Intel® oneAPI AI Analytics Toolkit

Data Analytics & Machine Learning

Agenda

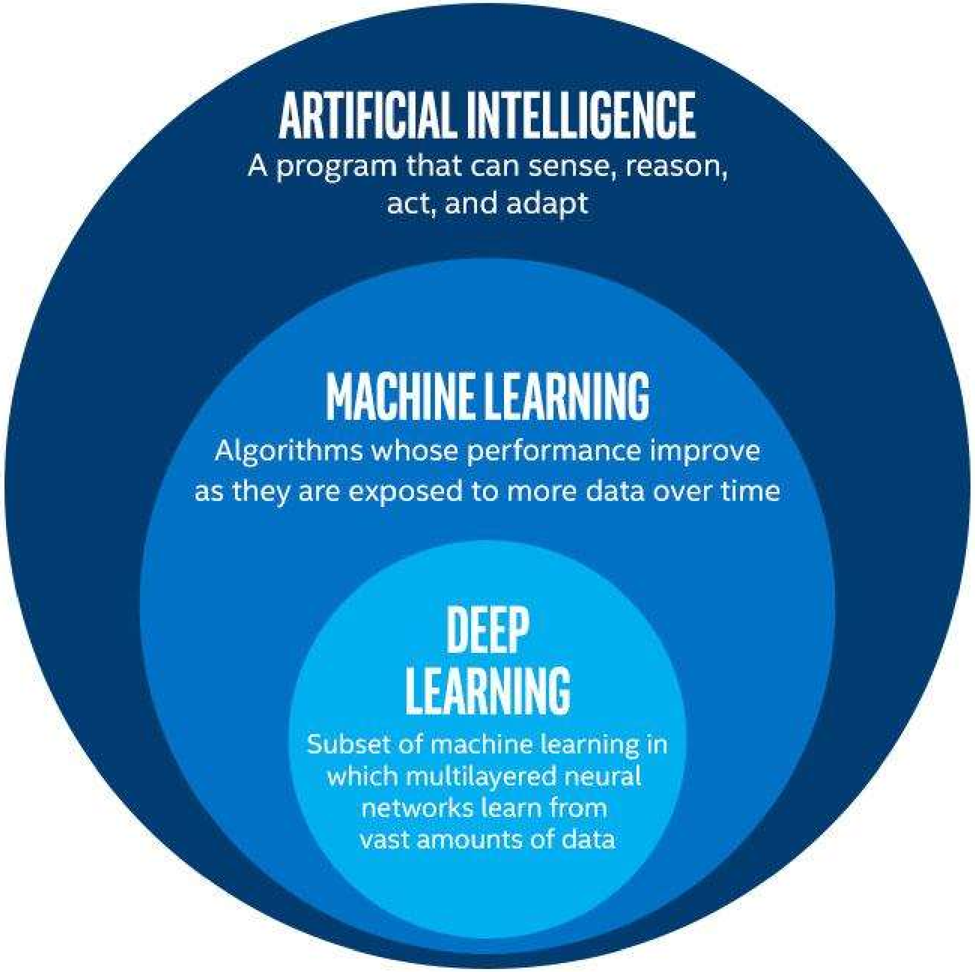
- Concepts of AI & ML & DL
- Data Analytics & Machine Learning Optimizations powered by Intel® oneAPI AI Anaytics Toolkit
- Demo
- Q&A

Concepts

Artificial Intelligence & Machine Learning & Deep Learning

Definitions

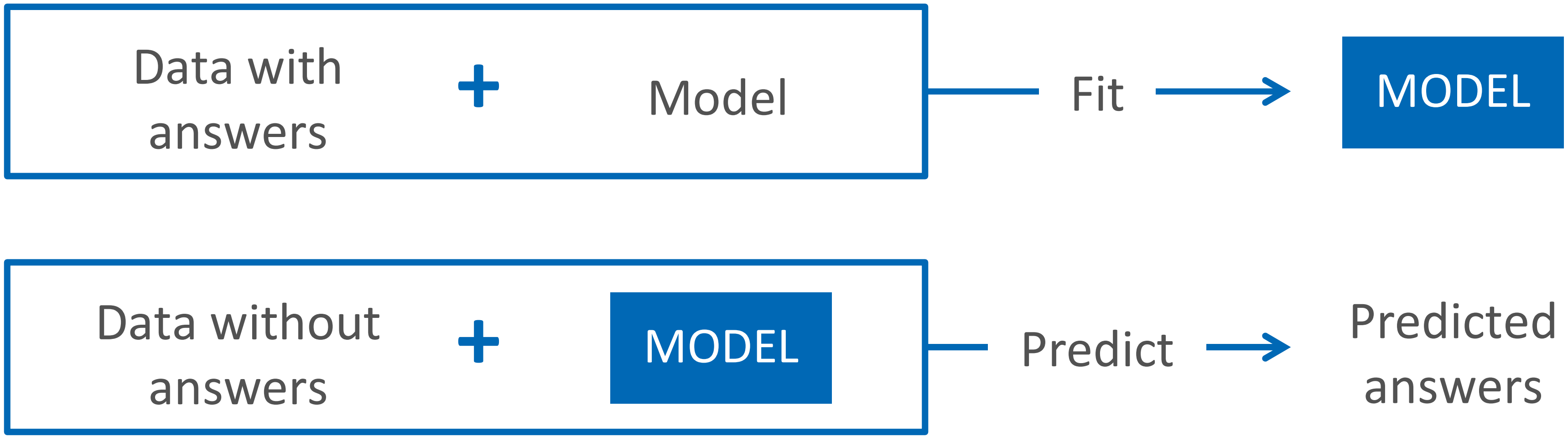
- Artificial Intelligence
- Machine Learning
- Deep Learning



Two Main Types of Machine Learning

	Dataset	Goal
Supervised Learning	Has a target column	Make predictions
Unsupervised Learning	Does not have a target column	Find structure in the data

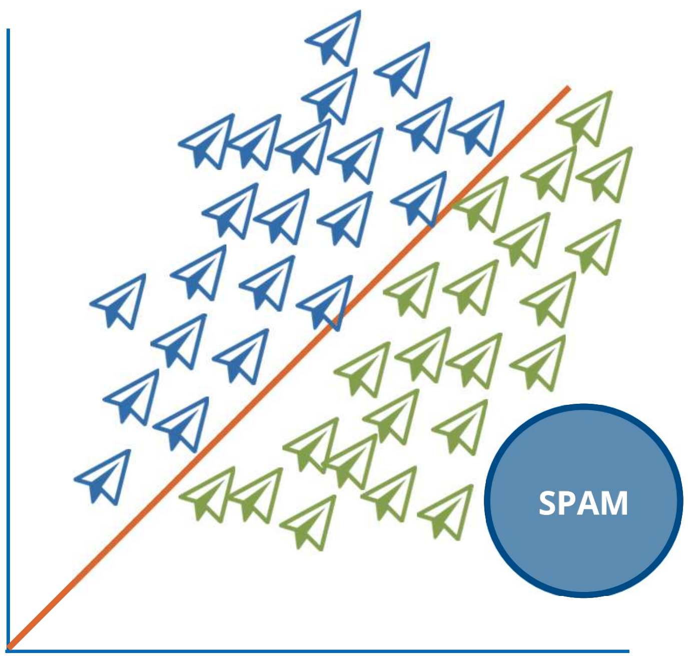
Supervised Learning



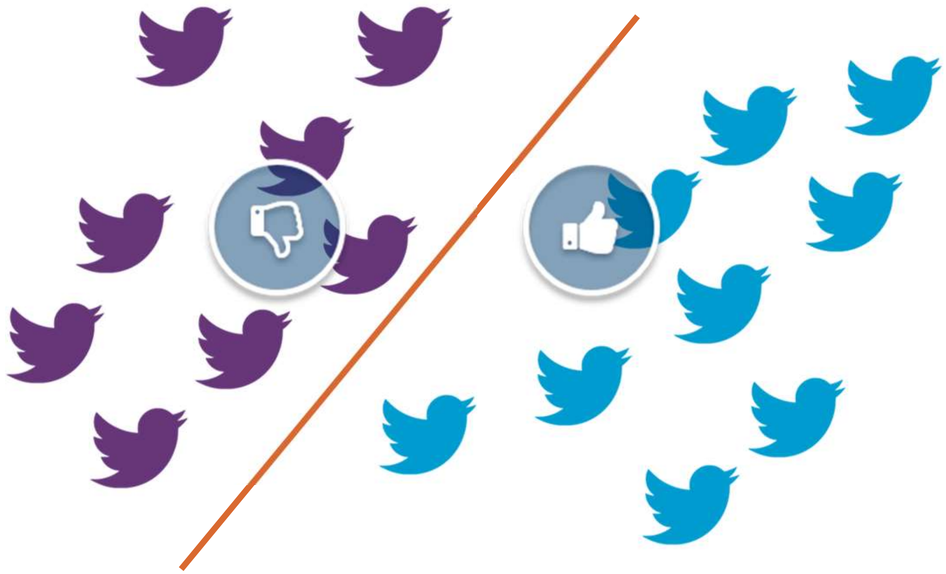
Classification

Predict a label for an entity with a given set of features.

Prediction



Sentiment Analysis



Regression

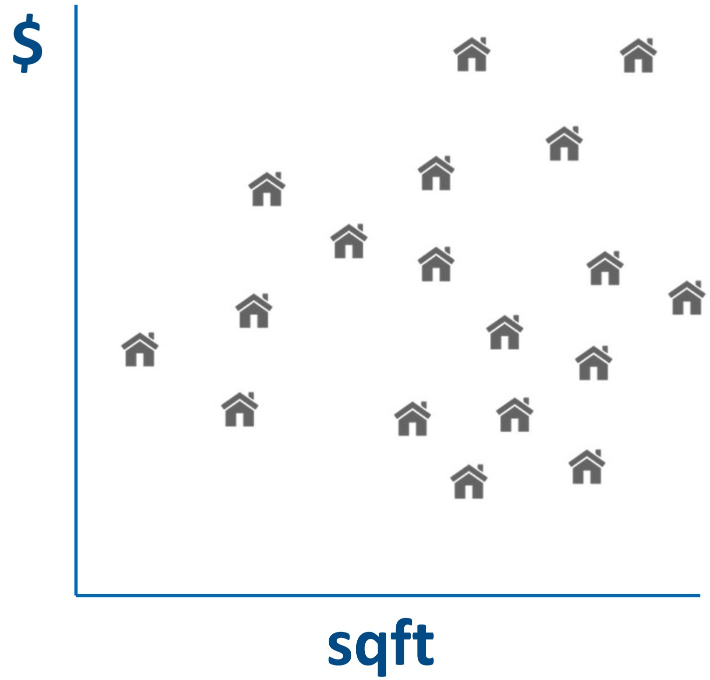
Predict a real numeric value for an entity with a given set of features.

Property Attributes

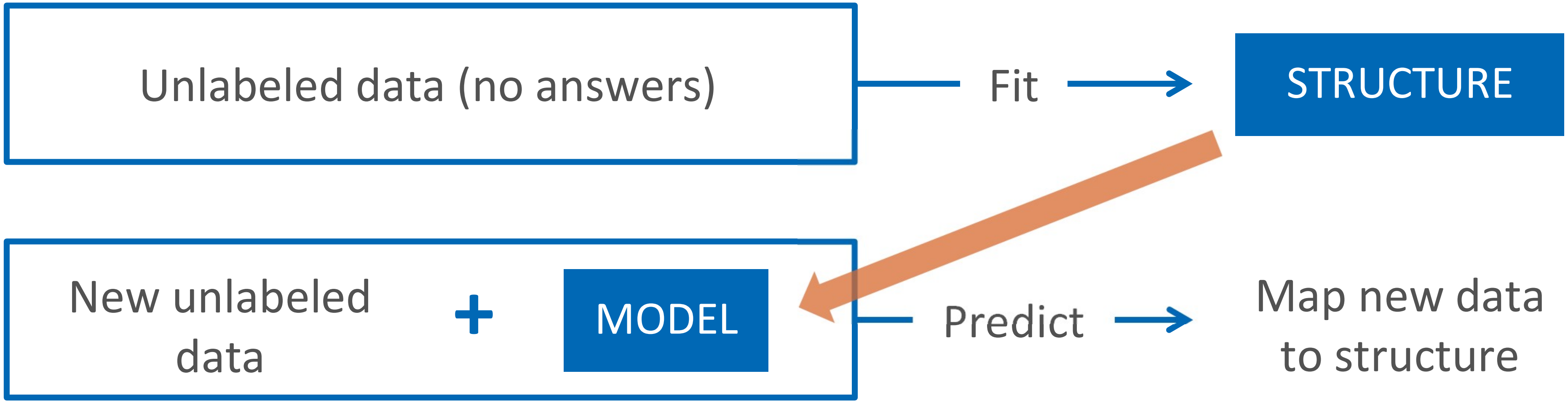
Price
Address
Type
Age
Parking
School
Transit

Total sqft
Lot Size
Bathrooms
Bedrooms
Yard
Pool
Fireplace

Linear Regression Model

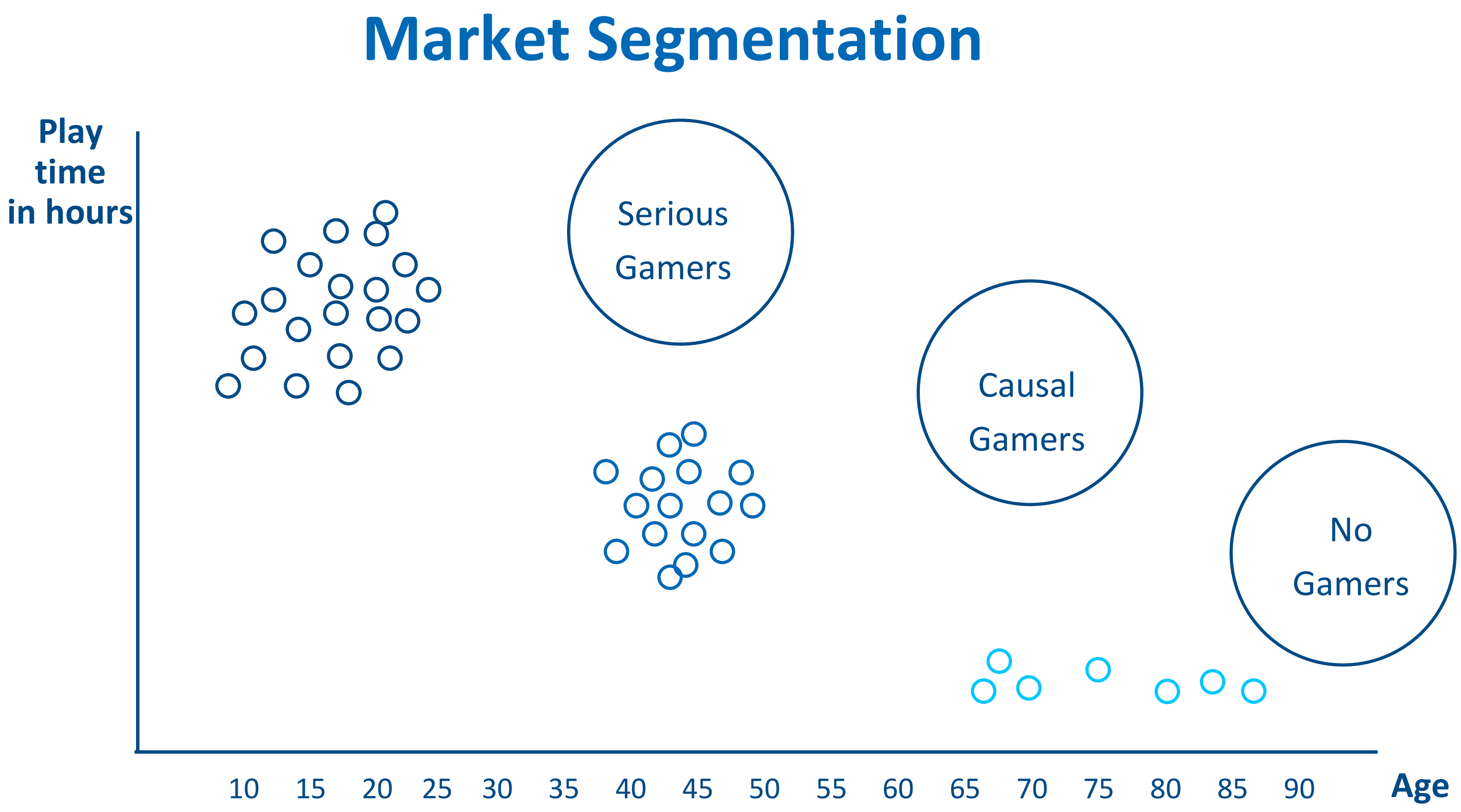


Unsupervised Learning



Clustering

Group entities with similar features



Recommendation

Recommend an item to a user based on past behavior or preferences of similar users.

Data

User Info+
Your Past Purchase Data+
Purchase of other user+
Product Info

Recommendation ML Method

Classifier

Matrix

Recommendations

YMAL

Classical Machine Learning vs. Deep Learning

Classical Machine Learning

Step 1: Determine features.
Step 2: Feed them through model.

Feature Detection

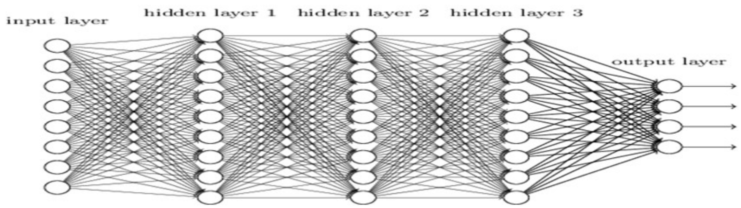
Machine Learning Classifier Algorithm

“Arjun"

Deep Learning

Steps 1 and 2 are combined into 1 step.

Neural Network



“Arjun"

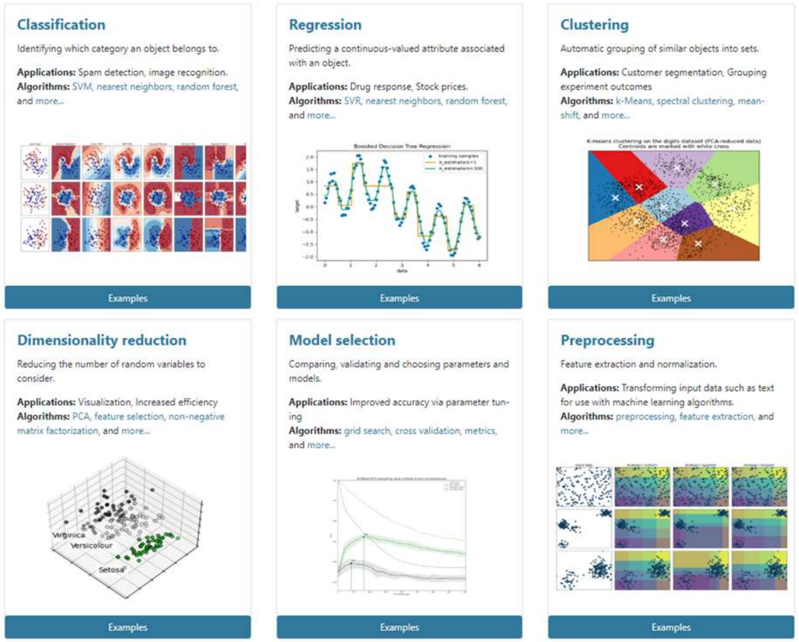
Intel® oneAPI AI Analytics Toolkit

Data Analytics & Machine Learning

Common Data Analytics & Machine Learning Python Libraries

NumPy SciPy pandas Numba

scikit learn



dmlc XGBoost

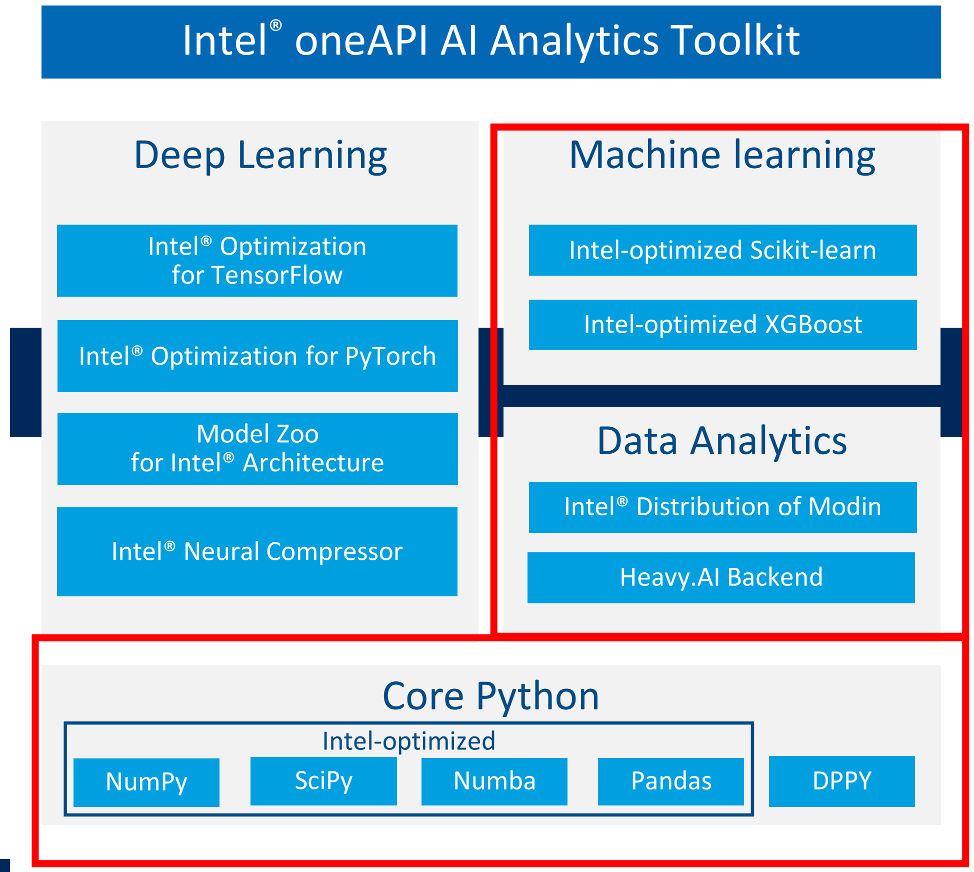
Intel® oneAPI AI Analytics Toolkit

Accelerates end-to-end Machine Learning and Data Analytics pipelines with frameworks and libraries optimized for Intel® architectures

Who Uses It?

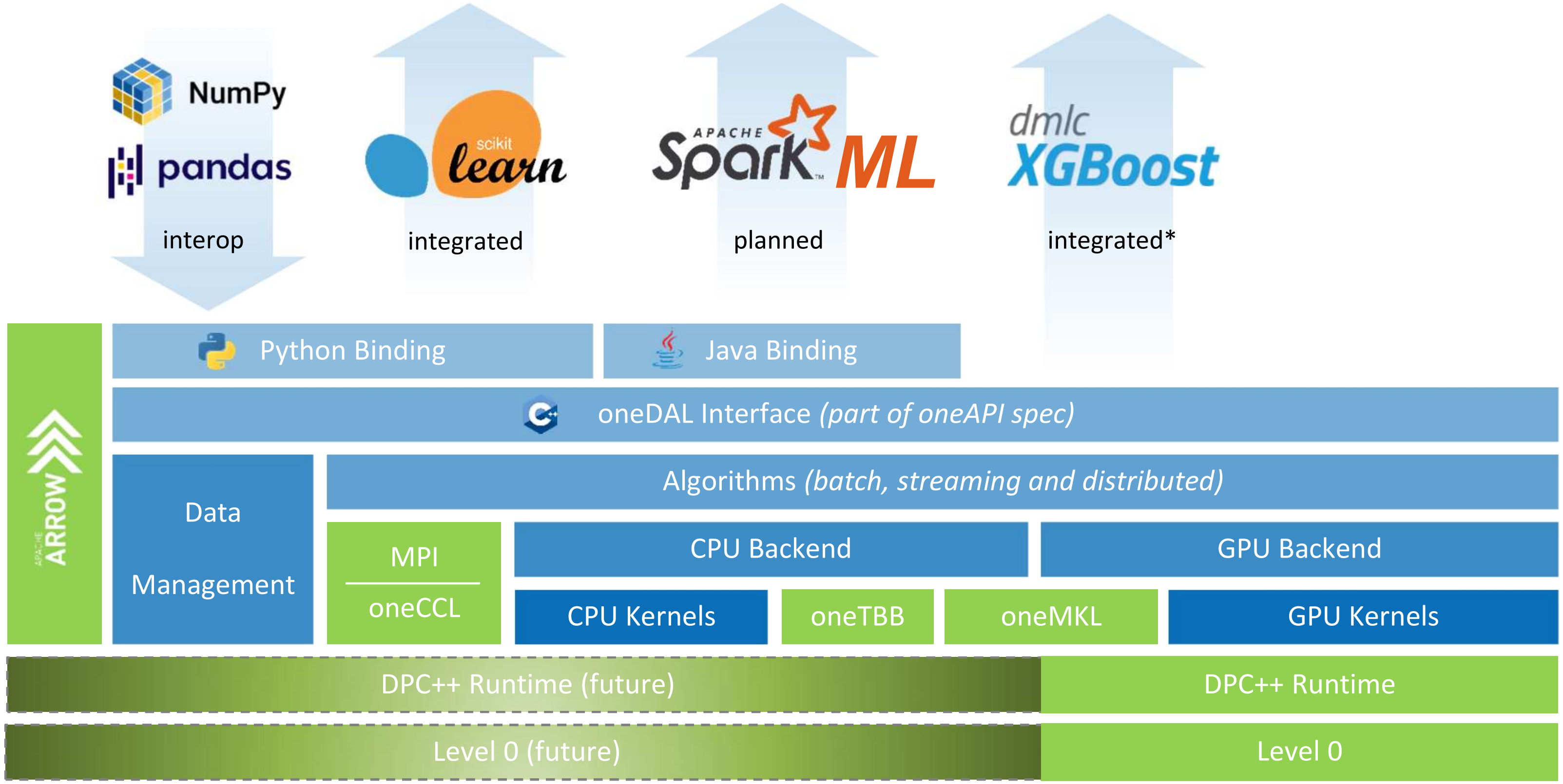
Data scientists, AI Researchers, Machine and Deep Learning developers, AI application developers

Learn More: intel.com/oneAPI-AIKit

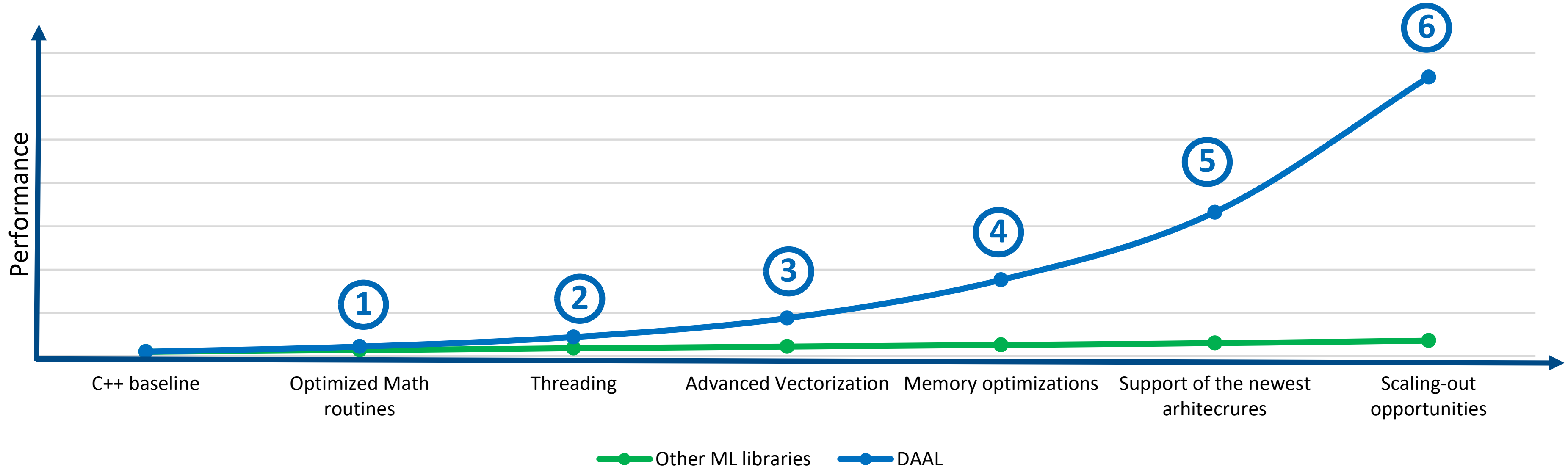


Intel®oneAPI Data Analytics Library (oneDAL)

Framework Interfaces & Software Stack



What makes oneDAL faster?



1 The best performance on Intel Architectures with oneMKL vs. less performance OS BLAS/LAPACK libs

2 onDAL 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, intrinsics. Usually other ML libs build application without vector-instructions support or sse4.2 only.

4 oneDAL's 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 lunch. Usually other ML libs do this with long delay.

6 oneDAL provides distributed algorithms which scale on many nodes

ML Performance with Intel-optimized scikit-learn *

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

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

Common Scikit-learn (mainline)

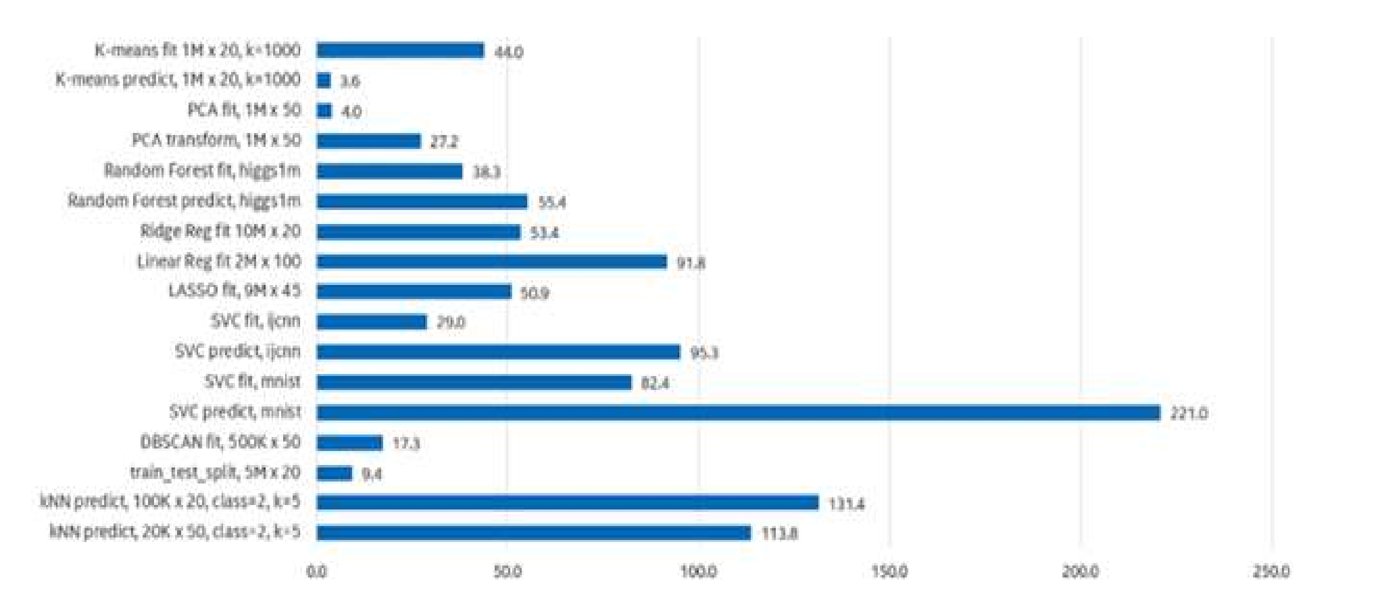
```python
from sklearnex import patch_sklearn
patch_sklearn()

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

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

Scikit-learn on Intel CPU optimized by Intel® oneAPI AI Analytics Toolkit

Stock scikit-learn vs Intel-optimized scikit-learn



Easy as adding two lines of code

*Measured March 2021

ML Performance with Intel-optimized XGBoost *

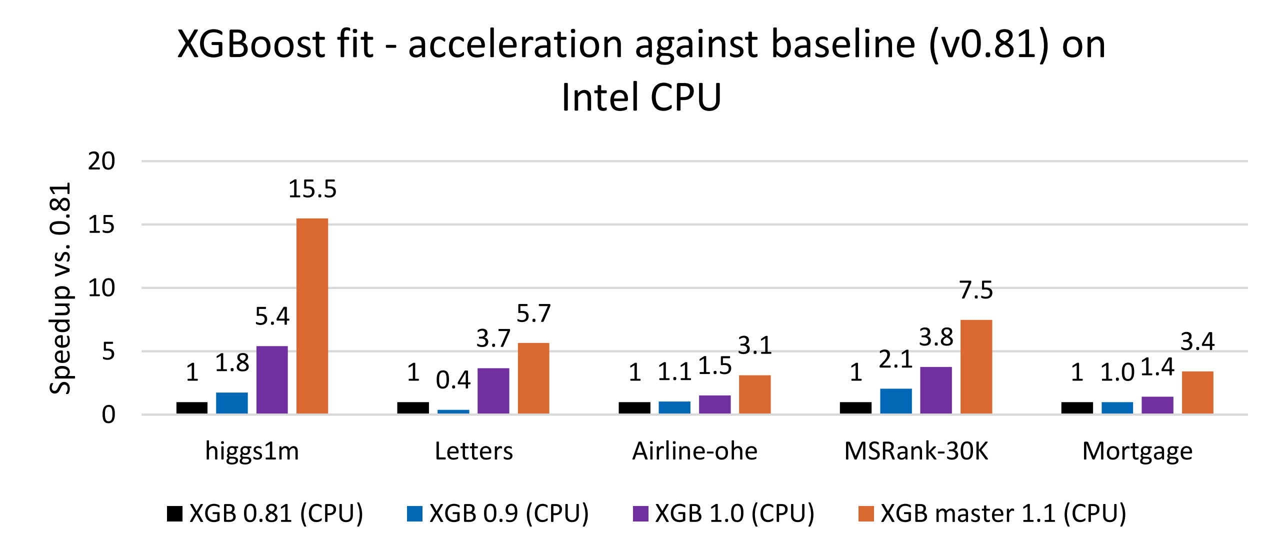
- Intel's contribution to XGBoost project on GitHub https://github.com/dmlc/xgboost
- Memory prefetching, nestled and advanced parallelism, usage of uint8

+ Reducing memory consumption

memory, Kb	Airline	Higgs1m
Before	28311860	1907812
#5334	16218404	1155156
reduced:	1.75	1.65

*Measured March 2021

XGBoost fit CPU acceleration ("hist" method)

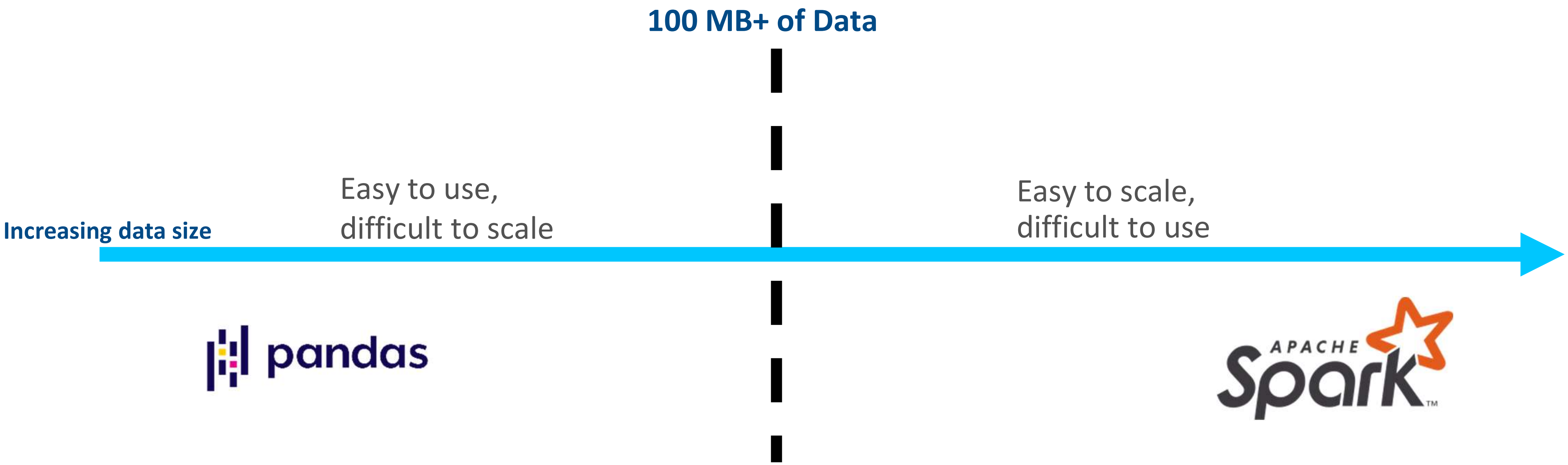


Intel® Distribution of Modin*

1 Line of Code. Infinite Scalability.

Current Data Loading & ETL Landscape

After a certain data size, need to change your API to handle more data



With Modin, use the same API no matter the scale

```python
import pandas as pd
```

MODIN

Easy to use,
Easy to scale

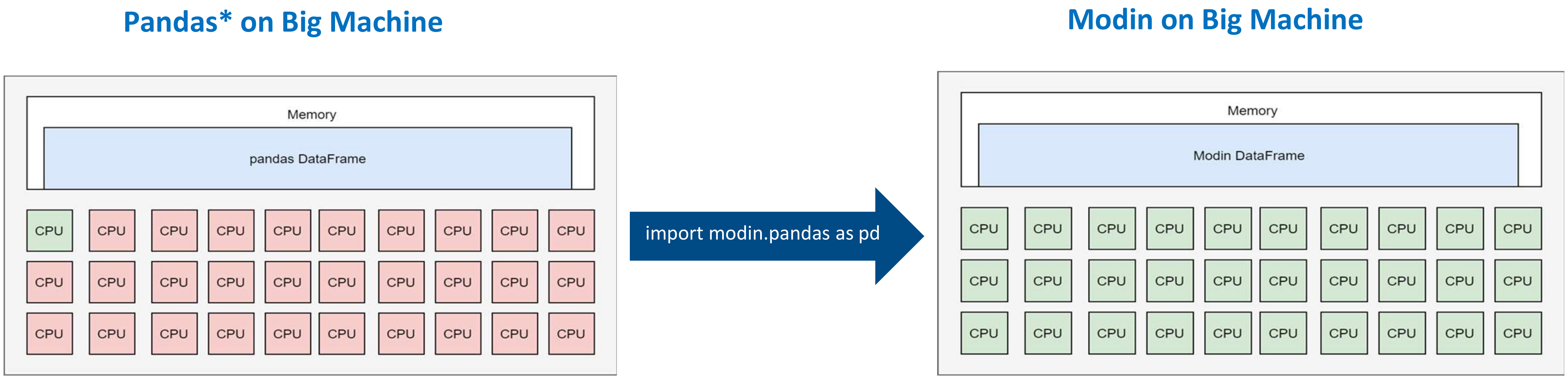
Increasing data size

pandas

0-1TB+ of Data

Modin: How it Works

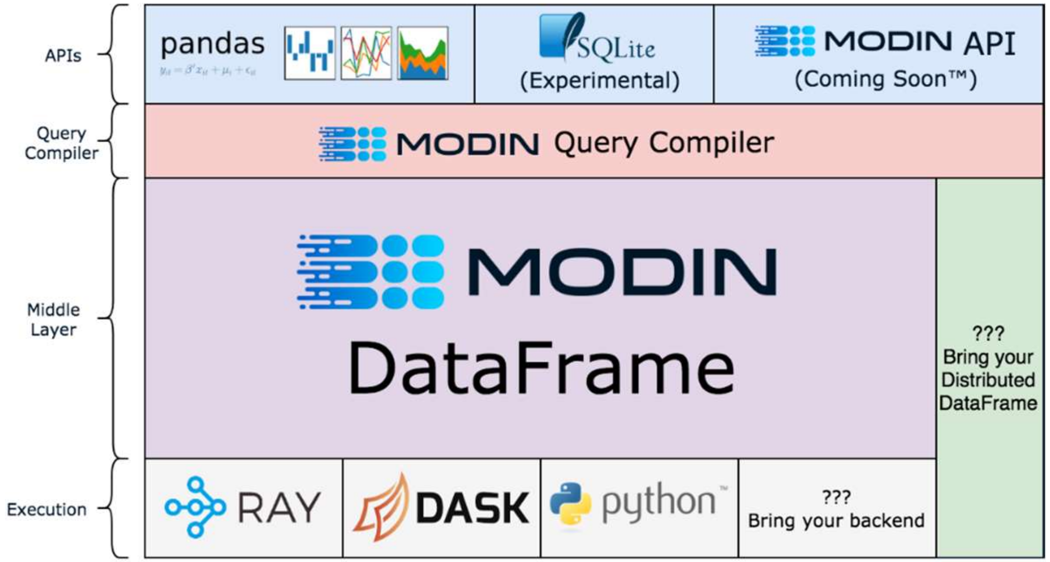
- Modin transparently **distributes the data and computation across available cores**, unlike Pandas which only uses one core at a time
- To use Modin, you **do not need to know how many cores your system has**, and you do not need to specify how to distribute the data



High-Level Architectural View

- **Ray Backend** (most recommended) – The Ray* backend is the recommended backend engine for Intel® Distribution of Modin. It has the most Pandas API functionality enabled as well as the most stable implementation with Intel® Distribution of Modin.
- **Dask Backend** – The Dask* backend is recommended for workloads running on Windows operating systems, Intel® DevCloud for oneAPI.
- **OmniSci Backend** – In partnership OmniSci* (now Heavy.AI)*, Intel® Distribution of Modin supports the OmniSci as a backend, a very performant framework for end-to-end analytics that has been optimized to harness the computing power of existing and emerging Intel® hardware. Please note that this backend is currently experimental.

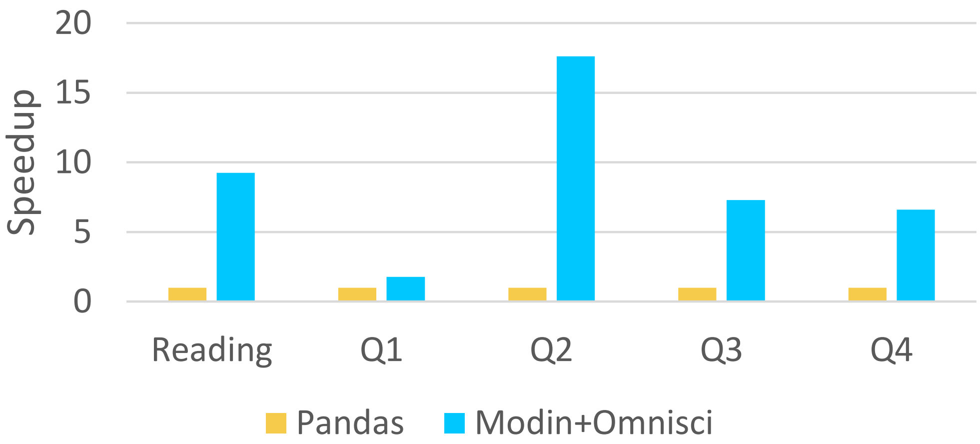
MODIN



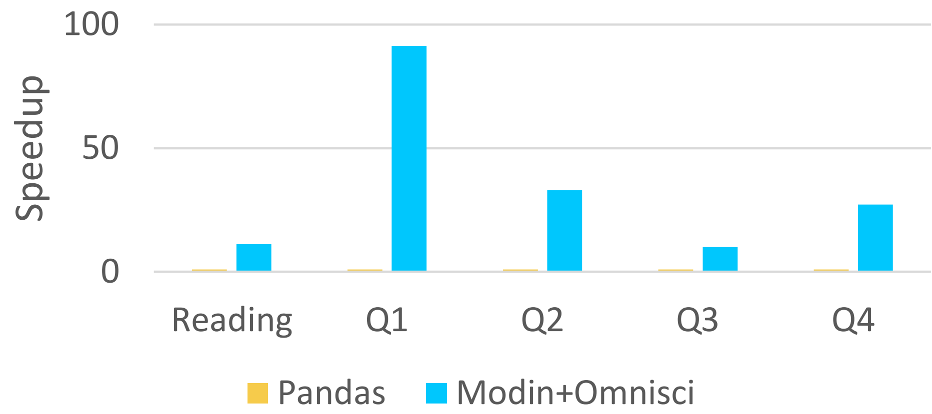
NYCTaxi Workload Performance

Pandas vs Modin – Higher is Better

NYCTaxi **(20 Million rows)** - Performance improvement with Modin+Omnisci



NYCTaxi **(1 Billion rows = 1.6 TB in mem)** - Performance improvement with Modin+Omnisci – using 3TB Optane



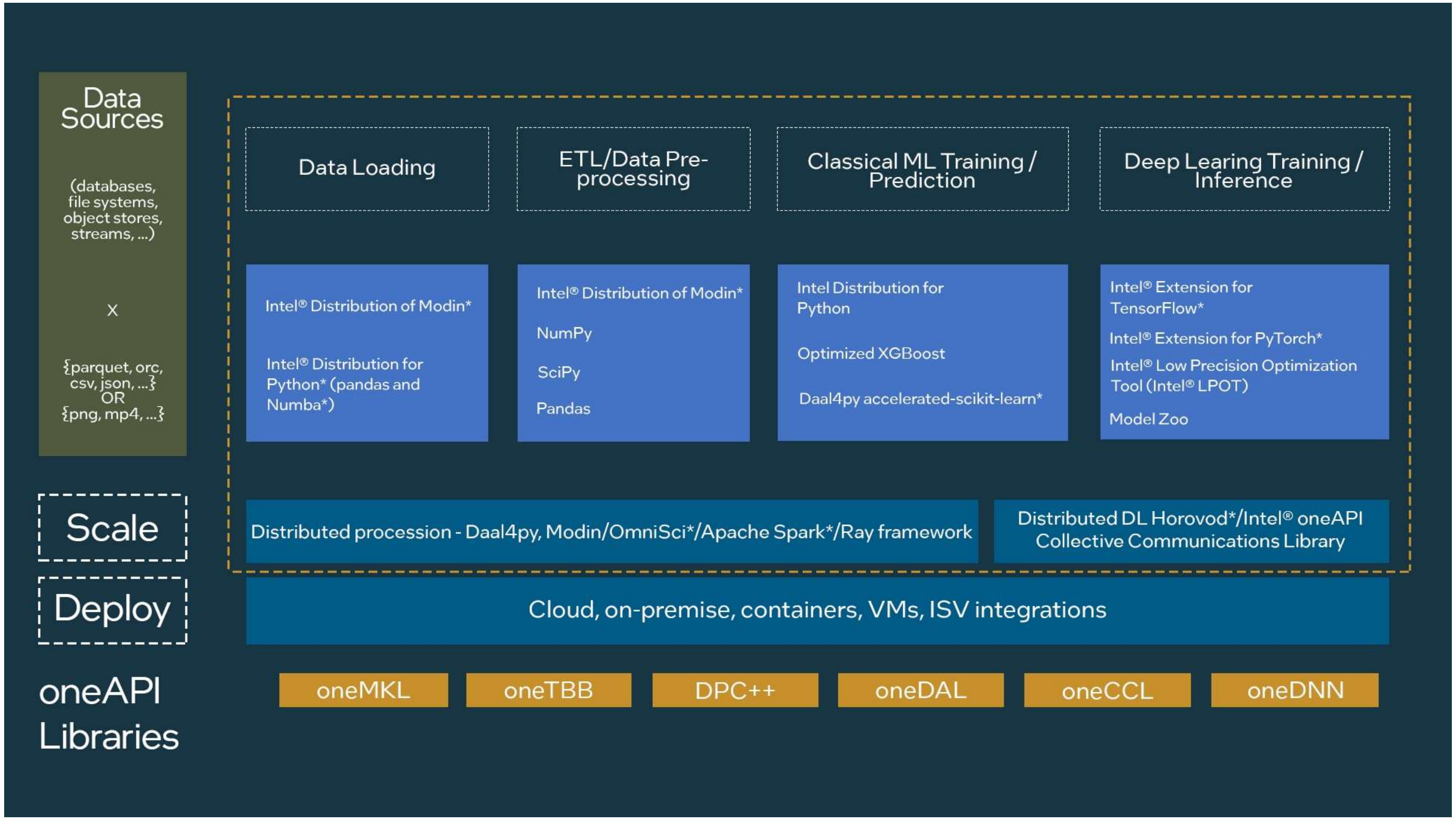
Dataset source: https://github.com/toddwschneider/nyc-taxi-data

Configurations: For 20 million rows: Dual socket Intel(R) Xeon(R) Platinum 8280L CPUs (S2600WFT platform), 28 cores per socket, hyperthreading enabled, turbo mode enabled, NUMA nodes per socket=2, BIOS: SE5C620.86B.02.01.0013.121520200651, kernel: 5.4.0-65-generic, microcode: 0x4003003, OS: Ubuntu 20.04.1 LTS, CPU governor: performance, transparent huge pages: enabled, System DDR Mem Config: slots / cap / speed: 12 slots / 32GB / 2933MHz, total memory per node: 384 GB DDR RAM, boot drive: INTEL SSDSC2BB800G7. For 1 billion rows: Dual socket Intel Xeon Platinum 8260M CPU, 24 cores per socket, 2.40GHz base frequency, DRAM memory: 384 GB 12x32GB DDR4 Samsung @ 2666 MT/s 1.2V, Optane memory: 3TB 12x256GB Intel Optane @ 2666MT/s, kernel: 4.15.0-91-generic, OS: Ubuntu 20.04.4

Demo

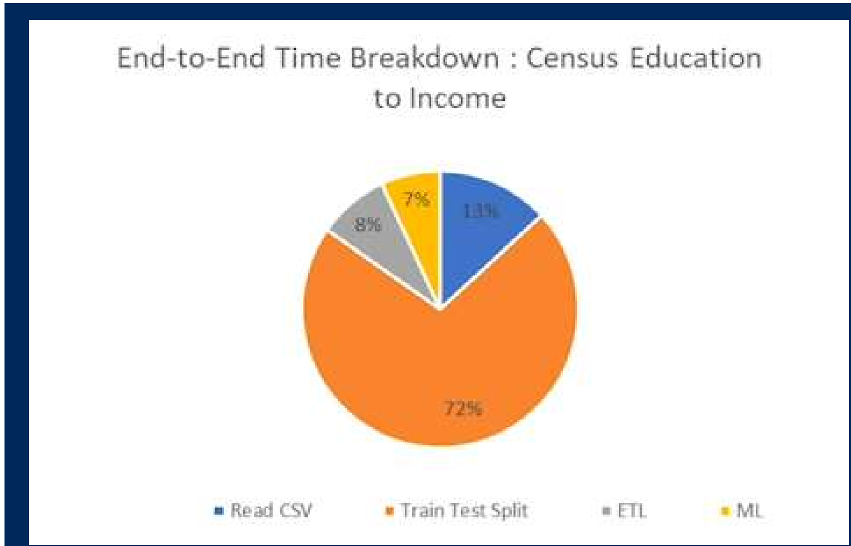
End-to-End Pipelines for AI and Machine Learning Applications

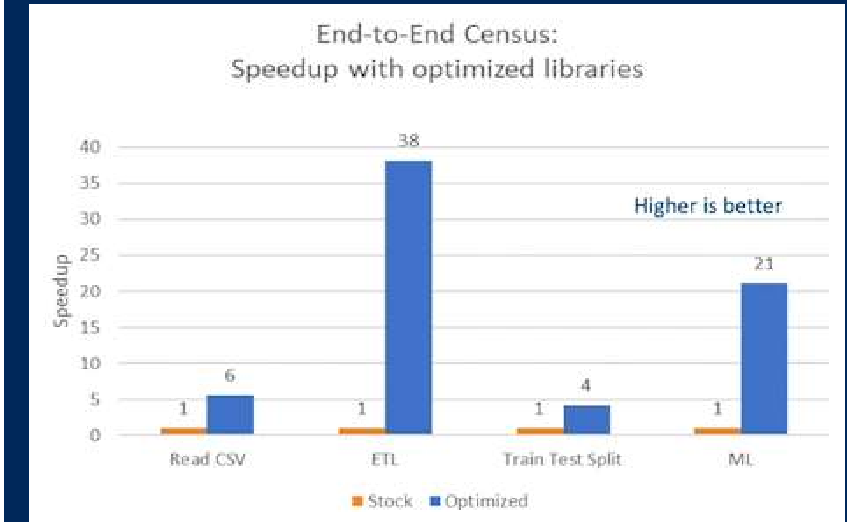
with Intel® optimization powered by oneAPI



End-to-End Data Pipeline Acceleration *

- **Workload:** Train a model using 50 years of Census dataset from IPUMS.org to predict education level.
- **Solution:** Intel Modin for data ingestion and ETL, Intel scikit-learn for model training and prediction
- **Performance Gains**
  - Read_CSV (Read from disk and store as a dataframe): **6x**
  - ETL operations: **38x**
  - Train Test Split: **4x**
  - ML training (fit & predict) with Ridge Regression: **21x**





*Measured March 2021

End-to-End Data Pipeline Acceleration

Minimal Code Changes to Original Census Workload for Intel Optimizations:

1 line of code change Intel® Distribution of Modin:

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

2 lines added for Intel® Extension for Scikit-Learn*:

```
from sklearnex import patch_sklearn
patch_sklearn()
from sklearn.model_selection import train_test_split
from sklearn.linear_model import lm
```

End-to-End Census Sample

master

oneAPI-samples / AI-and-Analytics / End-to-end-Workloads / **Census** /

amyskov and **JoeOster** Correct modin samples (#768) ...

..	
.gitkeep	new folder structures for AI toolkit samples from Saumya (#198)
Expected_output.jpg	Sample code for the end-to-end workload (Census) added to the fork. T...
License.txt	Revert "Remove license files from all samples, except root" (#404)
README.md	AI-samples-Readme-Updates (#848)
Running_Jupyter_notebook.jpg	Sample code for the end-to-end workload (Census) added to the fork. T...
Running_Jupyter_notebook_as_Python.jpg	Sample code for the end-to-end workload (Census) added to the fork. T...
census_modin.ipynb	Correct modin samples (#768)
sample.json	Correct modin samples (#768)
third-party-programs.txt	Add Third Party License (#449)

https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/End-to-end-Workloads/Census

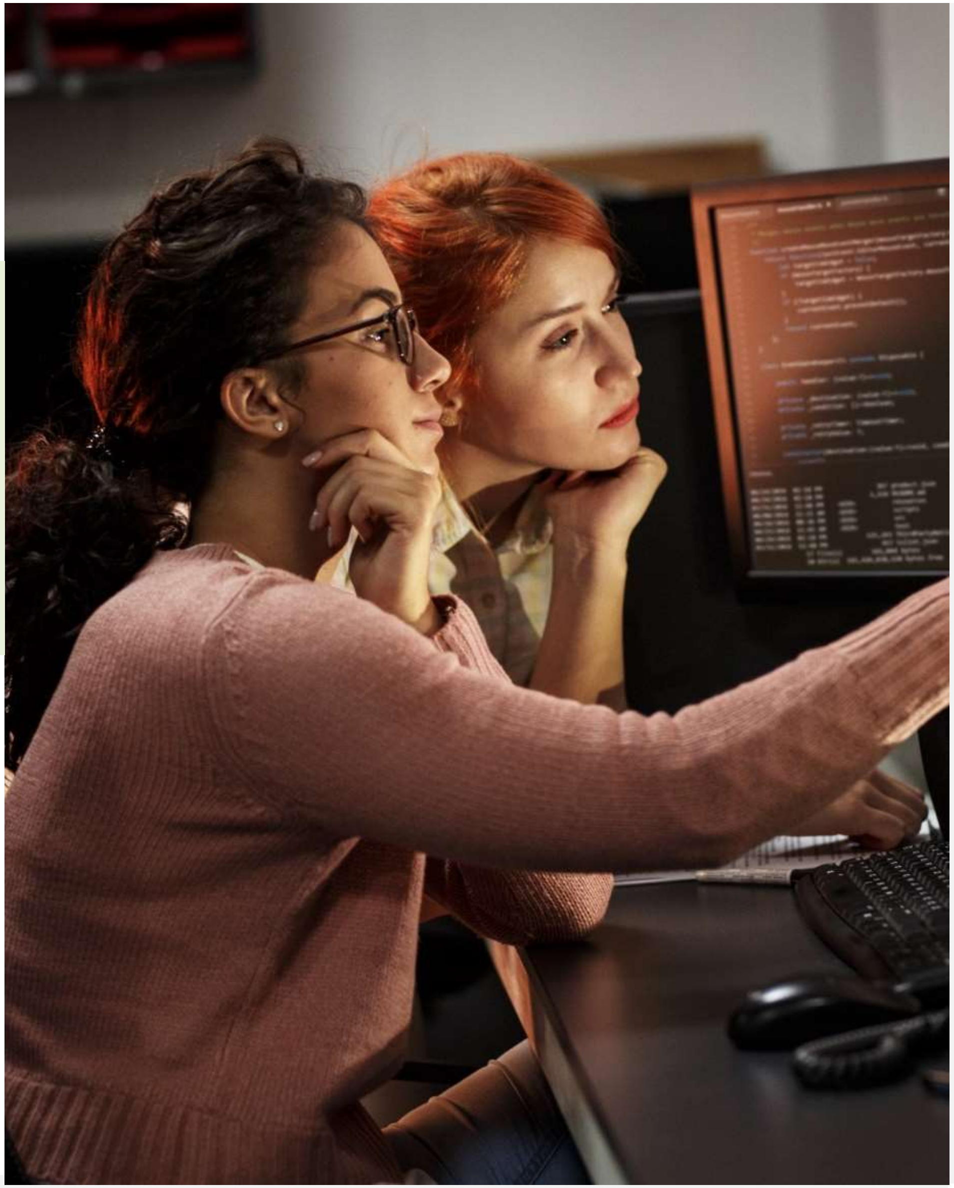
Key Takeaways & Call to Action

- Intel toolkits are FREE, complementary & work seamlessly together
- They help achieve performance & efficiency across different stages of AI Journey
- Recommend the toolkits based on current phase of customer pipeline

Download the toolkit:

Intel® oneAPI AI Analytics Toolkit

Learn more about Intel® oneAPI Toolkits
intel.com/oneAPI-AllToolkits



Resources

- Intel® Distribution for Python*
- Intel® Extension for Scikit-Learn* Getting Started Guide
- XGBoost Optimized for Intel® Architecture Getting Started Guide
- Intel® Distribution of Modin Getting Started Guide

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.

Configurations

Deep Learning Training and Inference Performance using Intel® Optimization for PyTorch with 3rd Gen Intel® Xeon® Scalable Processors
ResNet50/ResNext101 (FP32/BF16): batch size 128/instance, 4 instances.
ResNet50/ResNext101 dataset (FP32/BF16): ImageNet Dataset
DLRM batch size (FP32/BF16): 2K/instance, 1 instance
DLRM dataset (FP32/BF16): Criteo Terabyte Dataset
DLRM batch size (INT8): 16/instance, 28 instances, dummy data.
Tested by Intel as of 6/2/2020.
Intel® Xeon® Platinum 8380H Processor, 4 socket, 28 cores HT On Turbo ON Total Memory 768 GB (24 slots/ 32GB/ 3200 MHz), BIOS: WLYDCRB1.SYS.0015.P96.2005070242 (ucode: 0x700001b), Ubuntu 20.04 LTS, kernel 5.4.0-29-generic
PyTorch: https://github.com/pytorch/pytorch.git
Intel Extension for PyTorch: https://github.com/intel/intel-extension-for-pytorch.git
gcc: 8.4.0,
Intel® oneAPI Deep Neural Network Library (oneDNN) version: v1.4
ResNet50: https://github.com/intel/optimized-models/tree/master/pytorch/ResNet50
ResNext101 32x4d: https://github.com/intel/optimized-models/tree/master/pytorch/ResNext101_32x4d
DLRM: https://github.com/intel/optimized-models/tree/master/pytorch/dlrm

Inference Throughput FP32 vs Int8 optimized by Intel® Optimization for Tensorflow and Intel® Neural Compressor (part of the Intel® oneAPI AI Analytics Toolkit)
Tested by Intel as of : 10/26/2020: TensorFlow v2.2 (https://github.com/Intel-tensorflow/tensorflow/tree/v2.2.0); Compiler: GCC 7.2.1; DNNL(https://github.com/oneapi-src/oneDNN) v1.2.0 75d0b1a7f3586c212e37acebbb8acd221cee7216; Dataset: ImageNet/Coco/Dummy, refer to each model README; Precision: FP32 and Int8
Platform: Intel® Xeon® Platinum 8280 CPU; #Nodes: 1; #Sockets: 2; Cores/socket: 28; Threads/socket: 56; HT: On; Turbo: On; BIOS version: SE5C620.86B.02.01.0010.010620200716; System DDR Mem Config: 12 slots / 16GB / 2933; OS: CentOS Linux 7.8; Kernel: 4.4.240-1.el7.elrepo.x86_64

Stock scikit-learn vs Intel-optimized scikit-learn
Testing by Intel as of 10/23/2020. Intel® oneAPI Data Analytics Library 2021.1 (oneDAL), scikit-learn 0.23.1, Intel® Distribution for Python 3.8; Intel® Xeon® Platinum 8280LCPU @ 2.70GHz, 2Sockets, 28 cores per socket, 10M samples, 10 features, 100 clusters, 100 iterations, float32

XGBoost fit CPU acceleration

Test configs: Tested by Intel as of 10/13/2020; c5.24xlarge AWS Instance, CLX 8275 @ 3.0GHz, 2 sockets, 24 cores per socket, HT:on, DRAM (12 slots / 32GB / 2933 MHz); SW: XGBoost 0.81, 0.9, 1.0 and 1.1:build from sources. compiler – G++ 7.4, nvcc 9.1. Intel® DAAL: 2019.4 version; Python env: Python 3.6, Numpy 1.16.4, Pandas 0.25, Scikit-lean 0.21.2.

End-to-End Census Workload Performance

Tested by Intel as of 10/15/2020. 2x Intel® Xeon® Platinum 8280 @ 28cores, OS: Ubuntu 19.10.5.3.0-64-generic Mitigated, 384GB RAM. SW: Modin 0.8.1, scikit-learn 0.22.2, Pandas 1.0.1, Python 3.8.5, Daal4Py 2020.2 Census Data, (21721922, 45). Dataset is from IPUMS USA, University of Minnesota, www.ipums.org . Version 10.0.

Tiger Lake + Intel® Distribution of OpenVINO™ toolkit vs Coffee Lake CPU

System Board	Intel prototype, TGL U DDR4 SODIMM RVP	ASUSTeK COMPUTER INC. / PRIME Z370-A
CPU	11th Gen Intel® Core™ -5-1145G7E @ 2.6 GHz.	8th Gen Intel® Core™ i5-8500T @ 3.0 GHz.
Sockets / Physical cores	1 / 4	1 / 6
HyperThreading / Turbo Setting	Enabled / On	Na / On
Memory	2 x 8198 MB 3200 MT/s DDR4	2 x 16384 MB 2667 MT/s DDR4
OS	Ubuntu* 18.04 LTS	Ubuntu* 18.04 LTS
Kernel	5.8.0-050800-generic	5.3.0-24-generic
Software	Intel® Distribution of OpenVINO™ toolkit 2021.1.075	Intel® Distribution of OpenVINO™ toolkit 2021.1.075
BIOS	Intel TGLIFUI1.R00.3243.A04.2006302148	AMI, version 2401
BIOS release date	Release Date: 06/30/2020	7/12/2019
BIOS Setting	Load default settings	Load default settings, set XMP to 2667
Test Date	9/9/2020	9/9/2020
Precision and Batch Size	CPU: INT8, GPU: FP16-INT8, batch size: 1	CPU: INT8, GPU: FP16-INT8, batch size: 1
Number of Inference Requests	4	6
Number of Execution Streams	4	6
Power (TDP Link)	28 W	35W

intel®