Performance-Efficient E2E AI Pipelines

Meena Arunachalam, Principal Engineer, Artificial Intelligence and Analytics Group
Intel Corporation

13th International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures
In conjunction with VLDB 2022ADMS Workshop 2022

intel

Notices & Disclaimers

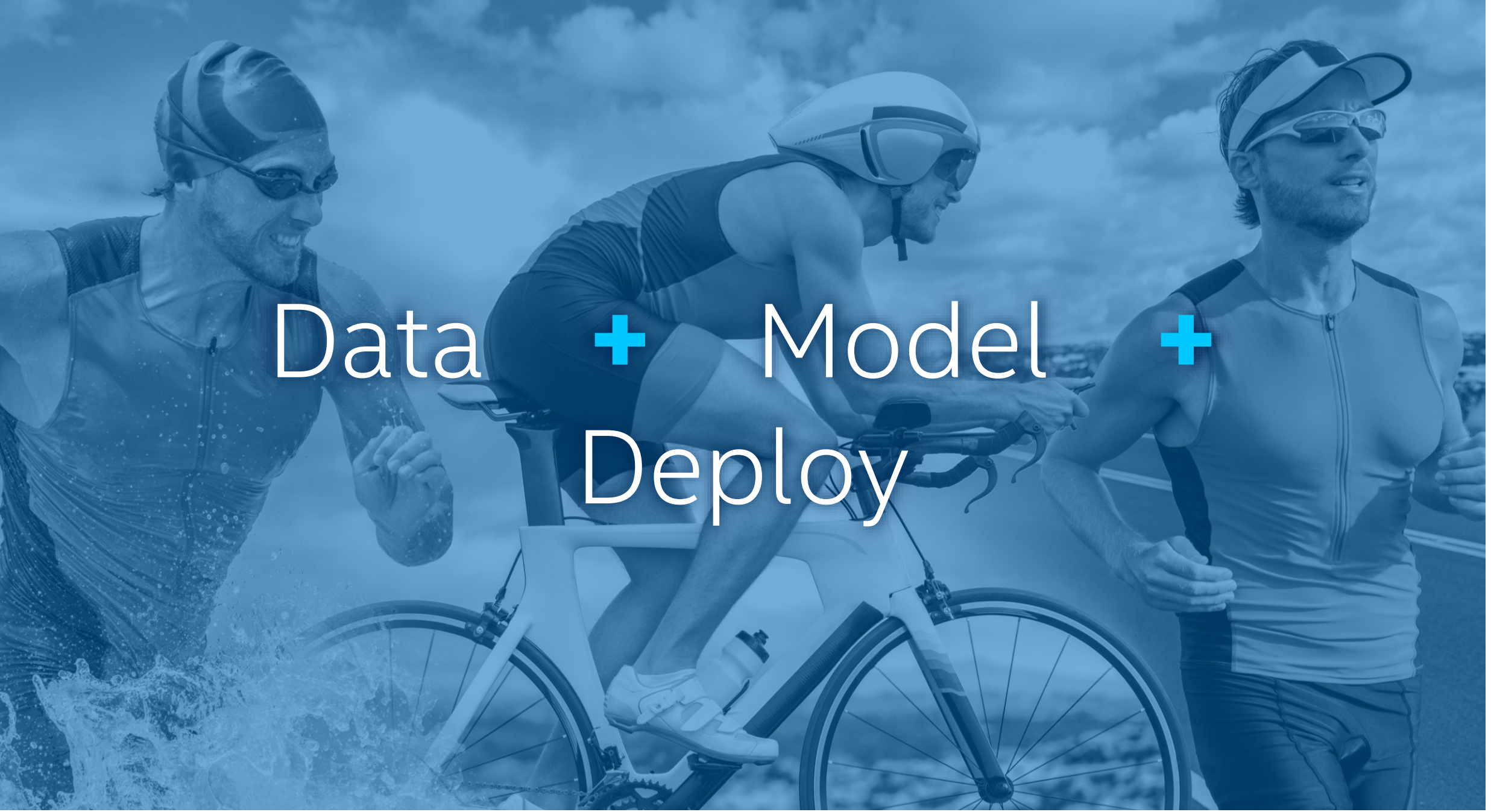
Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure

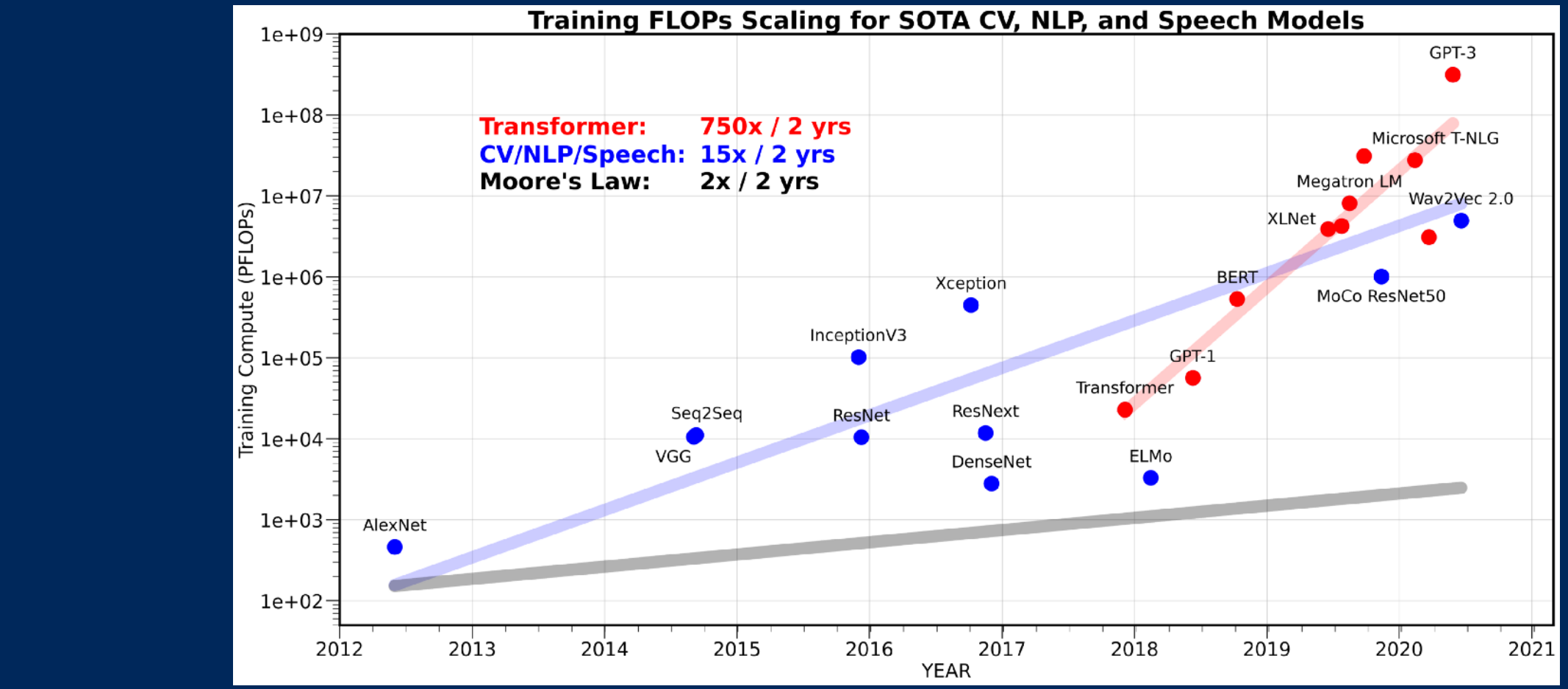
Your costs and results may vary.

Agenda

- Accelerated growth of AI demands on compute/memory/system
- AI HW and SW acceleration
- Multi-phased End-to-End AI optimization strategy
  - Data+AI SW, frameworks, graph compilers and low-level libraries
  - Low precision quantization
  - Learning efficiencies
  - System level optimizations
  - Hyperparameter optimizations
- Performance-efficient E2E AI pipelines
- Call to action


Data + Model +
Deploy

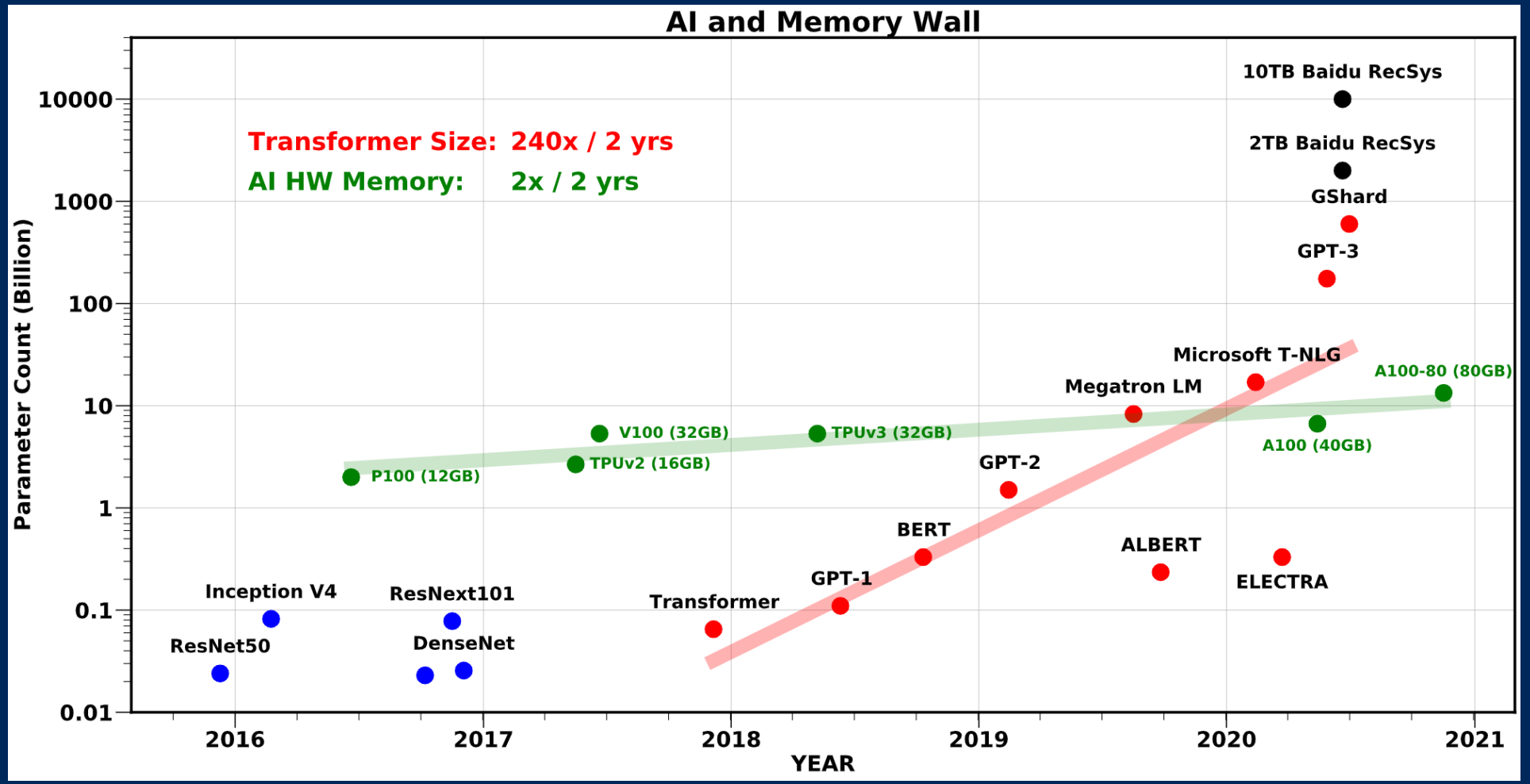
AI Loves TFLOPS?



But … how do we feed all that compute?

A. Gholami, et al., AI and Memory Wall. RiseLab Medium Blog Post, 2021

AI and the Memory wall



A. Gholami, et al., AI and Memory Wall. RiseLab Medium Blog Post, 2021

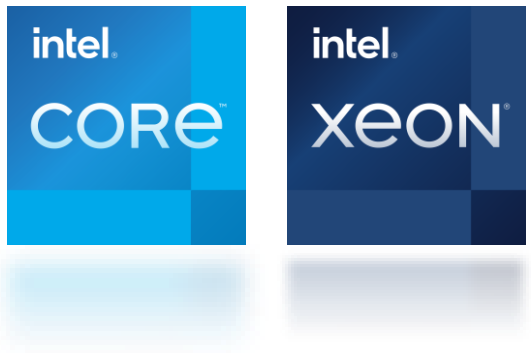
AI Hardware Acceleration

GENERAL PURPOSE — PURPOSE BUILT

CPU

GPU

ACCELERATORS



Does Software Acceleration Matter?

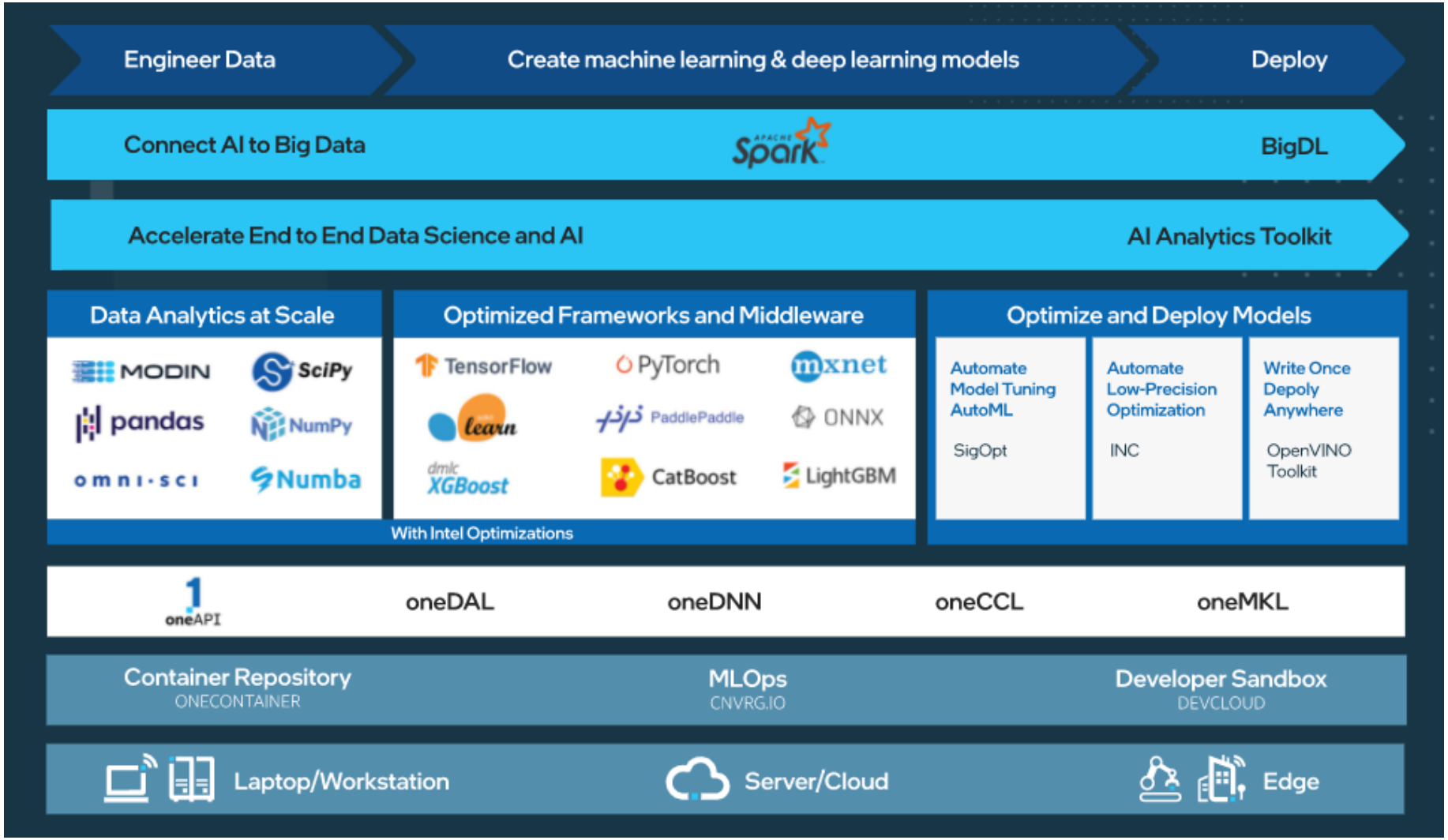
Hardware Acceleration

Up to

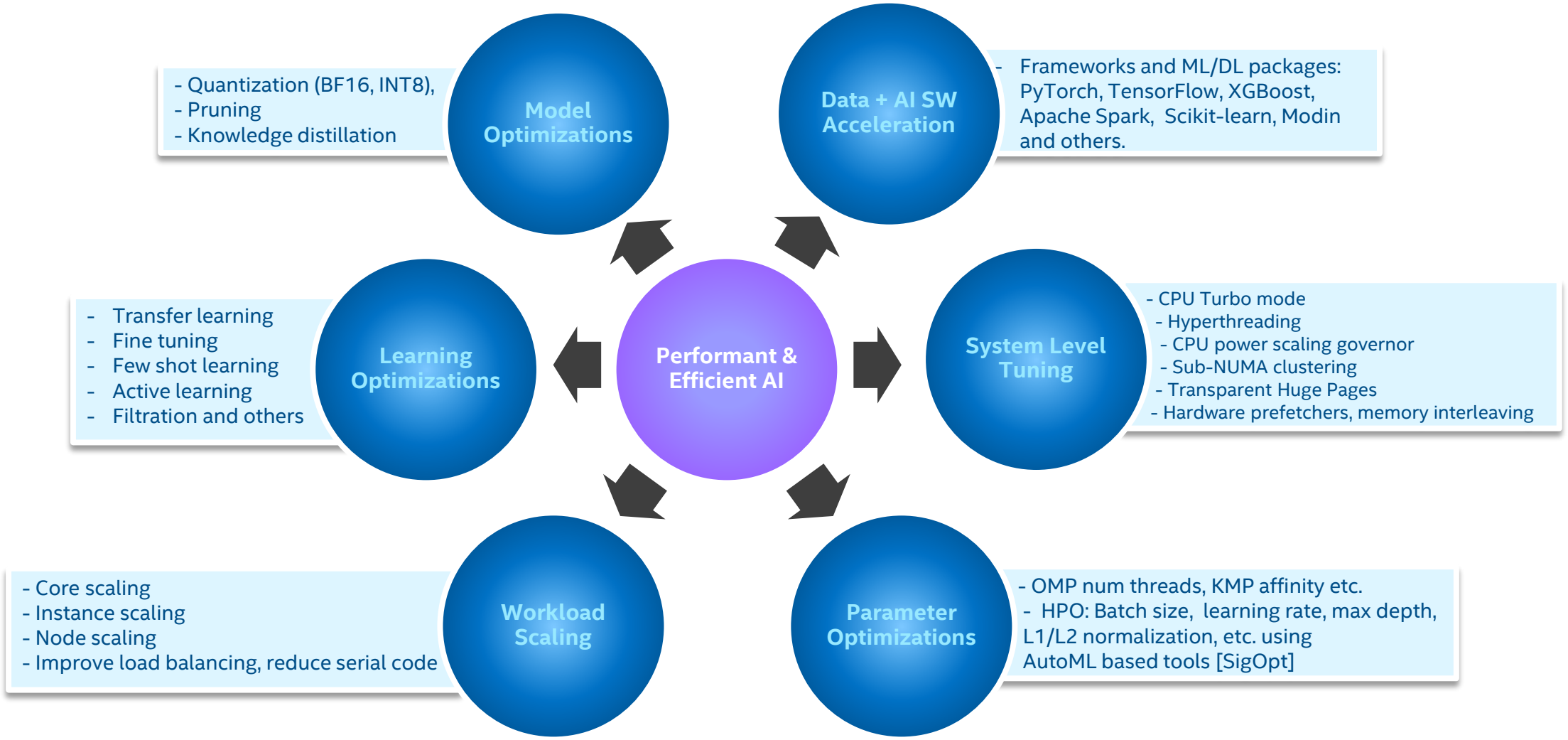
10 - 100x

with SOFTWARE Acceleration

End-to-End AI Software Suite

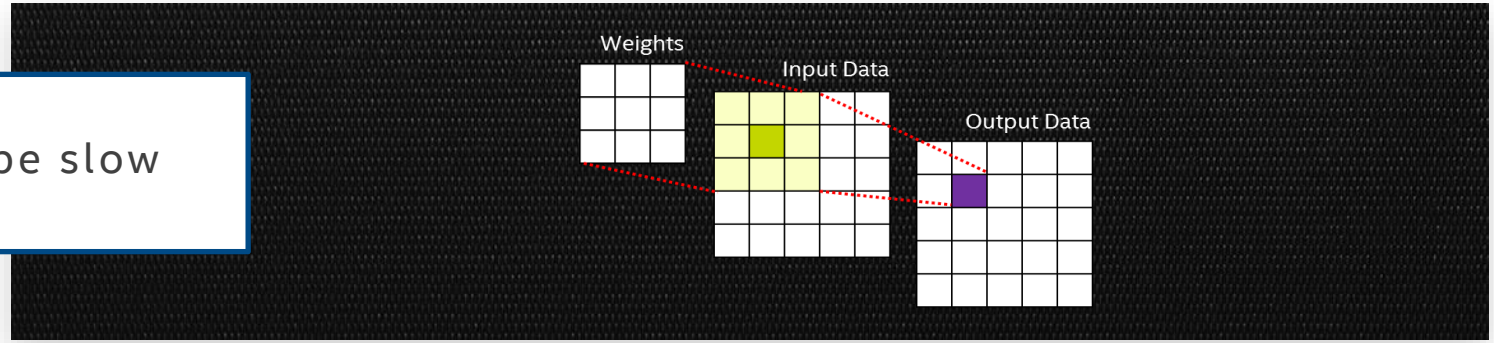


End-to-End AI Optimization strategies



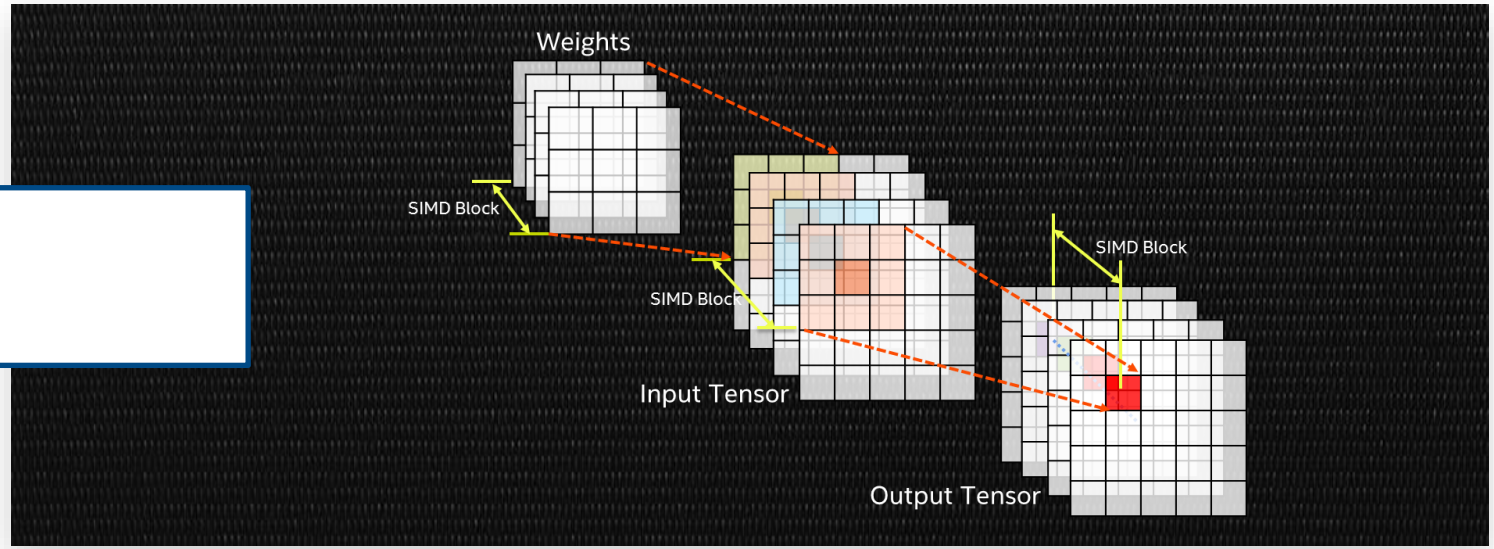
Deep Learning powered by oneDNN

A simple program is good, but may be slow

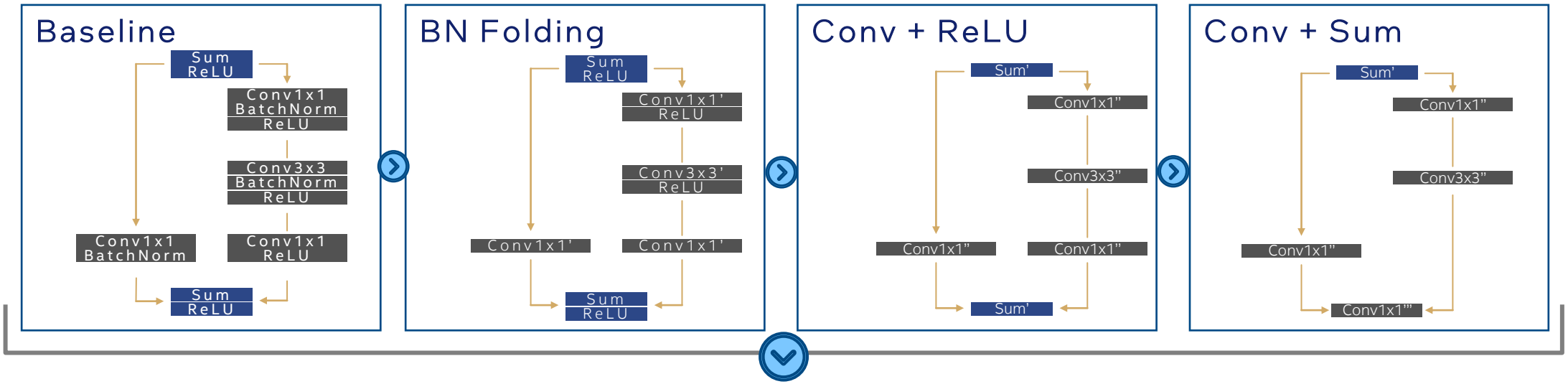


Optimizations: vectorization, data reuse, parallelization

Optimized convolution in oneDNN



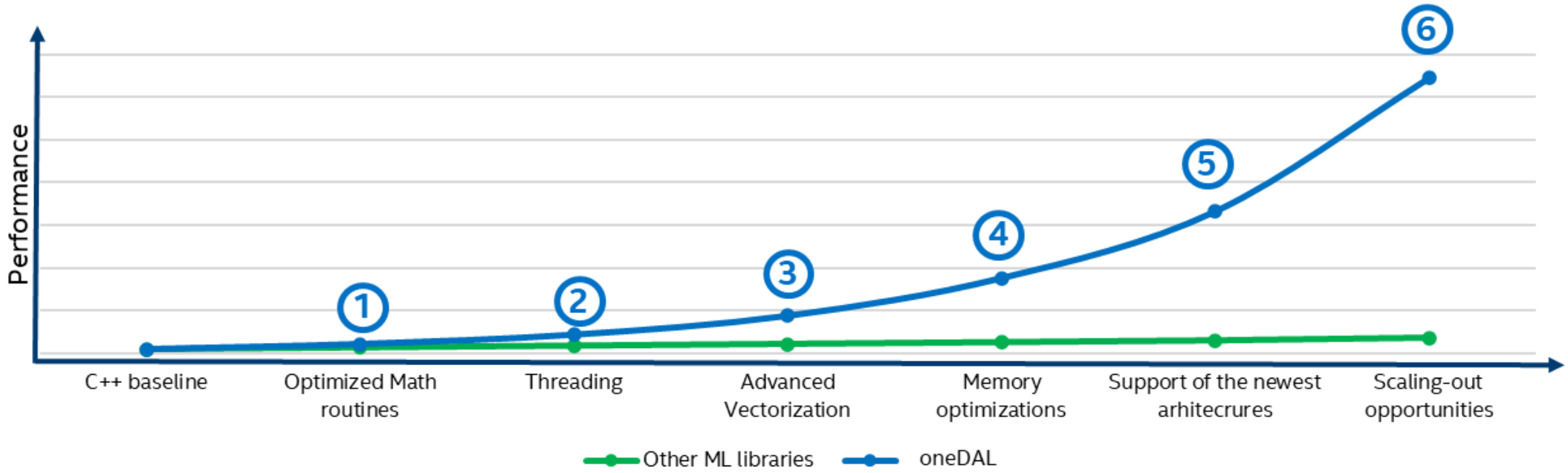
Graph Optimization Example



INT8 Optimized Model

A_0	A_1	A_2	A_3	...	A_{63}
B_0	B_1	B_2	B_3	...	B_{63}
C_0				...	C_{15}
$A_0*B_0+A_1*B_1+A_2*B_2+A_2*B_2+C_0$				...	$A_{60}*B_{60}+A_{61}*B_{61}+A_{62}*B_{62}+A_{63}*B63+C_{015}$

Machine learning powered by oneDAL



1 The best performance on Intel Architectures with oneMKL (Intel® MKL) vs. lower performance on BLAS/LAPACK libs

2 oneDAL targets to many-core systems to achieve the best scalability on Intel® Xeon, other libs mostly target to client versions with small amount of cores

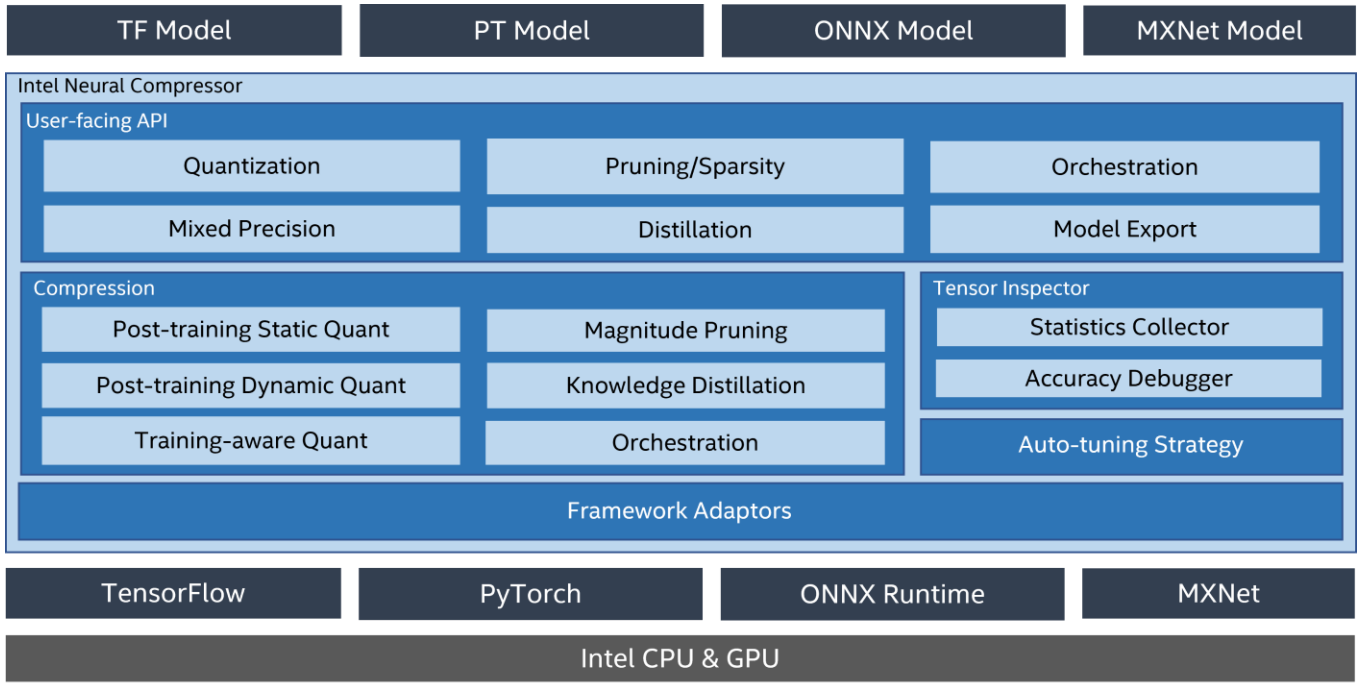
3 oneDAL uses the latest available vector-instructions on each architecture, enables them by compiler options, intrinsic. Usually, other ML libs build applications without vector-instructions support or sse4.2 only.

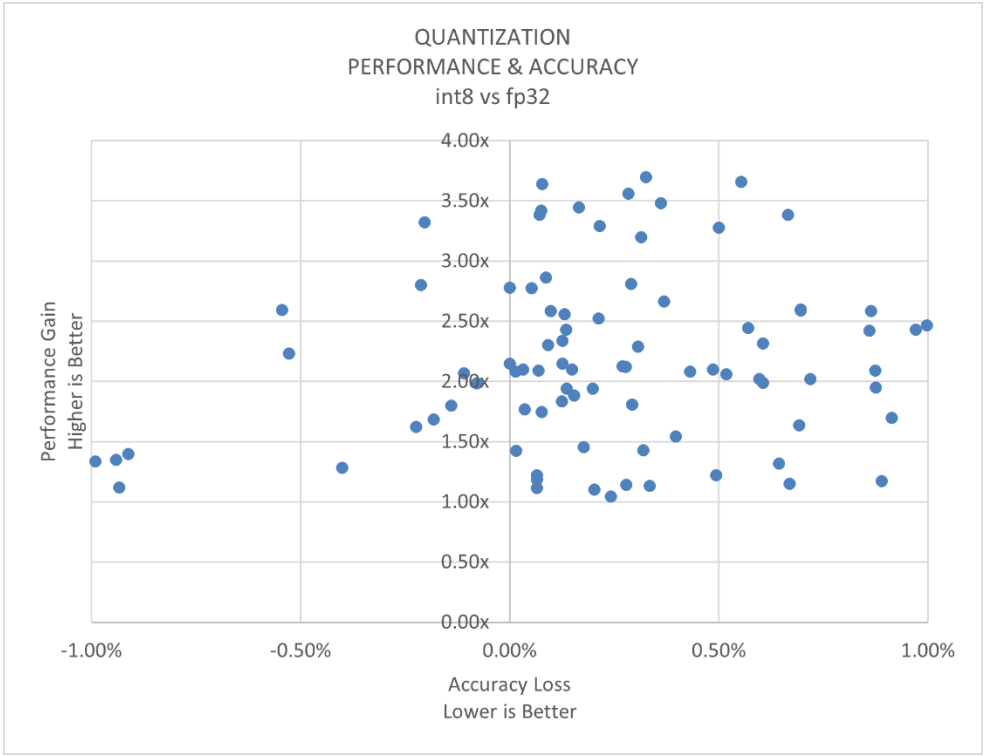
4 oneDAL uses the most efficient memory optimization practices: minimally access memory, cache access optimizations, SW memory prefetching. Usually Other ML libs don't make low-level optimizations.

5 oneDAL enables new instruction sets and other HW features even before official HW launch. Usually, other ML libs do this with long delay.

6 oneDAL provides distributed algorithms which scale on many nodes

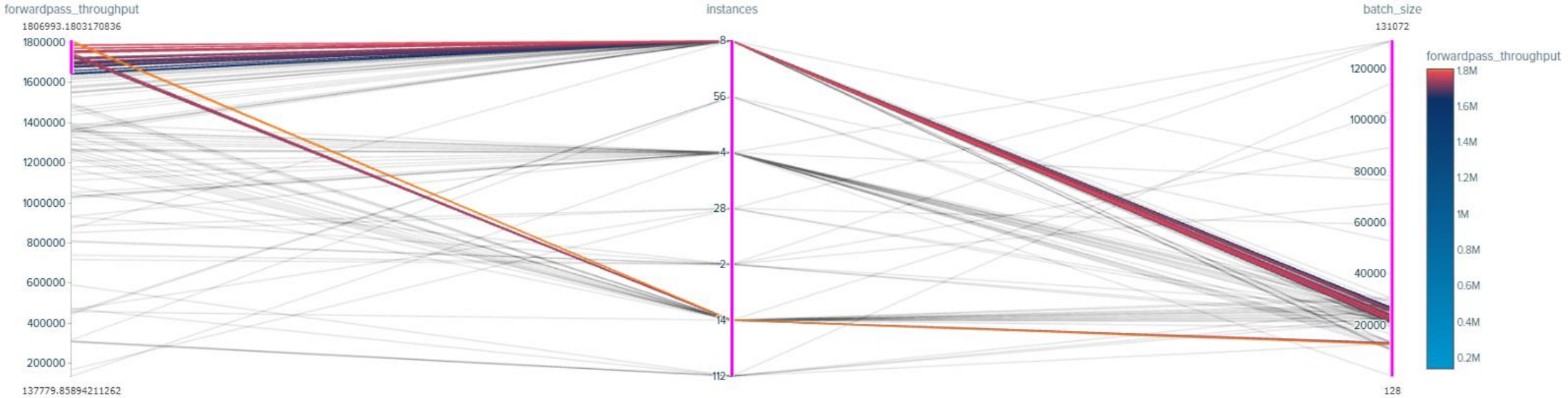
Low precision Inference: Model quantization

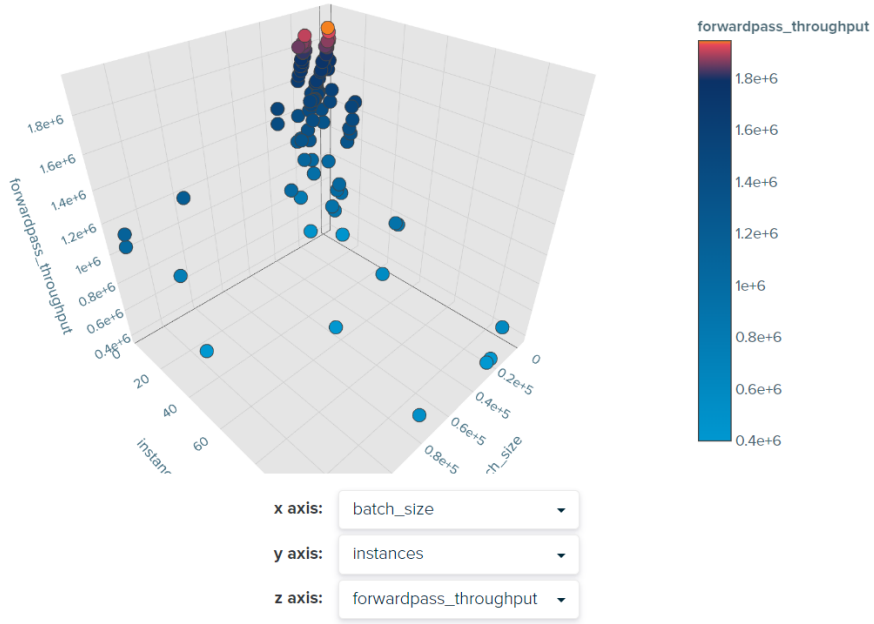




Model quantization to low precision yields significant performance speedups on a variety of models. Pruning, distillation and mixed precision strategies also further model efficiencies.

Hyper parameter and Instance tuning with SigOpt (DIEN)



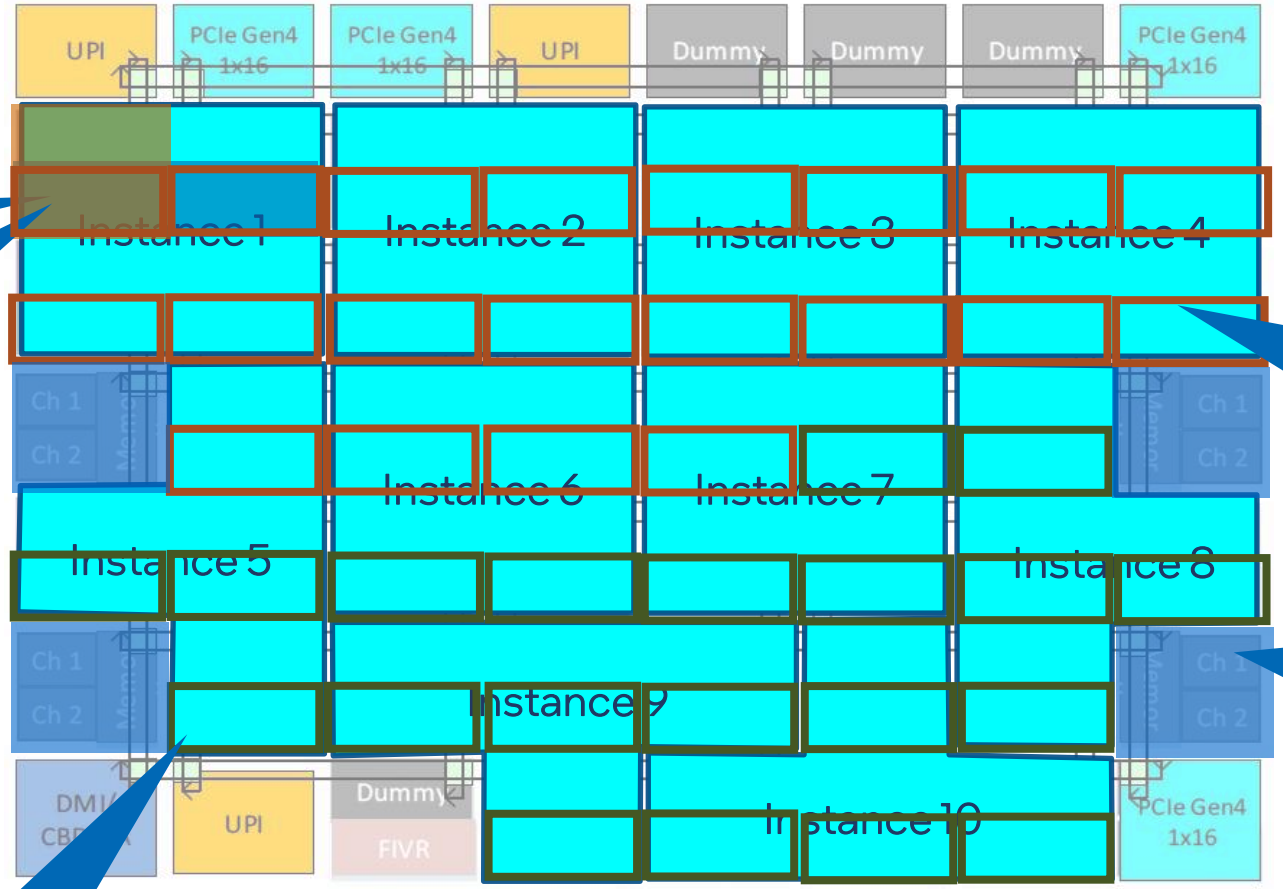


- https://sigopt.com/
- Sigopt features:
  - Easy to track runs, visualize training, and scale hyperparameter optimization
  - Advanced optimization Engine that delivers better results, faster and cheaper
  - Easy to use and parallelize for any type of model built with any library on any infrastructure

System level Tuning

Hyperthreading allows 2 threads to run on a core

Performance scaling drivers control core frequencies and power configurations



Intel Xeon Platinum 8380 Processor with 40 cores per socket

Instance scaling: This example is 10 Xeon instances per socket

Examples of BIOS/system level tuning:

- Hyperthreading
- CPU Turbo Boost Technology
- CPU power scaling governors
- NUMA optimizations
- Sub-NUMA Clustering
- Transparent Huge Pages
- Hardware prefetchers
- Channel interleaving
- Memory interleaving
- Data Direct I/O
- Hardware C-state

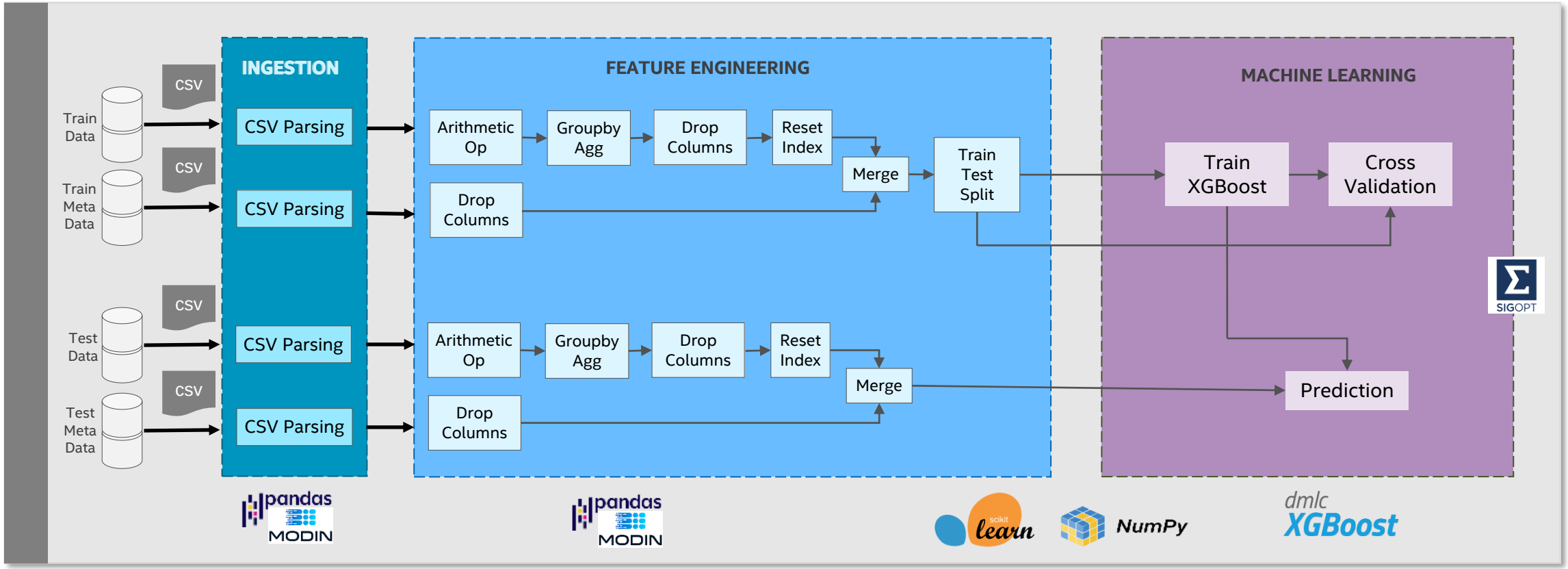
Sub-NUMA Clustering divides the cores, cache, and memory of the processor into multiple NUMA domains and helps workloads that are NUMA aware & optimized

Channel interleaving divides memory blocks and spread contiguous portions of data across interleaved channels, thereby increasing potential read bandwidth

Memory Interleaving allows physical ranks of memory to be accessed while another is being refreshed

PLAsTiCC Astronomical Classification

PLAsTiCC is an open data challenge to classify objects in the sky that vary in brightness using simulated astronomical time-series data. The challenge is to determine a probability that each object belongs to one of 14 classes of astronomical filters.



End-to-end ML optimizations

PLAsTiCC Astronomical Classification

Ingestion	Feature Engg	Train-Test-Split	Training	Inference
• Read data • Create dataframe	• Drop columns • Groupby agg • Arithmetic ops • Create feature set/ test set	• Train test split • Load numpy array to dmatrix objects		• Model prediction • Calculate accuracy

Modin

Transparently distributes the data and computation across available cores, unlike Pandas which only uses one core at a time.

Modin can be installed from PyPI:

```
pip install modin
```

```
# import pandas as pd
import modin.pandas as pd
```

Single line import change to run Modin instead of pandas

Scikit-learn

Foundational library to speed up your Scikit-learn application, that is highly optimized with low-level HW feature enabling to cover data analytics and machine learning.

```
from sklearn.svm import SVC
X, Y = get_dataset()

clf = SVC().fit(X, y)
res = clf.predict(X)
```

Scikit-learn mainline

Scikit-learn with Intel CPU opts

```
from sklearnex import patch_sklearn
patch_sklearn()
from sklearn.svm import SVC
X, Y = get_dataset()
clf = SVC().fit(X, y)
res = clf.predict(X)
```

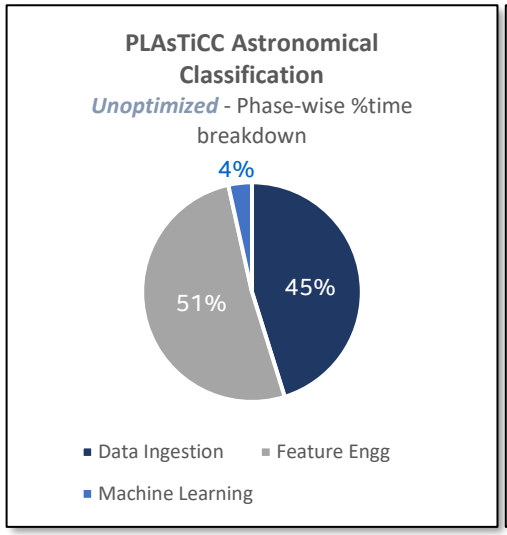
Available through PyPi
pip install scikit-learn-intelex

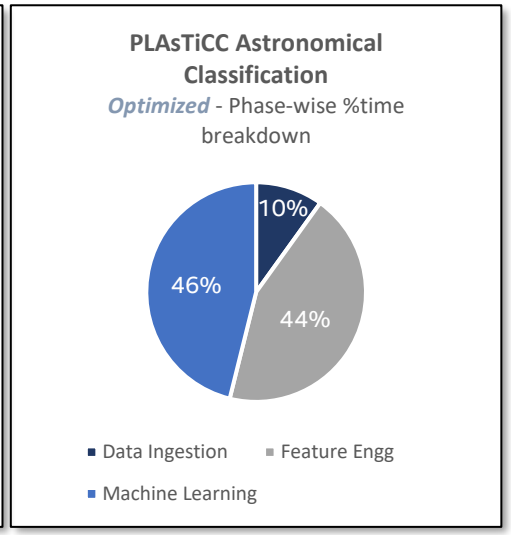
XGBoost

Intel® optimizations are now available as part of mainline XGBoost repository.

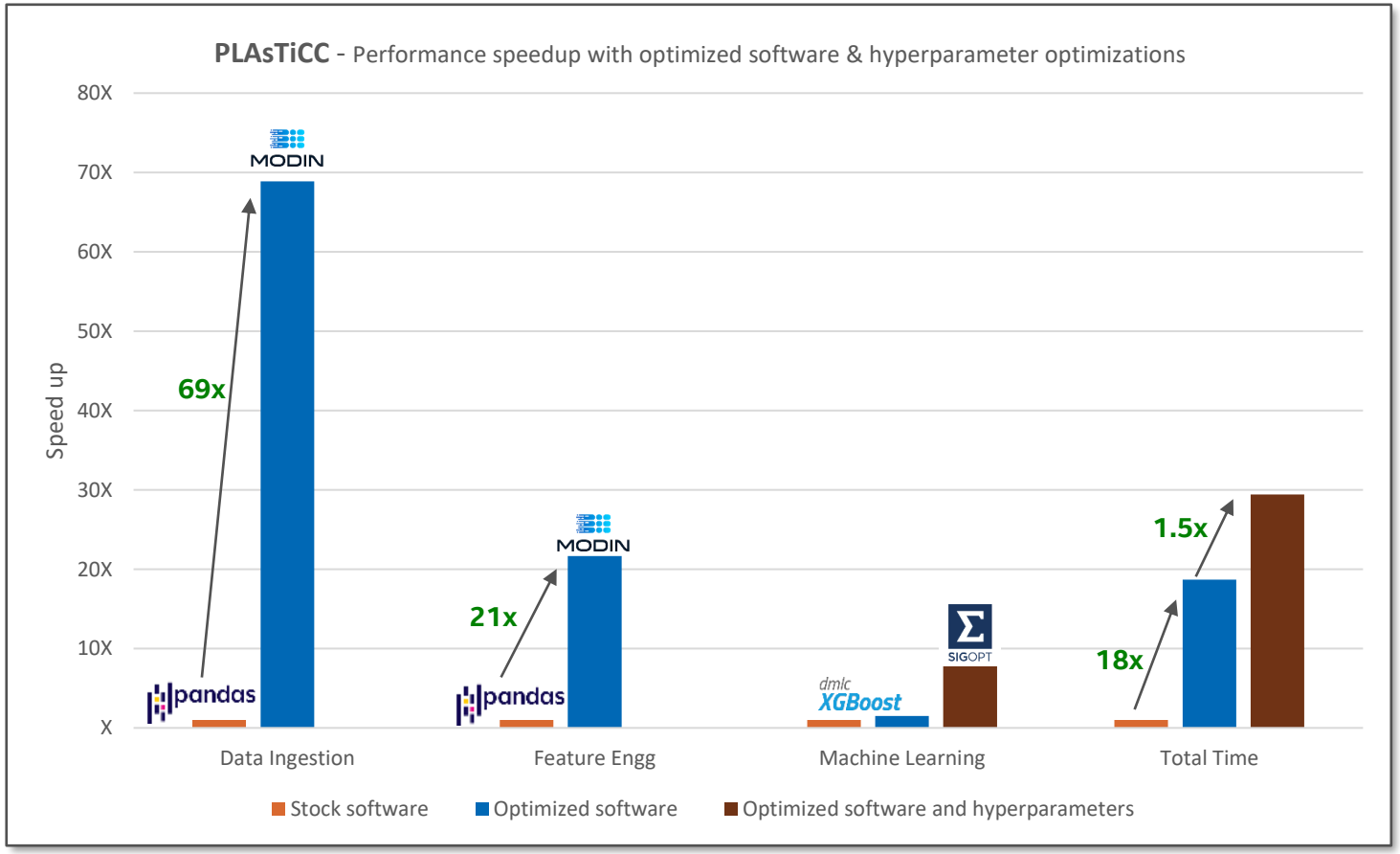
PLAsTiCC Astronomical Classification

Performance with optimized software & hyperparameters





- 14GB dataset with 1.4 millions rows in training and 189 million rows in test dataset – takes advantage of Modin's extremely light-weight, robust Dataframe & readcsv operation which scales with cores, unlike pandas
- Operations in feature engineering are memory bound and benefit from the faster memory access speeds
- Microarchitecture factors like higher core frequency, cache size, cache BW help improve XGBoost performance

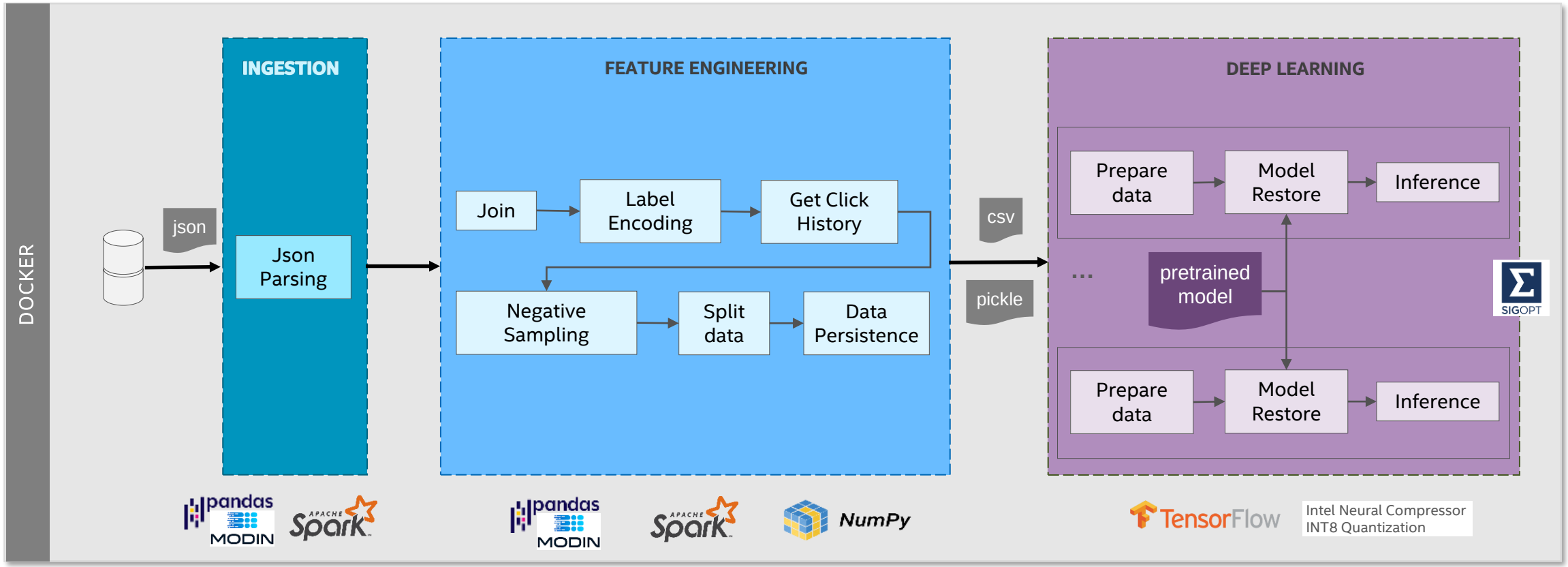


Hardware: 2 x Intel Xeon Platinum 8280L (28 cores), OS: Ubuntu 20.04.1 LTS Mitigated, 384 GB RAM (384 GB RAM: 12 x 32 GB 2933 MHz), kernel: 5.4.0–65-generic, microcode: 0x4003003, CPU governor: performance.
Software: scikit-learn 0.24.1, pandas 1.2.2, XGBoost 1.3.3, Python 3.9.7, scikit-learn-intelex 2021.2, modin 0.8.3, omniscidbe v5.4.1.

Higher is better

E2E Recommendation System (E2E DIEN)

E2E DIEN: A representative E2E inference workload of recommendations that provides the capability of estimating use clicks at scale.



E2E DIEN

Deeper Analyses and insights: Optimizations

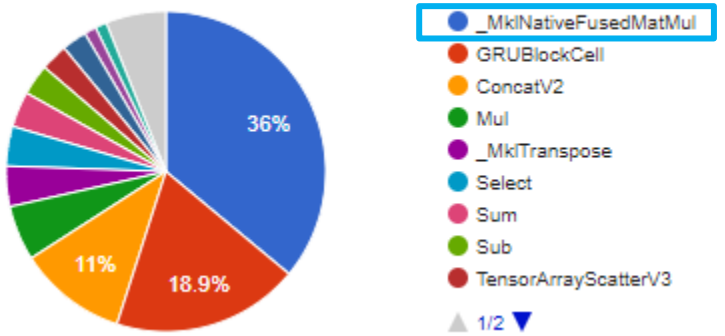
Optimized frameworks take advantage AVX512 and AMX instructions.

Conversion to Ops to Intel oneDNN (MKL) Ops optimized for Intel HW

Operator Fusions enable even more speedup & efficient use during runtime execution.

- GRU(MklFusedMatmul)
- Attention layer(MklFusedMatmul)

(in microseconds) of a TensorFlow operation type

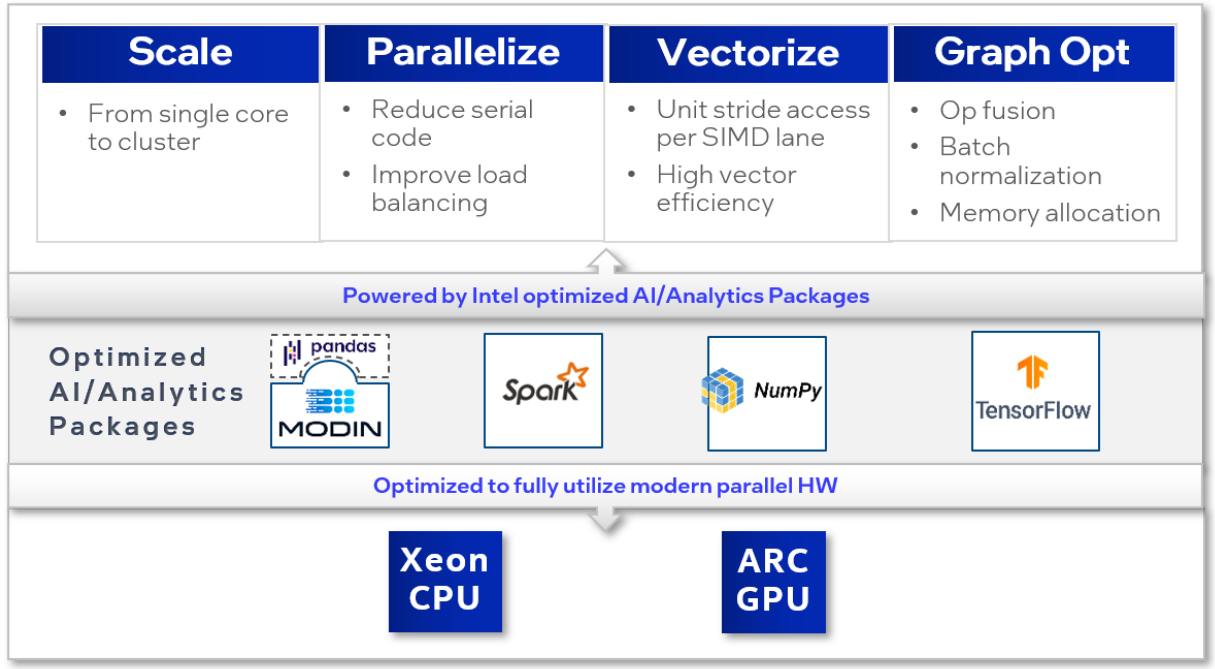


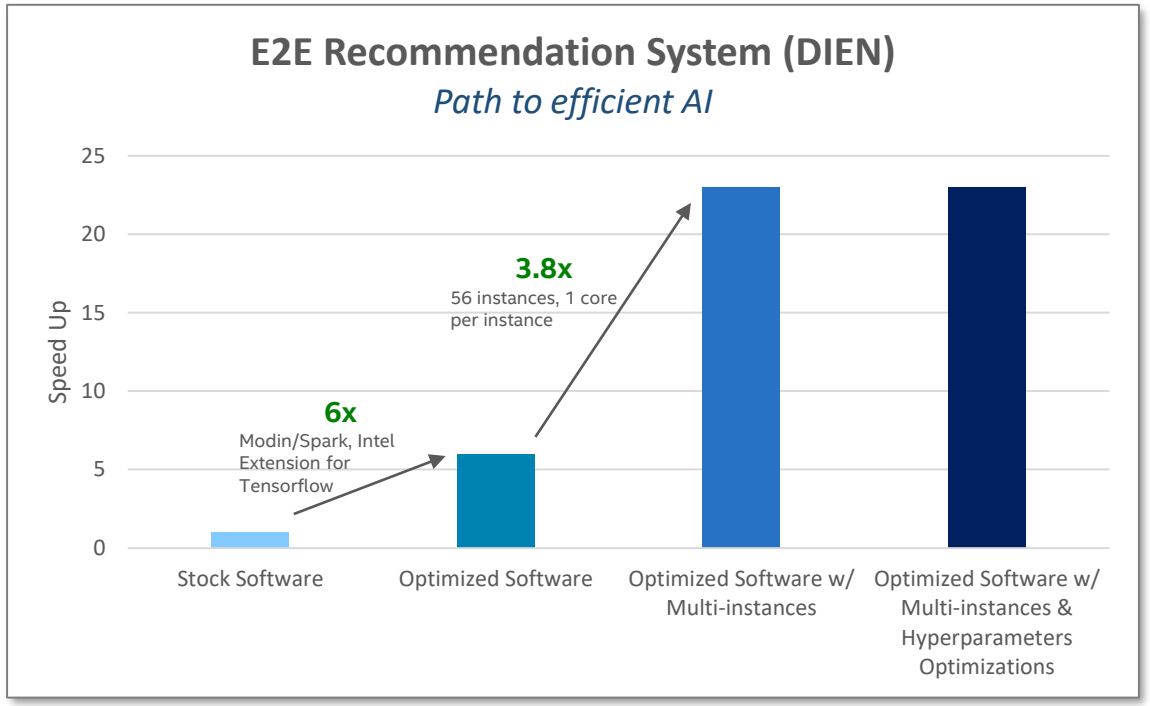
Host-side TensorFlow operations (grouped by TYPE)

Type	#Occurrences	Total self-time (us)	Total self-time on Host (%)
_MklNativeFusedMatMul	1,609	585,140	36%
GRUBlockCell	598	307,566	18.9%
ConcatV2	1,550	179,002	11%
Mul	3,041	90,238	5.6%
_MklTranspose	118	66,129	4.1%
Select	1,794	65,005	4%
Sum	59	58,533	3.6%
Sub	1,491	50,455	3.1%
TensorArrayScatterV3	118	44,474	2.7%
GatherV2	295	41,063	2.5%
TensorArrayGatherV3	59	19,727	1.2%
Split	598	18,840	1.2%
AddV2	2,325	15,684	1%
Tile	59	14,771	0.9%
Square	118	12,585	0.8%
RealDiv	118	10,951	0.7%
Mean	295	6,242	0.4%
TensorArrayWriteV3	598	6,122	0.4%
Softmax	118	5,053	0.3%
Sigmoid	118	4,476	0.3%
SelectV2	59	4,337	0.3%
StridedSlice	893	4,272	0.3%
Fill	236	3,820	0.2%
GreaterEqual	1,196	1,766	0.1%
TensorArrayV3	177	1,615	0.1%
Pack	1,373	1,337	0.1%
Minimum	59	1,302	0.1%
Less	1,314	1,187	0.1%
Equal	236	1,155	0.1%

Type
_MklNativeFusedMatMul

E2E Recommendation System (E2E DIEN)



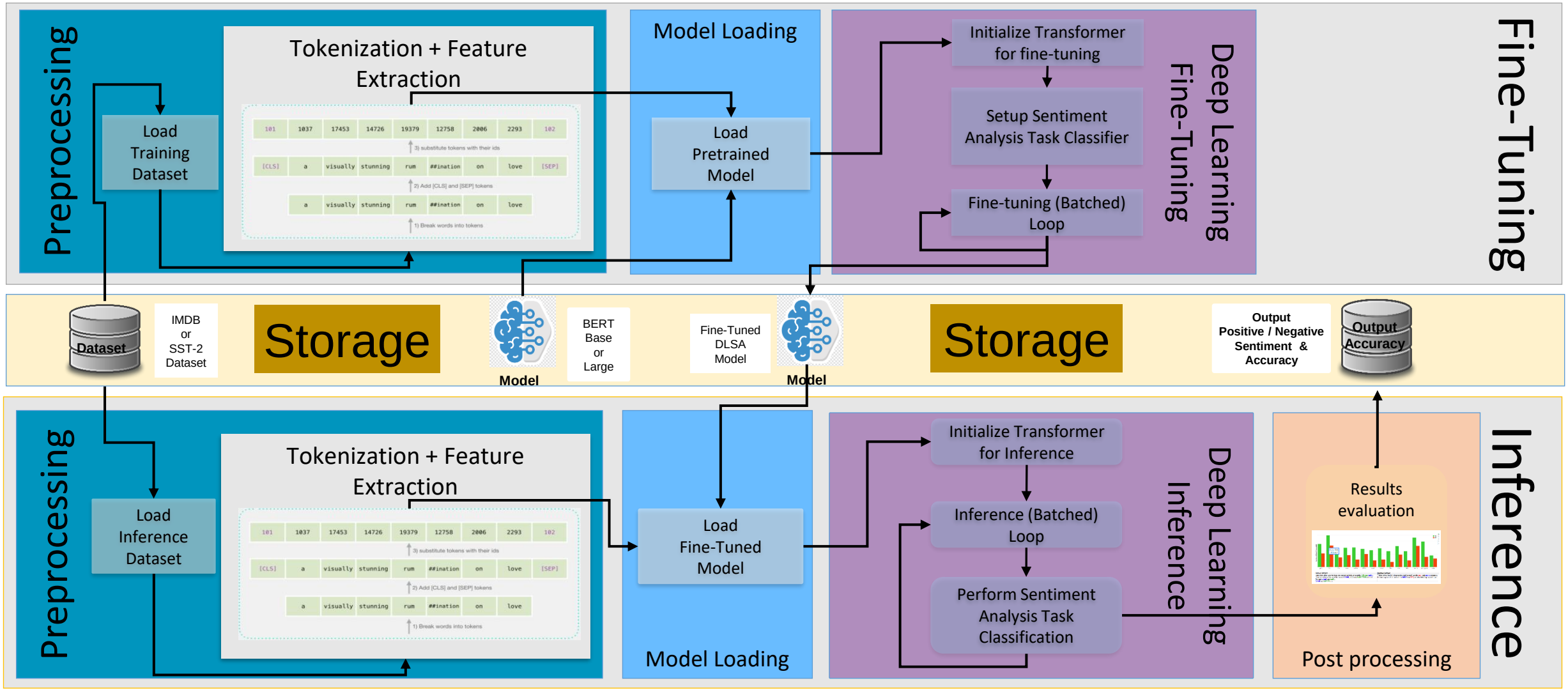


Configuration: Hardware: Single node, 2 x Intel Xeon Gold 6348 (28 cores), OS: RHEL 8.4, 384 GB RAM (512 GB RAM: 12 x 32 GB 3200 MHz), kernel: 4.18.0-305.el8.x86_64, microcode: 0xd000280, CPU governor: performance.
Software: Modin 0.12.0, Python 3.8.10, intel-tensorflow-avx512 2.7.0.

E2E Recommendation system got significant improvement of ~23x over baseline by applying multiple optimization strategies together.

E2E NLP Document-level Sentiment Analysis (DLSA)

A representative E2E Fine-Tuning & Inference NLP workload with Sentiment Analysis Task on Huggingface Transformer API



Transformers

Transformers
PYTORCH

PYTORCH
Transformers
TensorFlow

E2E NLP Sentiment Analysis (DLSA)

Optimized frameworks take advantage AVX512 and AMX instructions.

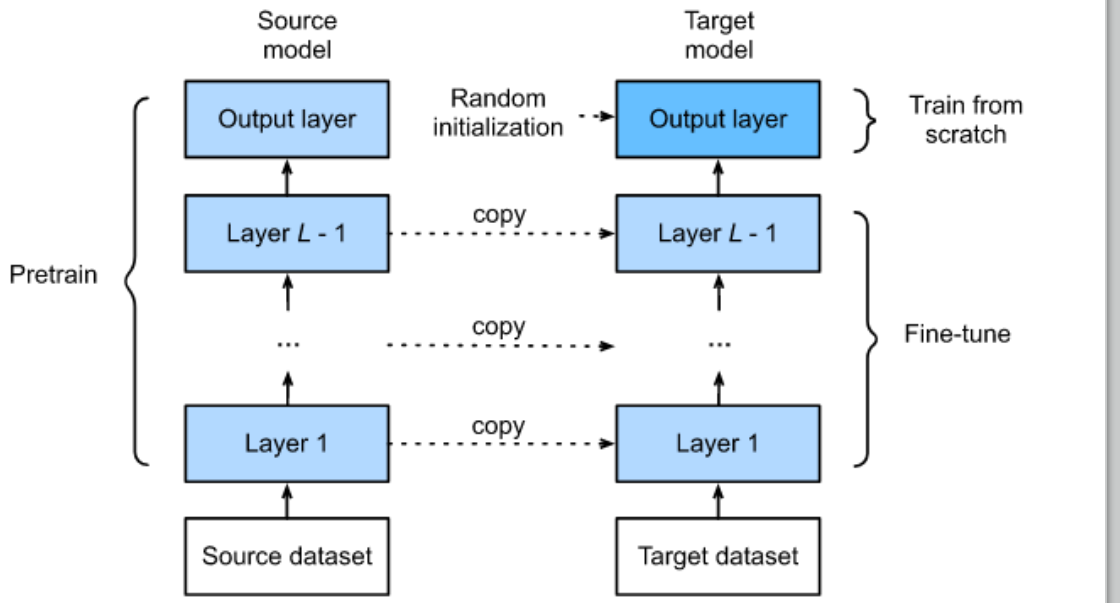
Conversion to Ops to Intel oneDNN (MKL) Ops optimized for Intel HW

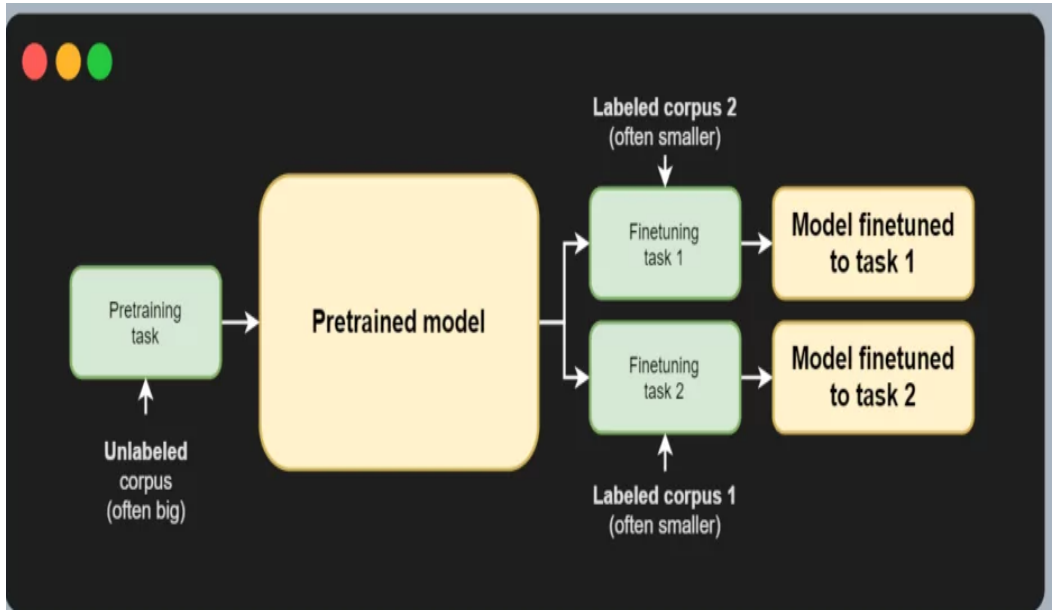
Operator Fusions enable even more speedup & efficient use during runtime execution.

- Gelu (Bunch of smaller ops fused to Gelu Op tf.nn.gelu)
- Fusion of MatMul + BiasAdd + Gelu(MklFusedMatmul)
- Fusion of BatchMatMul + Mul + Add(MklBatchMatmul)
- LayerNormalization (Bunch of smaller ops for keras layernorm api)(MklLayernorm)

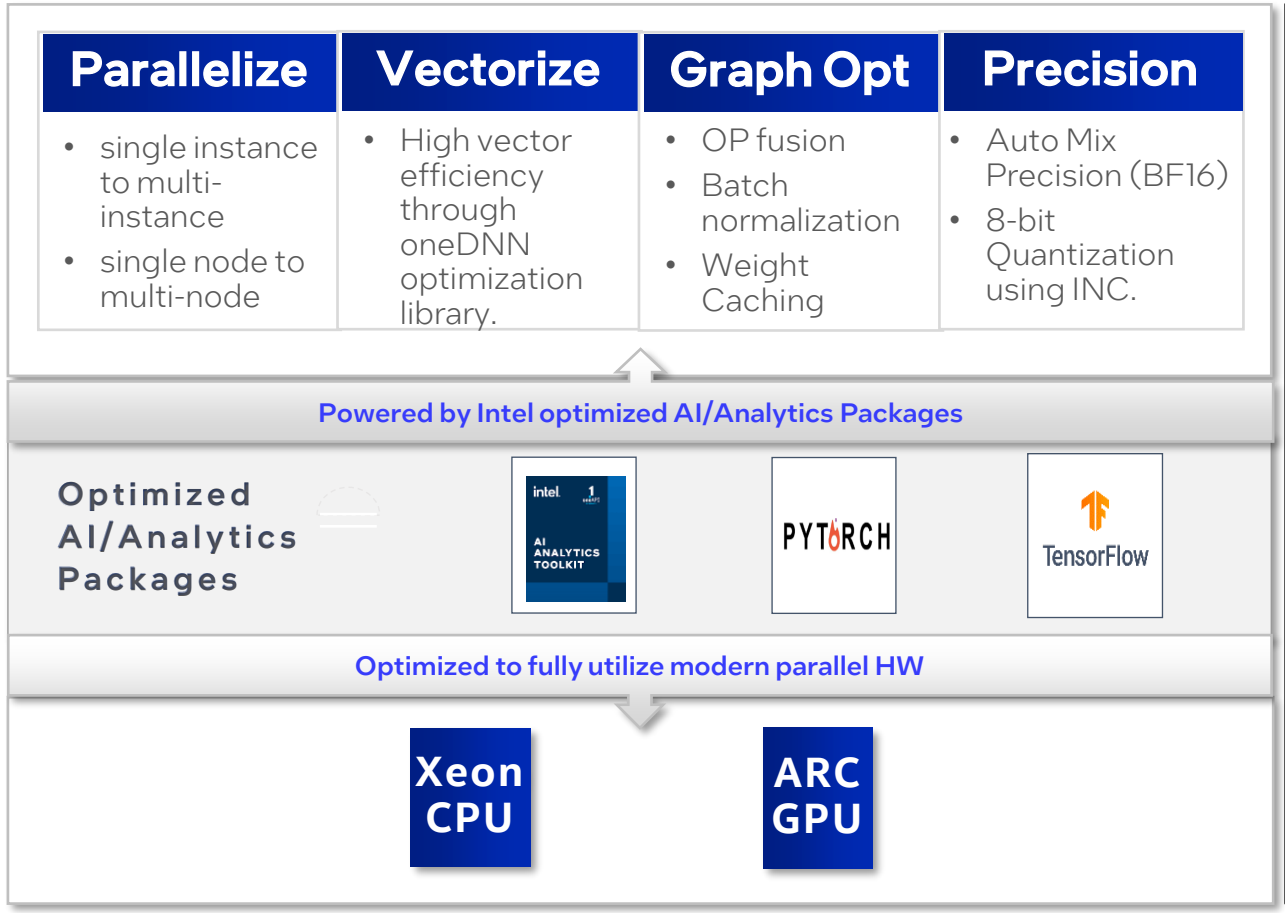
Fine-Tuning Explained with Context to Hugging face DLSA

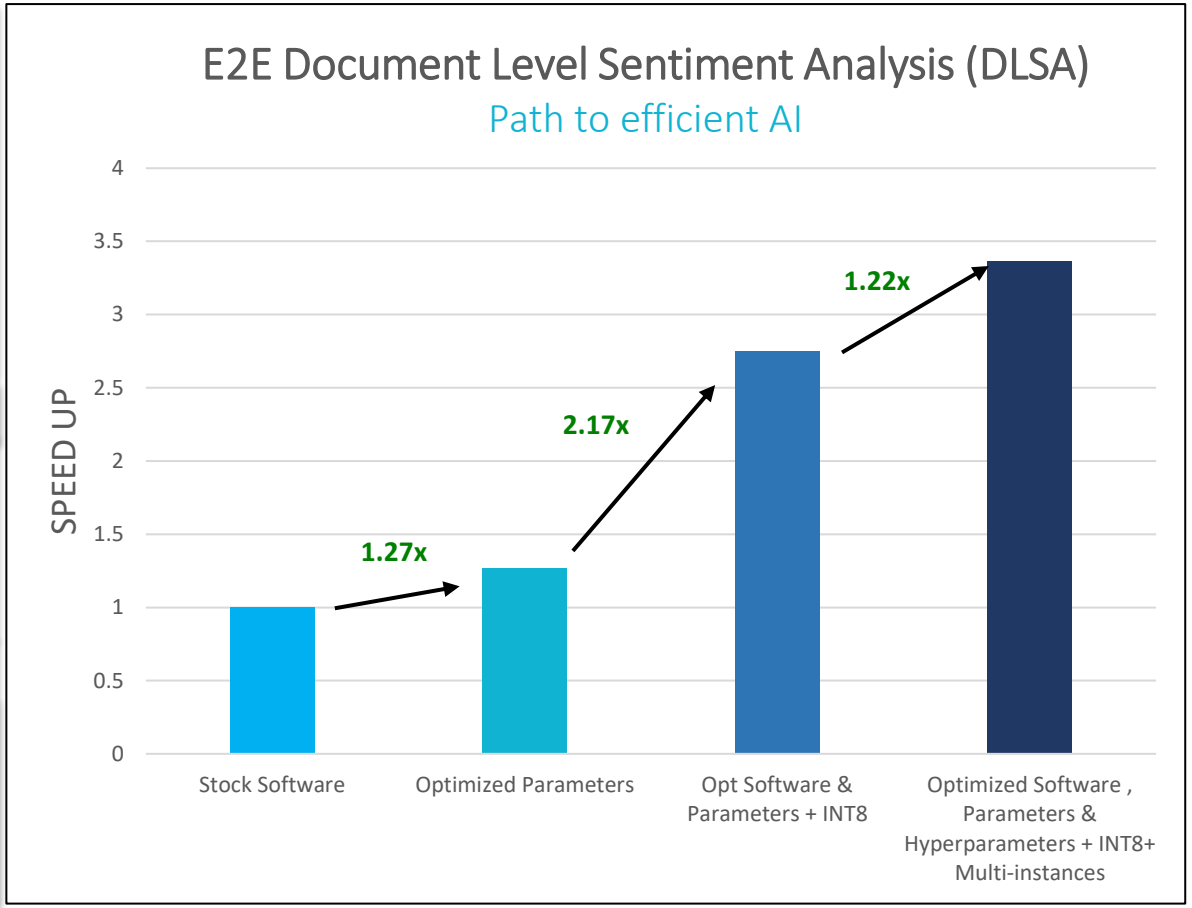
- Process of using a pretrained model, trained on a different source dataset, to train (modify training parameters) of a new target model (fine-tune it) with a different task (output layer) using a different target dataset .
- In DLSA we use BERT model for Masked-Language-Modeling task pretrained on large corpus of English data, to fine tune a new BERT model for sentiment analysis task on SST-2 or IMDB dataset.





E2E NLP Sentiment Analysis Inference (DLSA)





Configuration: Hardware: Single node, 2 x Intel Xeon Gold 6348 (28 cores), OS: RHEL 8.4, 384 GB RAM (512 GB RAM: 12 x 32 GB 3200 MHz), kernel: 4.18.0-305.el8.x86_64, microcode: 0xd000280, CPU governor: performance.
Software: Python 3.8.10, PyTorch 1.10, intel-extension-for-pytorch 1.10, transformers 4.15.0

E2E DLSA got significant total improvement of ~3.36x over baseline by applying multiple optimization strategies together.

Call To Action

- Efficient AI happens when AI frameworks and libraries, system tuning, model & hyperparameter optimization and run-time parameter **tuning all work together cohesively**
  - A clear path towards end-to-end AI performance roofline is achievable
- Every phase of an **end-to-end AI pipeline** needs to be efficient and optimized to realize an overall effective AI solution
- **Performance acceleration with "optimization toolbox"** strategies on CPUs brings significant boost in efficiency for end-to-end AI pipelines