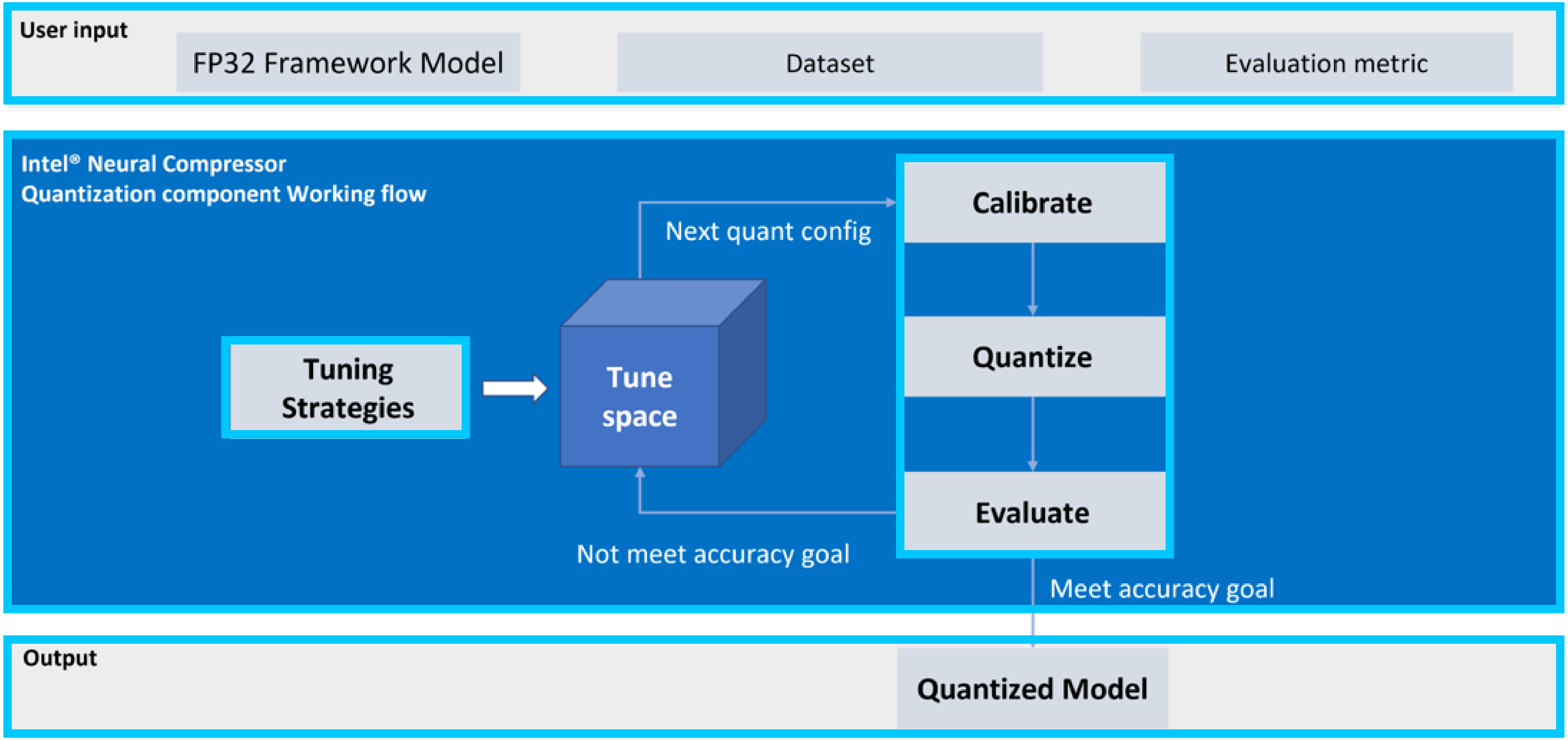
INC: Intel Neural Compressor

The Intel Neural Compressor, is an open-source Python library, which delivers unified interfaces across multiple deep learning frameworks for popular network optimization technologies.

It supports quantization, mixed precision, pruning, knowledge distillation, and graph optimizations, and uses accelerations by Intel Deep Learning Boost or Intel Advanced Matrix Extension in SPR.


Intel® Neural Compressor Architecture
Model
TensorFlow
PyTorch
ONNX
MXNet
User-Facing APIs
Quantization, Pruning, Knowledge Distillation, Graph Optimization, …
Compressions
Quantization
Post training static quantization
Post training dynamic quantization
Quantization-aware training
Pruning
magnitude pruning
Gradient sensitivity pruning
Knowledge Distillation
Mix Precision
FP32 -> INT8/BF16
FP32 -> BF16
FP32 -> Opt Fp32
Auto Tuning
Tuning Strategies
Backends
TensorFlow, PyTorch, ONNX Runtime, MXNet, Engine
Hardware platforms
Intel CPU
Intel GPU

INC: Workflow



Hugging Face

- Hugging Face 🤗 is an NLP-focused startup, which developed the popular Transformers library that exposes state-of-art transformer architectures to end-users.
- 2021 they released Optimum, a toolkit to compress Transformers, which also builds upon Intel Neural Compressor.

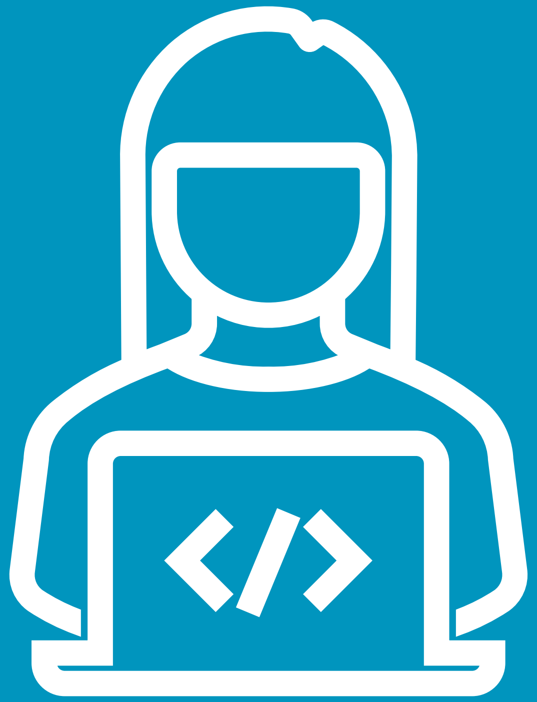
```python
from optimum.intel.neural_compressor.quantization import IncQuantizerForSequenceClassification

# Create quantizer from config
quantizer = IncQuantizerForSequenceClassification.from_config(
    "echarlaix/bert-base-dynamic-quant-test",
    config_name="quantization.yml",
    eval_func=eval_func,
)

# Apply dynamic quantization
model = quantizer.fit_dynamic()
```

INC: Intel Neural Compressor

INTEL'S GENERAL PURPOSE MODEL OPTIMIZATION TOOLS, INCL. QUANTIZATION, PRUNING, DISTILLATION, ETC.

COVERING CV, RECOMMENDER, NLP, ETC.

Demo



Thank you for your attention!

Questions?