STORAGE DEVELOPER CONFERENCE
SDC 22
Fremont, CA
September 12-15, 2022
BY Developers FOR Developers
A SNIA Event

Data Platform For End-to-End AI Democratization

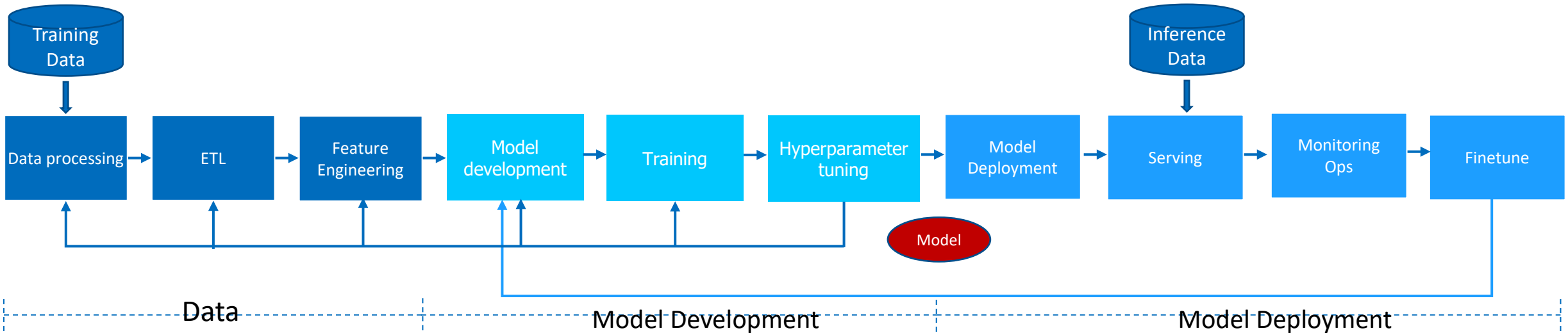
Jack Zhang, System Architect, Intel
Jian Zhang, AI software engineering manager, Intel

Agenda

- Backgrounds and motivation
- AI Democratization
- Data Platform for E2E AI democratization
- Intel® End-to-End AI Optimization Kit
- Performance results
- Summary

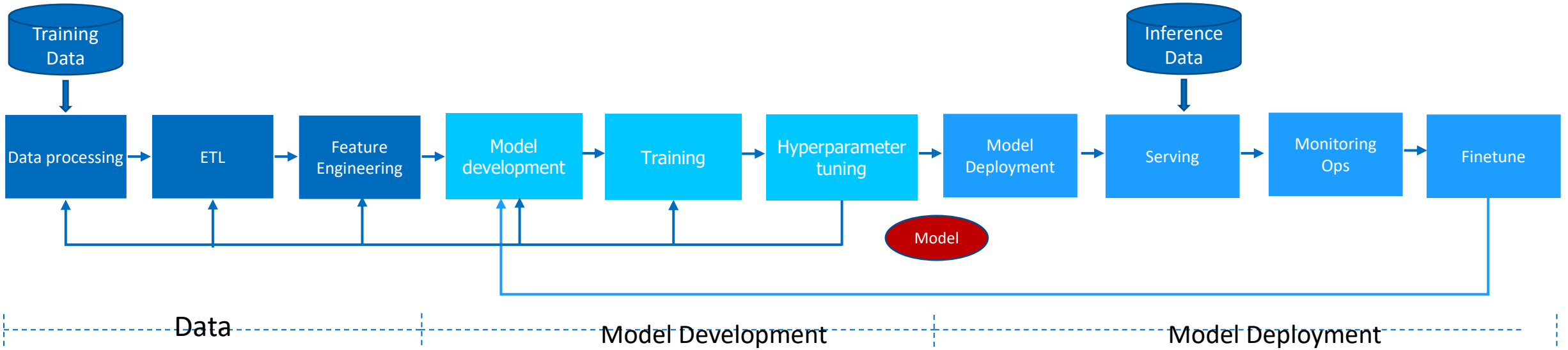
Background and Motivation

Life cycle of End-to-End AI



- E2E AI life cycle
  - data processing, feature engineering, training, model serving
  - Iterative process – single stage might be gone through for several times
  - ML code is a small part of real-world AI system
    - A vast-complex pipeline/infrastructure is required for AI to deliver successful predictive results/values

Challenges



stacks	Kafka*, Sqoop*, Flume*, Spark*, Nifi	Spark, Pandas,Numpy Spark Mllib, scikit-learn, XGBoost	Tensorflow, pytorch, Keras, Cafee, MXNet	Serving	MLflow
challenges	Data growing at extreme pace – 61% annually worldwide[1] Huge dataset - TB scale, ETL takes too much time, quite often could be longer than training	Data scientists spend ~40% of their time gathering and cleaning data[2] Scalable data process Iterative data processing create numerous sets of feature	training as core part of AI High expertise requirement Extremely compute intensive	No one-fit-for-all solution Integration with existing env Hardware-restrict env Realtime inference & retrain	Keep track on all the experiments, tests and models Automation, improvement, monitoring, logging

E2E AI is challenging - A complex process requires multiple roles with different skills involved: product manager, developer, infrastructure engineer, data scientists, operation engineer…

1. https://www.networkworld.com/article/3325397/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html
2. https://businessoverbroadway.com/2019/02/19/how-do-data-professionals-spend-their-time-on-data-science-projects/

Motivations – Data platform for End-to-End AI democratization

- Unified Data platform for data analytics, training and inference pipeline
  - in-situ data processing, training and inference to avoid massive data movement
  - Eliminate data silo, improve resource utilization
  - Unified pipeline, simplified software stacks
- Simplified model development and optimization with AI Democratization
  - Built-in high-performance models or tunings to reduce model optimization cost
  - AutoML and Automatic network construction (Neural architecture search) for popular domains
  - Access to a broad range of features/AI algorithms
- Bring End-to-End AI to commodity hardware
  - Scalable AI development, deployment on multi-node clusters
  - Reduce model development and training computational cost

AI democratization

End-to-End AI democratization

- Goal: Make AI **accessible** & **affordable** to everyone
  - AI restricted to data scientists and data analysts who are specifically trained in AI -> expand it to citizen data science users
  - AI requires high-end hardware -> make it scalable to commodity hardware

• Enabling scale-up/out commodity HW with optimized framework and toolkits • Delivering popular lighter DL Models with close enough AUC and higher inference throughput • Reducing E2E time (including training) on CPU to acceptable range through full pipeline optimization	**Making AI Faster**
• Automating E2E democratization pipeline with click to run workflows and AutoML • Abstracting complex API for data processing, feature engineering, simplifying distributed training • Providing independent pluggable components complements current toolkits and easily integrated with 3rd party ML solutions/platforms	**Making AI Easier**
• Bring complex DL to commodity HW – commodity CPU & ethernet • Built-in democratized models through parameterized models generated by smart democratization advisor (SDA) • Domain specific compact networks searched by smart network accelerator (NAS)	**Making AI more accessible**

What to Democratize?

- What to democratize?
  - Data accessibility & quality
    - Make data access easier & simpler & faster
    - Democratize the data management - Data ingestion, data warehouses and data lakes -> explore & visualize & processing
  - Storage & compute platforms
    - The Infrastructure E2E AI runs on: Cloud computing, auto-scaling, GPU vs. CPU
    - Democratize the HW – scalable infrastructure on commodity hardware
  - Algorithms
    - The use, development and sharing of ML & DL algorithms
    - Democratize the algorithms - Reduce entry barriers, automatic model searching, AutoML, improve explanation of the results
  - Model development
    - Select the most suitable models
    - Democratize the end-to-end model development, training and deployment
  - Marketplace
    - Access to the models
    - Democratize the use, exchange, and monetization of data, algorithms, models and outcomes

Data Platform for E2E AI democratization

From case study to Data platform

Data Processing – An RecSys Datasets example

Model	Dataset	Size	Data Schema	Processing Framework	Represented Data
DLRM	**Criteo**	• 1.3TB (24 days)	• 40 columns • 0 – label • 1-14 integer • 15-39 categorical	• pandas, numpy, spark	• Click logs of display ads
DIEN	**Amazon Dataset: books & Electronics**	• 6.2GB+788MB	• Json files • Converted to csv	• Python, Spark, Pandas	• E-commerce: Amazon product review
W&D	**Kaggle outbrain**	• 93GB	• 9 csv files, each have multiple columns	• Spark, NVTabular	• User page view & clicks of multiple publisher sites
RecSys	**Twitter**	• 47GB training • 4.8GB test • 4.8GB val	• tsv files, 23 columns	• Pandas, NVTabular	• Social networks: tweets

RecSys representative datasets (link):

- Fitness tracking dataset (EndoMondo), review data (Amazon, Goggle, Steam games), Social data (Twitter, google), image dataset (Pinterest), Geographical dataset (Google Local)….

Common features:

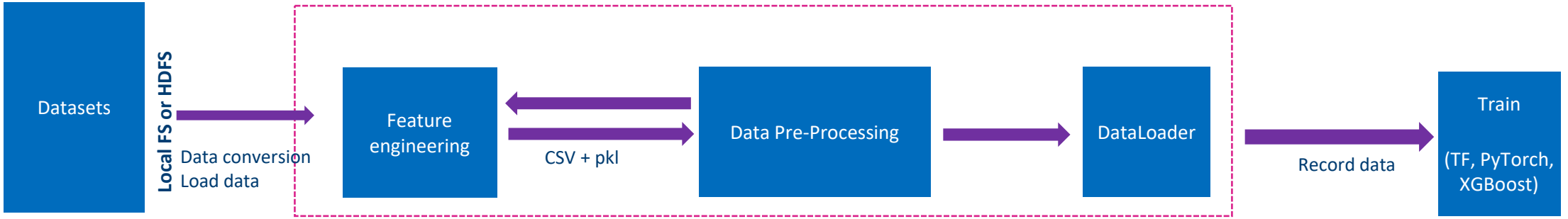
- User/item interactions, star rating, timestamps, product reviews, social networks, item-to-item relationship, images, price/brand/category info, gps data, other metadata

Characteristics

- Huge size – need distributed storage system
- Numerical values or categorical values – can be handled by simple APIs

Observations: Dataset share similar/common characteristics

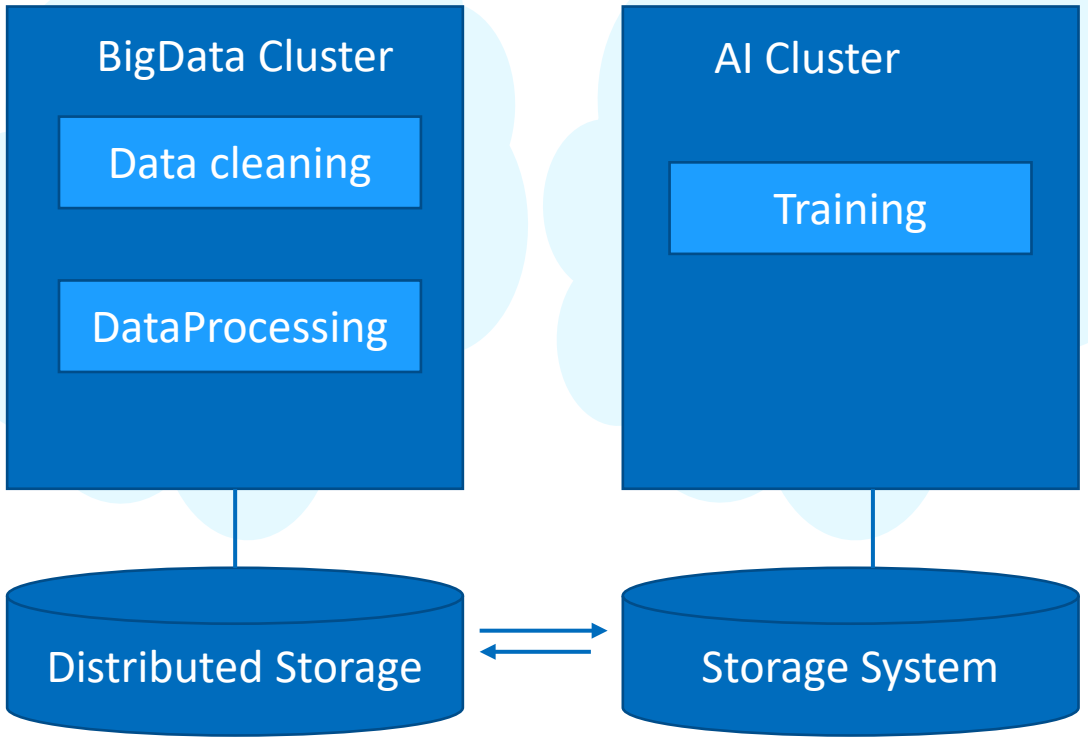
Data Processing - workflows



Model	Dataset	Feature Engineering	Data Pre-Processing	Data Loader
DLRM	**Local FS: csv** **HDFS: csv**	• Load data • Fill missing data • binaries	• Convert & transform numerical & categorical values • Write (convert) processed data to	• Binary files to Pytorch directly
DIEN	**Local FS: json**	• Convert Json to csv • Load csv data • Split • Generate VOC dictionaries from columns	• Join & adding negative records • Aggregation history record	• TFRecord files to TensoFlow
W&D	**Local FS: csv** **HDFS: csv**	• Load separate data files and convert to data frames • Feature engineering with UDFs • compute cosine similarities between features • TF Transformation on RDD	• Drop empty • Fillna	• TFRecord
RecSys	**Local FS: tsv**	• load train,test,val tsv; data type conversion; drop features • Encoding: Target ending, count encoding, difference encoding • building sample csv from validate dataset for submission	• Transforming categorical features from string to integer • Extracting text features from tweets w/ bert tokenizer	• Parquet files -> data frames -> XGboost

Observations: High performance, parallel data processing needed!

Data Connector

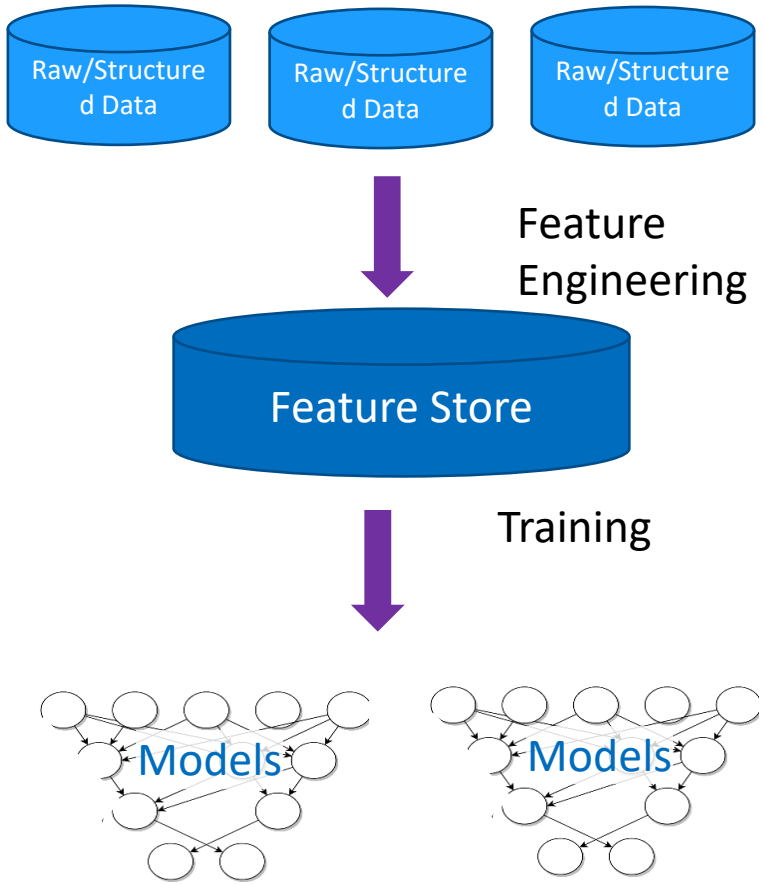


- **Data Access characteristics:**
  - **Dataset Size:** datasets (processed) often exceed the capacity of node-local disk storage, requiring distributed storage systems and efficient network access.
  - **Number of Files:** datasets often consist of billions of files with uniformly random access patterns, something that often overwhelms both local and network file systems.
  - **Data Rates:** training jobs on large datasets often use many GPUs, requiring high aggregate I/O bandwidths to the dataset (many GBytes/s), which can only be satisfied by massively parallel I/O systems.
  - **Shuffling and Augmentation:** training data needs to be shuffled and augmented prior to training.
  - **Scalability:** users often want to develop and test on small datasets and then rapidly scale up to large datasets.
- **Challenges**
  - **Separate cluster** – CPU for data analytics, GPU for training
  - **Data movement issue** – moving huge dataset will be of great cost for training
  - **SW complexity** – need to enable data loading/shuffling functions for DL frameworks

Observations: Separate data analytics and AI cluster creates challenges, a parallel data connector is required

Feature Store

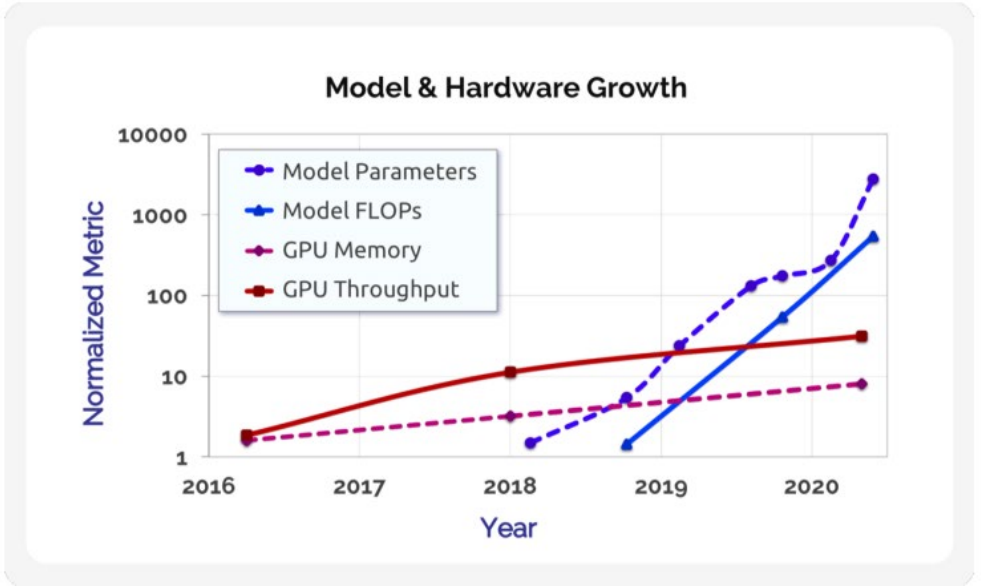
- Challenges
  - Problem: Feature engineering is one of the most time-consuming activities in building an end-to-end ML system.
  - many teams continue to redevelop the same features from scratch for every new project.
- Solution:
  - A centralized feature store to build up a foundation of features can be reused across projects.
- Feature Store
  - Find and share features
  - One work for all: can ensures that the same code used to compute the feature values is used for model training and inference.
  - Save time and cost

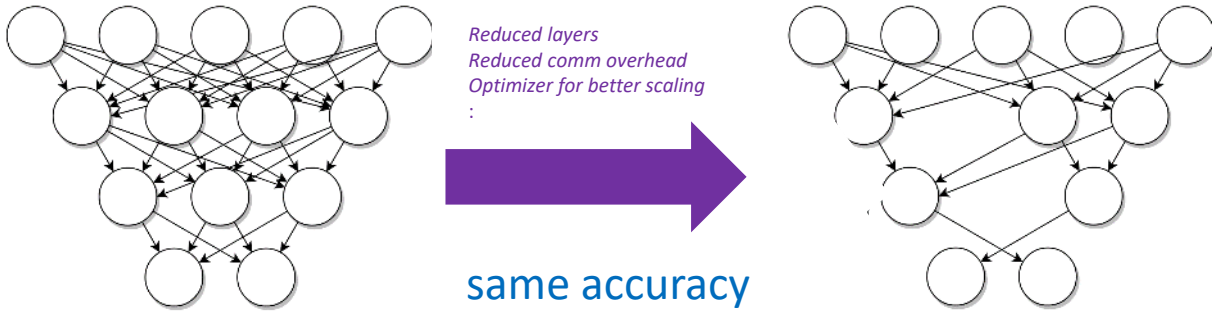


Observations: A feature store bridge data engineering and data science, significantly simplified FE process

Model Democratization

- Overparameterized DL models size & HW growth gap becomes increasingly larger
- Compute intensive model is hard to deploy in resource constraint environment
- Various model optimization technology emerging
  - Compression techniques – pruning/quantization
  - Learning techniques – distillation
  - Automation – HPO, NAS
  - Efficient Architectures – fundamental blocks (convolution layer, attention layer – parameter sharing..)
  - Infrastructure – high performance training & inference frameworks PyTorch & TF – framework level (TFLite, PyTorch Mobile) and optimized it on target HW (i.e. IPEX and ITEX)





- Picture Source: On the Opportunities and Risks of Foundation Models

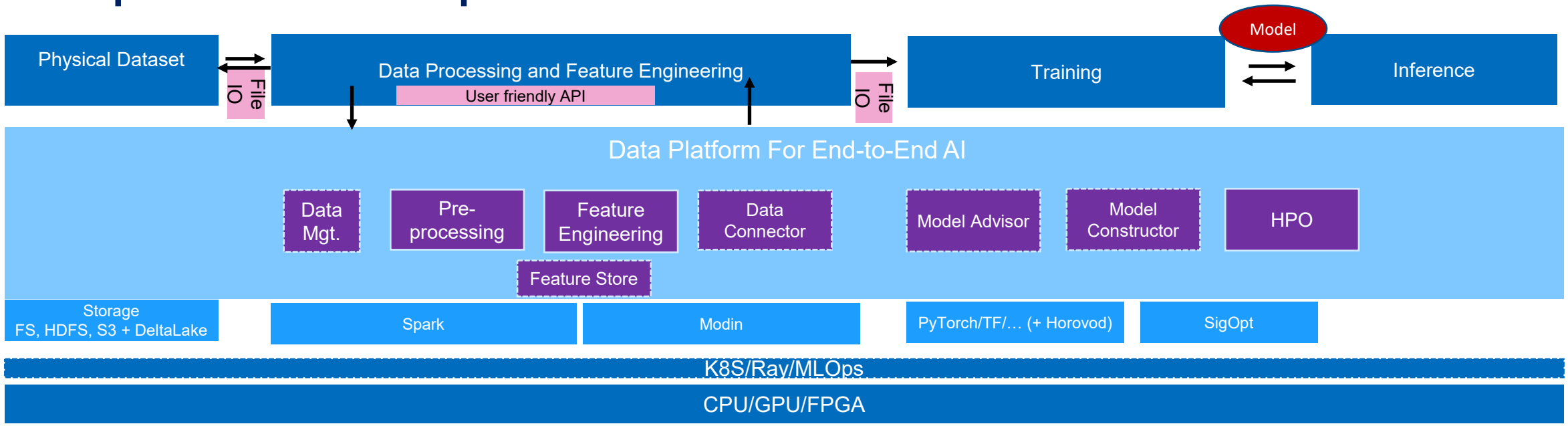
Observations: Overparametrized models needs to be optimized from different perspective!

E2E AI Optimization

- Each AI pipeline includes multiple stage
  - Data processing, feature engineering, training, HPO, inference, finetune…
  - Bottleneck or inefficiency in single stage can impact entire pipeline
- E2E AI optimization
  - A coherent optimization will be needed:
    - Data processing framework optimization: parallel data processing i.e., Modin, Spark with OAP, RecDP
    - Enhanced feature engineering: better encoding algorithms
    - Reusable feature store
    - Distributed training/scaling: enhanced frameworks: Intel PyTorch Extension (IPEX)/Intel TensorFlow Extension (ITEX); multiple-node training; communication optimization
    - Model optimization: compression/pruning/knowledge distillation/NAS
    - Hyper-parameter tuning: LR/step/batch size…

Observations: Democratization needs to be coherent from end-to-end!

Proposed Data platform Architecture



- **Data platform**
  - **A unified, E2E platform to facilitate E2E AI pipeline**
  - **Data Management**
    - Data discovery, exploration, loading of rich data format & semantics from huge datasets
    - Data Model/Data sharding/Data version/intermediate data format
  - **Data Processing**
    - simplified, user-friendly parallel data processing tool to transform data – filtering, normalizing, log transformation…
  - **Feature engineering**
    - high performance
    - rich feature engineering
  - Feature Store
    - Re-usable centralized storage for features

- **Data Connector**
  - Distributed data loader to load data from centralized storage systems
  - optimized shuffling/sampling,
  - customized record-level iterators or data readers plugin
- **Model Advisor**
  - Built-in intelligent for model optimization
  - Limited number of models
  - Click to deploy recipes
- **Model constructor**
  - Constructs compact models directly for popular domains, support a broad range of problems
  - Domain specific supernet
  - Reduced search cost, hardware-aware
- **HPO**
  - Automatic hyper parameter tuning

Intel® End to End AI Optimization Kit

Intel® End to End AI optimization Kit

Positioning

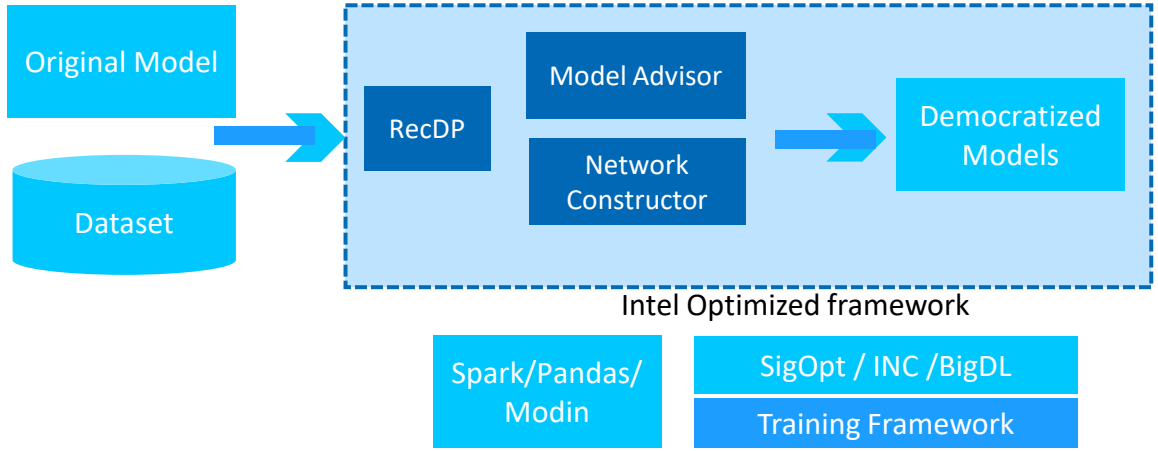
- **A composable framework for AI democratization** in data processing, training and inference, delivering high performance lightweight networks/models efficiently on commodity HW

Objectives

- Bring E2E AI to existing commodity hardware with 10x speedup over vanilla CPU solution and reduce CPU vs. GPU gaps
- Improve citizen data scientists AI accessibility through built-in models and domain-specific auto-searched compact networks
- Composable design that can be easily plugged/integrated into third-party software

Key Features

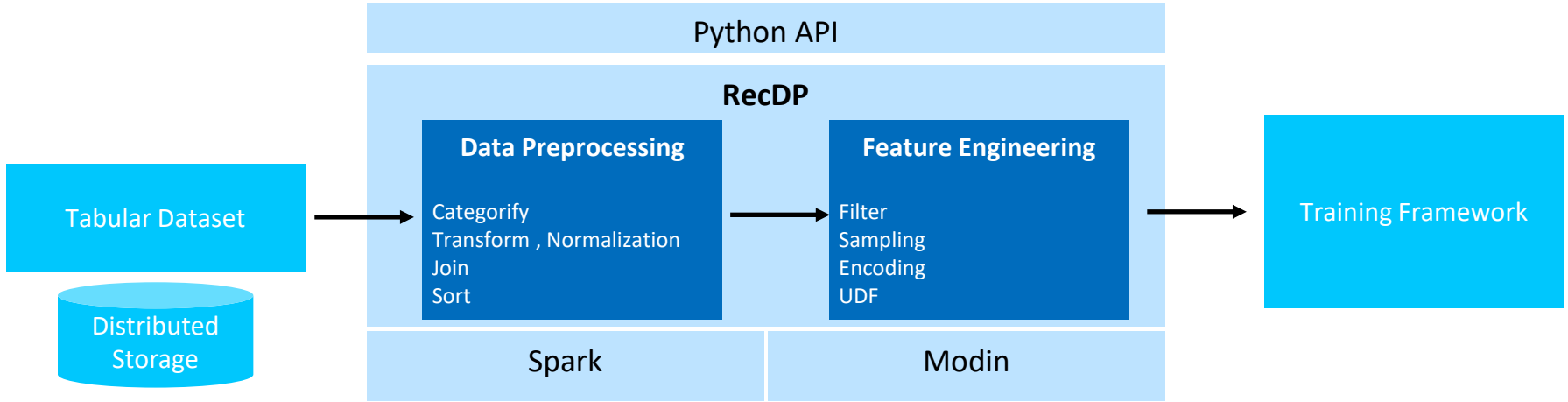
- Parallel large-scale data processing
- Pre-trained lightweight models
- Train-free Neural network constructor
- Click to run, deploy E2E workflow



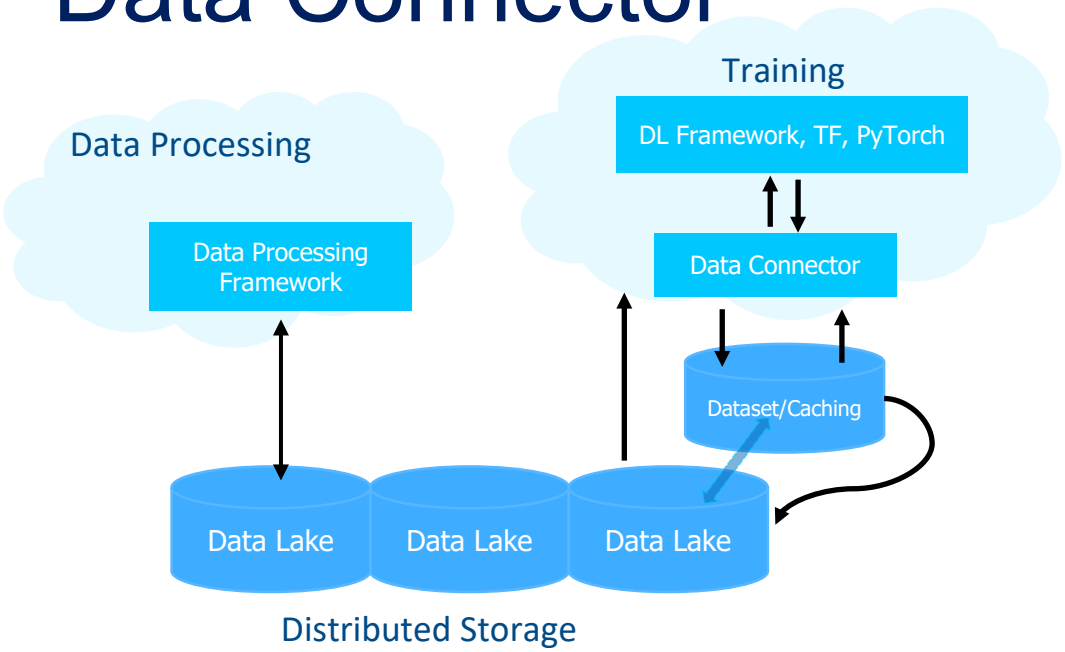
Github: https://github.com/intel/e2eaiok

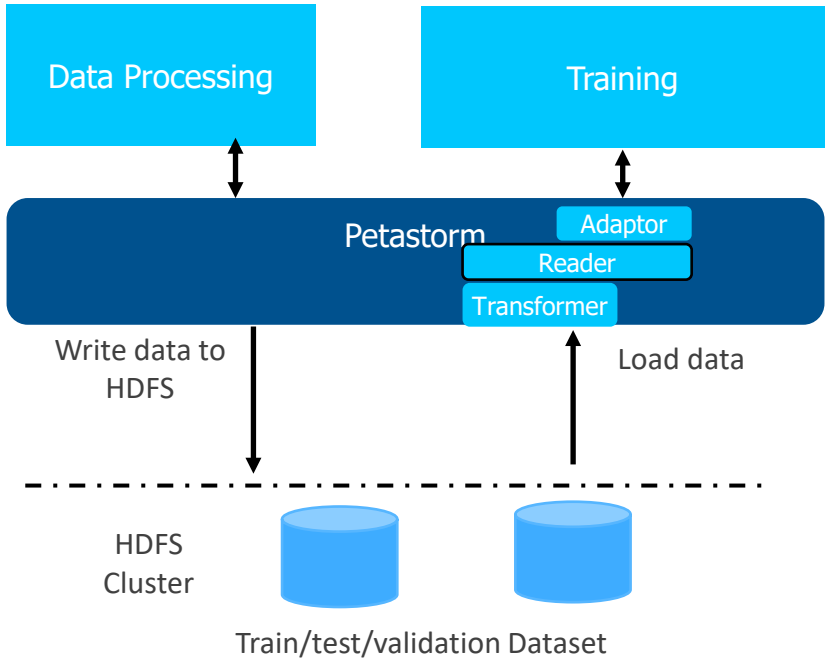
RecDP – Data Processing

- RecDP: Parallel data processing and feature engineering lib
  - Started with Tabular dataset for Recommendation System
  - **Easy-to-use** – simple and easy-to-use APIs for data scientists
  - **Optimized Performance -** adaptive query plan & strategy and OAP extension
  - **Pluggable –** abstraction layer supports different backend (Spark, extensible to Modin)
  - **Integration** –plugin interface to integrated into 3rd party SW with optimized adaptive plan for Data Process pipeline
  - **Feature Engineering** – advanced feature engineering functions (target encoding)



Data Connector



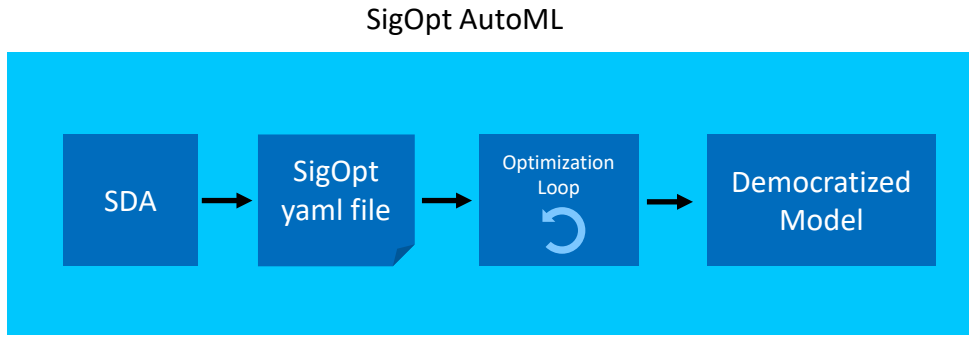


- A parallel data connector
  - Separated data processing and training cluster
  - Distributed data processing/training cluster on multi-nodes CPU/GPU servers
  - Data cannot be fit into single servers, need to be fetched from remote/centralized/disaggregated storage
- Training cluster requires distributed data access
  - High bandwidth data-loading
  - No single point of data aggregation
  - Support local caching to improve performance
  - Support data promotion/flushing

- PetaStorm based Data connector
  - PetaStorm
    - Open-source data access lib enables single/distributed training of deep learning models directly from multi-terabytes dataset in Parquet format
    - It is scalable, support Native TensorFlow & PyTorch, support shuffling, sharding caching etc.
  - Enhancement
    - Raw data loader extension to replace DLRM binary data loader
    - Implemented a method to ensure consistent batch size across ranks
  - Benefits
    - With the help of Petastorm, preprocessed data can be written to/read from HDFS directly without manually split or combination

Model Advisor - Smart Democratization Advisor (SDA)

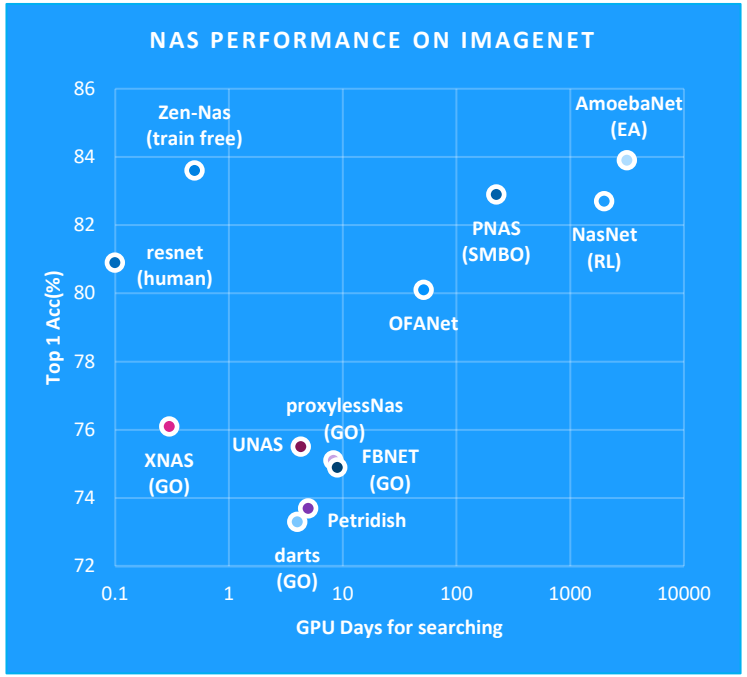
- A user guided tool to facilitate SigOpt recipes generations based with **parameterized models**
  - Generate SigOpt **yaml files** based on user choice
  - Provided build-in intelligence through **parameterized models**
  - Convert manual model tuning to assisted **AutoML**
  - **Automates** model democratization process and experiments tracking with SigOpt
  - **Speedup** efficient frontier (Pareto frontier) search with SigOpt multimetric experiments
  - **Extensible** to integrate other tools



Input	Output
User selections • Dataset • Feature shapes • Models • Parameters	SigOpt Yaml file Features • Targets • Numerical feature columns • Category feature columns Training related • Models • Metrics • Training parameters SigOpt • Experiments budget • …

```
run_aiok.py --data_path ${dataset_path} --model_name [dlrm, wnd, dien, …]
```


Smart Network Constructor



An automatic approach to democratize Deep Neural networks

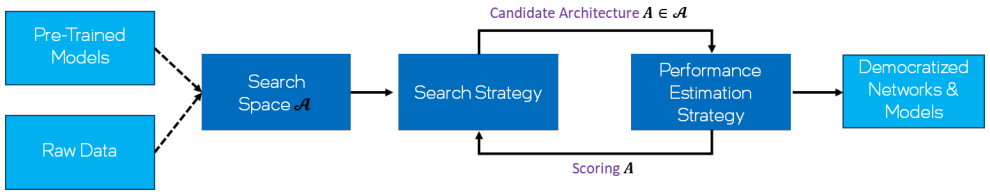
- Compression, HPO cannot cover all the required optimizations for efficient DL
- New technologies emerging to improve DL efficiency
- NAS becomes increasingly popular to design neural networks

NAS Challenges

- Conventional NAS is extremely computation extensive, and GPU based
  - T(NAS) = M * N * T, where M = # of candidates, N= epochs to generate candidates, T = candidate evaluation time
- We need to improve NAS efficiency for automatic AI democratization to construct compact networks

Proxy performance predictor

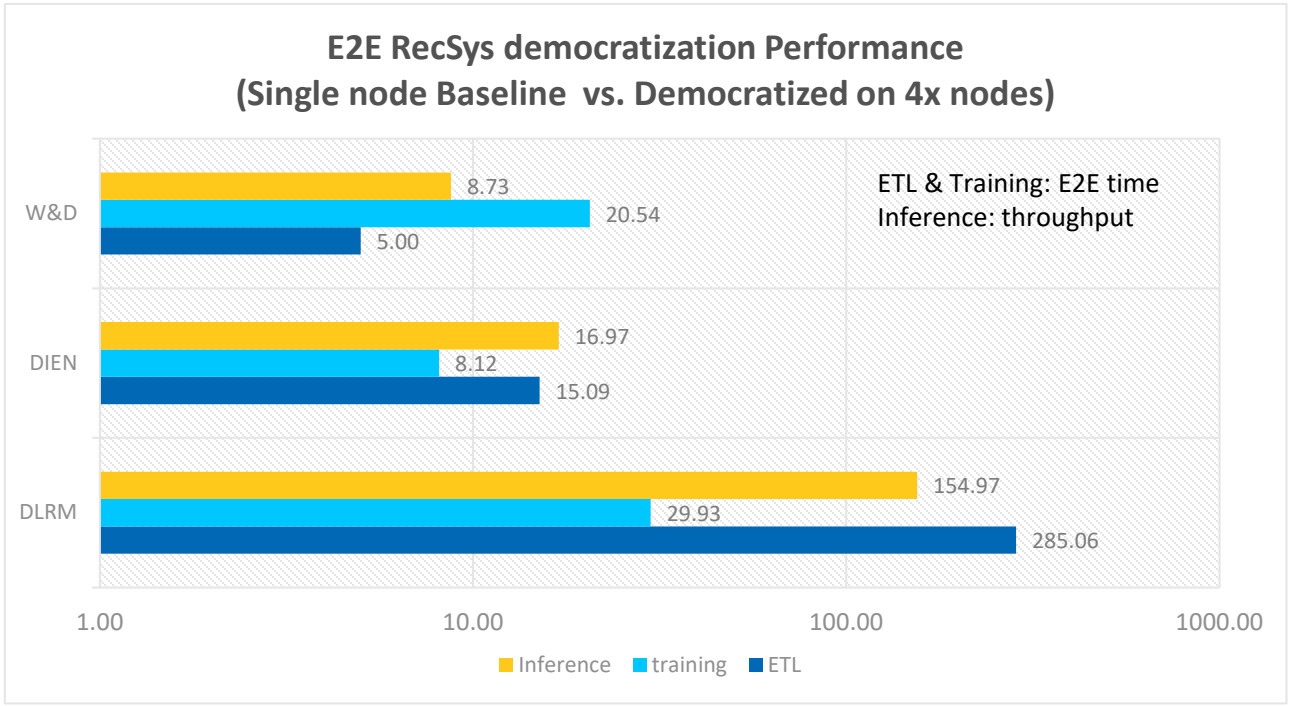
- Using a zero-cost proxy performance predictor that positively related with training accuracy to score candidate network rather than training



*zero-cost proxy: https://arxiv.org/abs/2101.08134

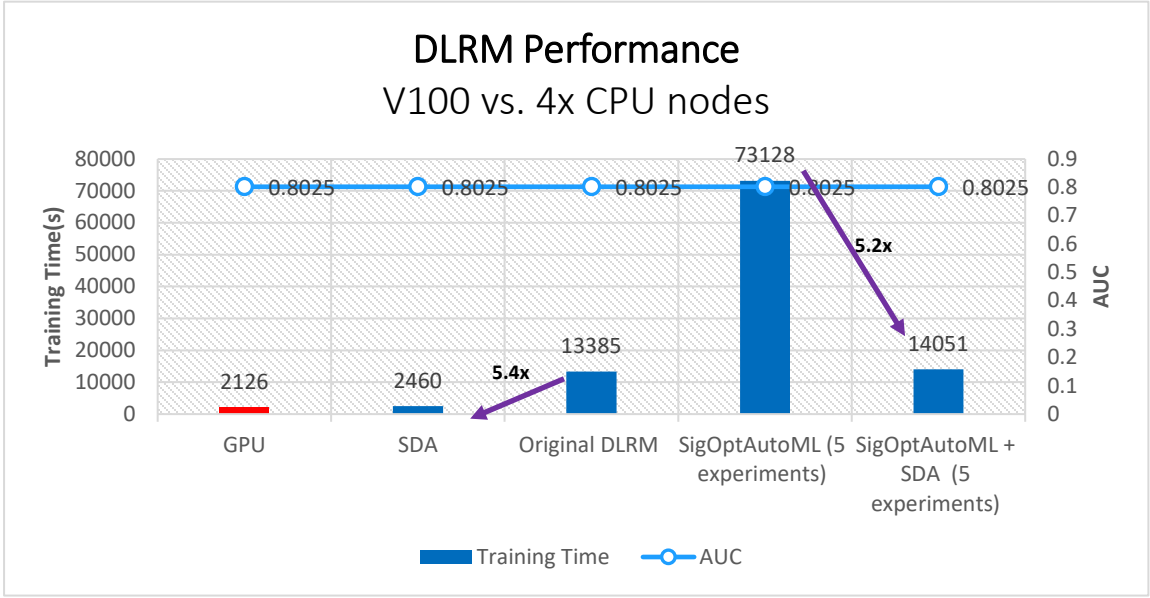
Performance

Performance: RecSys E2E AI Democratization



- E2E democratization delivered **29.9x**, **8.12x** and **20.5x** training time speedup, **154x, 17x** and **8.7x** inference throughput speedup for DLRM, WnD and DIEN respectively
- Key optimizations/contributors:
  - Scale-out data processing with Spark to replace single thread python data processing (DLRM, DIEN) and scalable distributed training with multiple Xeon nodes
  - Intel optimized training frameworks - IPEX & Intel optimized TF & BF16
  - **Lighter models** – reduced MLP & NN layers, reduced communication overhead on Ethernet
  - **Optimizer tuning** (DLRM) to converge faster with larger batch size
  - Feature Engineering (embedding table, encoding) optimizations

Performance: SDA with SigOpt



- Enhanced SigOpt hyper-parameter tuning with SDA
  - Transform manual optimization to AutoML
  - Reduced total time to get the desired model, 5.2x speedup of 5x SigOpt experiments

Summary

Summary

- End-to-End AI is a complex process involve multiple stages and posed lots of challenges for today's data platform infrastructure
- A unified data platform specifically optimized for End-to-End AI helps to improve performance and reduce cost
- AI democratization makes AI **accessible** & **affordable** to everyone
- Based on opensource frameworks/solutions, Intel® End-to-End AI optimization Kit complements the Data platform for E2E AI democratization
  - It delivers E2E AI optimization kits from data processing, feature engineering, data connector, model creation & optimization, hyper-parameter optimization
  - Preliminary results with Recommendation systems demonstrated very promising results
  - More workloads – Compute Vision/Neural Language processing/Automatic Speech Recognition is coming
- Welcome to download Intel® End-to-End AI optimization kit (https://github.com/intel/e2eaiok) and share with us your feedbacks!

Please take a moment to rate this session.

Your feedback is important to us.

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.

Your costs and results may vary.

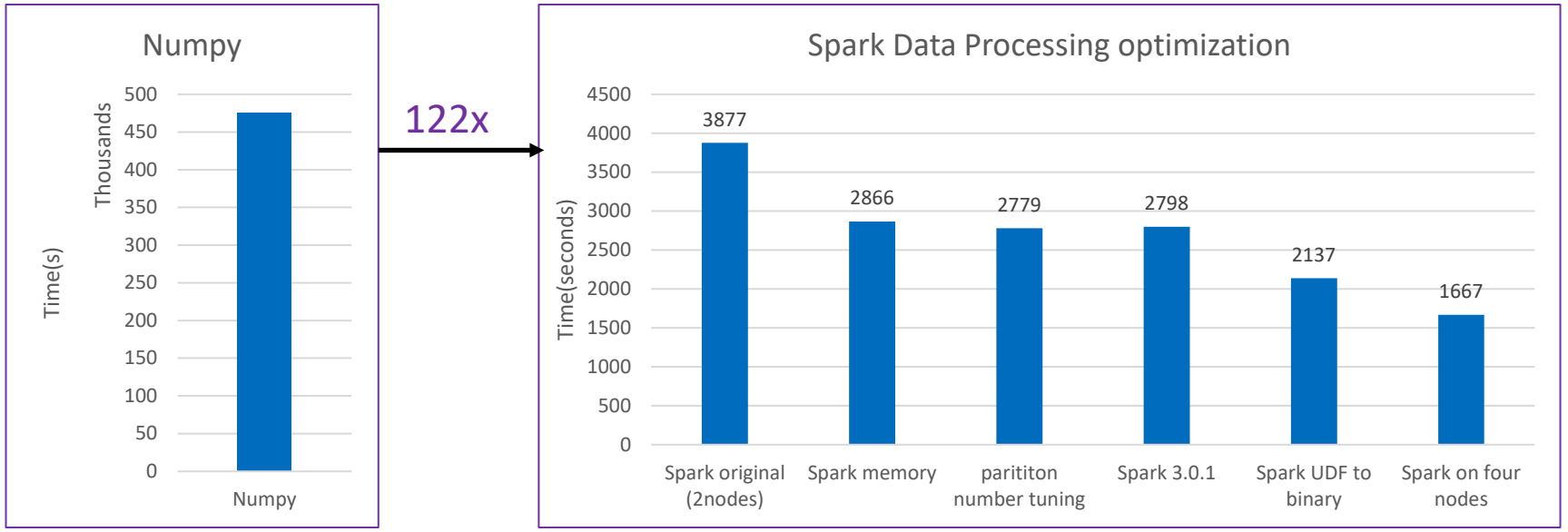
Intel technologies may require enabled hardware, software or service activation.

Configuration Details

BaseLine: Test by Intel as of 2021/10. 1-node, 2x Intel® Xeon® Gold 6240 Processor, 18 cores HT On Turbo ON Total Memory 384 GB (12 slots/ 32GB/ 2666 MHz), BIOS: SE5C620.86B.0X.02.0094.102720191711 (ucode:0x500002C), Fedora 29, 5.3.11-100.fc29.x86_64, pytorch, tensorflow, spark 3, DLRM, WnD, DIEN

Democratization Config: Test by Intel as of 2021/10. 4-nodes, 2x Intel® Xeon® Gold 6240 Processor, 18 cores HT On Turbo ON Total Memory 384 GB (12 slots/ 32GB/ 2666 MHz), BIOS: SE5C620.86B.0X.02.0094.102720191711 (ucode:0x500002C), Fedora 29, 5.3.11-100.fc29.x86_64, pytorch, IPEX, Intel optimized tensorflow, horovod, spark 3, modified DLRM, WnD, DIEN workloads

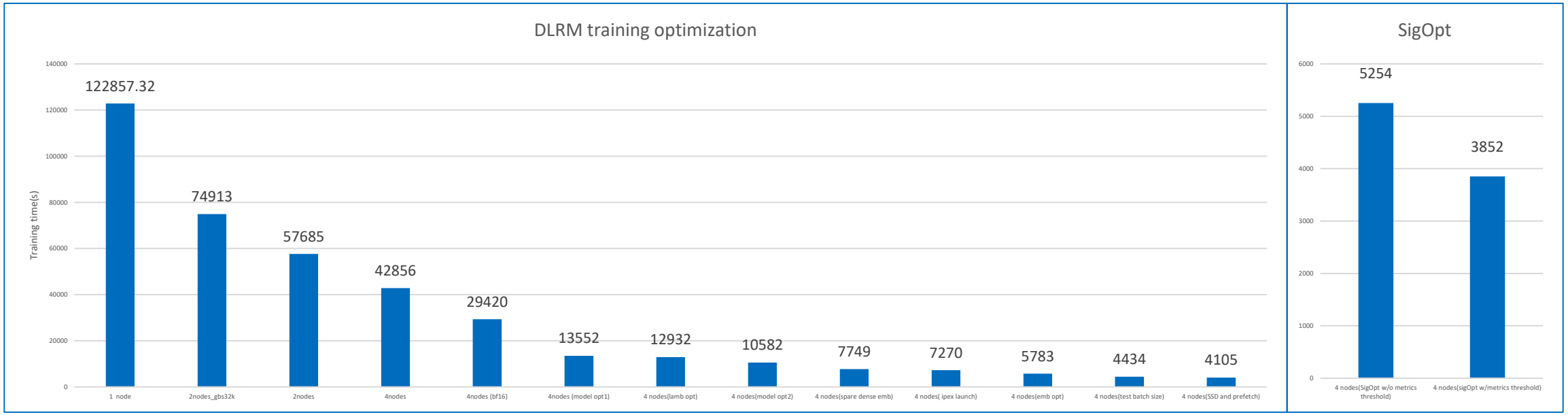
DLRM Data processing optimizations with RecDP



- Significant speedup w/ Spark parallel data processing
  - Excellent scalability of Spark DP: 2.32x from 2 nodes to 4 nodes
- Ease of use with Spark DP

* with frequent limit – OOM if frequent limit not added
* data connector not included

DLRM Training Optimization with SDA



- Key Optimizations
  - **Intel Optimized training framework + Lamb optimizer:** 1.29x speedup (74913s to 57865s)
  - **BF16:** 1.45x speedup (42856s to 29420s)
  - **Lighter Model-part1**: model optimization(decrease both dense embedding and sparse embedding output dimension from 128 to 64, reduced bot_mlp layer size from 13-512-256-128 to 13-256-128-64, top_mlp layer size from 1024-1024-512-256-1 to 512-512-256-1) delivers 2.17x speedup(from 29420s to 13552s)
  - **Optimizer**: lamb optimizer delivers 1.04x speedup(from 13552s to 12932s),
  - **Lighter Model-part2**: model optimization(reduced bot_mlp layer size from 13-256-128-64 to 13-128-64, top_mlp layer size from 512-512-256-1 to 256-128-1) delivers 1.22x speedup(from 12932s to 10582s)
  - **Embedding**: reducing sparse embedding table number from 16 to 8 delivers 1.36x speedup(from 10582s to 7749s)
  - **Framework optimization:** using latest IPEX launch scripts (auto bind CCL to specific core) and add KMP setting delivers 1.06x speedup (from 7749s to 7270s)
  - **AlltoAll optimization:** reduce the output dimension size from 64 to 16 and then repeat 4 times to 64 dimension after alltoall, which delivers 1.25x speedup(from 7270s to 5783s)
  - **Learning rate and test optimization:** change lamb optimizer learning rate from 16 to 30;remove gradient computing in the test part; increase test batch size from 16K to 128K. These optimizations delivers 1.30x speedup(from 5783s to 4434s)
  - **Other optimization:** (1) Using NVMe to replace HDD as storage;(2) Leveraging prefetch_generator PyTorch dataloader optimization package to enable batch data generator working in background thread parallelly with training. These optimizations delivers 1.08x speedup(from 4434s to 4105s)
  - **SigOpt optimization**: In limit experiments trials, SigOpt with metrics threshold delivers 1.06x speedup(from 4105s to 3852s) than previous manual optimized results

Model Optimization with SDA

- Reduce both dense embedding and sparse embedding output dimension from 128 to 64
- Reduce bottom mlp layer size from original size of “13-512-256-128” to current size of “13-128-64”
- Reduce top mlp layer size from original size of “1024-1024-512-256-1” to current size of “256-128-1”

```
DLRM_Net(
  (emb_dense): ModuleList(
    (0): EmbeddingBag(3, 128, mode=sum)
    (1): EmbeddingBag(1543, 128, mode=sum)
    (2): EmbeddingBag(63, 128, mode=sum)
    (3): EmbeddingBag(10, 128, mode=sum)
    (4): EmbeddingBag(155, 128, mode=sum)
    (5): EmbeddingBag(4, 128, mode=sum)
    (6): EmbeddingBag(976, 128, mode=sum)
    (7): EmbeddingBag(14, 128, mode=sum)
    (8): EmbeddingBag(108, 128, mode=sum)
    (9): EmbeddingBag(36, 128, mode=sum)
  )
  (emb_sparse): ModuleList(
    (0): EmbeddingBag(39884406, 128, mode=sum)
    (1): EmbeddingBag(39043, 128, mode=sum)
  )
  (bot_l): Sequential(
    (0): IpexMLPLinear(C=13, K=512, bias=True)
    (1): IpexMLPLinear(C=512, K=256, bias=True)
    (2): IpexMLPLinear(C=256, K=128, bias=True)
  )
  (top_l): Sequential(
    (0): IpexMLPLinear(C=479, K=1024, bias=True)
    (1): IpexMLPLinear(C=1024, K=1024, bias=True)
    (2): IpexMLPLinear(C=1024, K=512, bias=True)
    (3): IpexMLPLinear(C=512, K=256, bias=True)
    (4): IpexMLPLinear(C=256, K=1, bias=True)
    (5): Cast(to(torch.float32))
    (6): Sigmoid()
  )
) dpcpp True
```

```
DLRM_Net(
  (emb_dense): ModuleList(
    (0): EmbeddingBag(3, 64, mode=sum)
    (1): EmbeddingBag(1543, 64, mode=sum)
    (2): EmbeddingBag(63, 64, mode=sum)
    (3): EmbeddingBag(10, 64, mode=sum)
    (4): EmbeddingBag(155, 64, mode=sum)
    (5): EmbeddingBag(4, 64, mode=sum)
    (6): EmbeddingBag(976, 64, mode=sum)
    (7): EmbeddingBag(14, 64, mode=sum)
    (8): EmbeddingBag(108, 64, mode=sum)
    (9): EmbeddingBag(36, 64, mode=sum)
  )
  (emb_sparse): ModuleList(
    (0): EmbeddingBag(39884406, 64, mode=sum)
    (1): EmbeddingBag(39043, 64, mode=sum)
  )
  (bot_l): Sequential(
    (0): IpexMLPLinear(C=13, K=128, bias=True)
    (1): IpexMLPLinear(C=128, K=64, bias=True)
  )
  (top_l): Sequential(
    (0): IpexMLPLinear(C=415, K=256, bias=True)
    (1): IpexMLPLinear(C=256, K=128, bias=True)
    (2): IpexMLPLinear(C=128, K=1, bias=True)
    (3): Cast(to(torch.float32))
    (4): Sigmoid()
  )
) dpcpp True
```

--arch-mlp-top="1024-1024-512-256-1"

```
top_l
└─ module
   ├─ 0
   │  ├─ unsigned short   | 3.423ms   | 3.423ms   | 128
   │  └─ addmm            | 2.015s    | 2.015s    | 128
   ├─ 1
   │  └─ relu             | 500.718ms | 500.718ms | 128
   ├─ 2
   │  ├─ unsigned short   | 4.442ms   | 4.442ms   | 128
   │  └─ addmm            | 3.327s    | 3.327s    | 128
   ├─ 3
   │  └─ relu             | 488.669ms | 488.669ms | 128
   ├─ 4
   │  ├─ unsigned short   | 4.180ms   | 4.180ms   | 128
   │  └─ addmm            | 1.609s    | 1.609s    | 128
   ├─ 5
   │  └─ relu             | 283.411ms | 283.411ms | 128
   ├─ 6
   │  ├─ unsigned short   | 4.157ms   | 4.157ms   | 128
   │  └─ addmm            | 441.480ms | 441.480ms | 128
   ├─ 7
   │  └─ relu             | 83.995ms  | 83.995ms  | 128
   ├─ 8
   │  ├─ unsigned short   | 3.316ms   | 3.316ms   | 128
   │  └─ addmm            | 32.001ms  | 32.001ms  | 128
   └─ 9
      └─ sigmoid          | 8.356ms   | 8.356ms   | 128
```

--arch-mlp-top=" 512-512-256-1 "

```
top_l
└─ module
   ├─ 0
   │  ├─ unsigned short   | 3.343ms   | 3.343ms   | 128
   │  └─ addmm            | 753.999ms | 753.999ms | 128
   ├─ 1
   │  └─ relu             | 219.717ms | 219.717ms | 128
   ├─ 2
   │  ├─ unsigned short   | 4.080ms   | 4.080ms   | 128
   │  └─ addmm            | 957.987ms | 957.987ms | 128
   ├─ 3
   │  └─ relu             | 267.011ms | 267.011ms | 128
   ├─ 4
   │  ├─ unsigned short   | 4.375ms   | 4.375ms   | 128
   │  └─ addmm            | 440.613ms | 440.613ms | 128
   ├─ 5
   │  └─ relu             | 84.693ms  | 84.693ms  | 128
   ├─ 6
   │  ├─ unsigned short   | 3.320ms   | 3.320ms   | 128
   │  └─ addmm            | 31.724ms  | 31.724ms  | 128
   └─ 7
      └─ sigmoid          | 9.431ms   | 9.431ms   | 128
```

Original Model

```
Finished training it 464/16004 of epoch 0, 1736.66 ms/it, loss 0.127475, accuracy 96.725 %
Finished training it 464/16004 of epoch 0, 1726.97 ms/it, loss 0.127765, accuracy 96.717 %
Finished training it 464/16004 of epoch 0, 1721.85 ms/it, loss 0.127917, accuracy 96.717 %
Finished training it 464/16004 of epoch 0, 1734.39 ms/it, loss 0.128917, accuracy 96.693 %
Finished training it 464/16004 of epoch 0, 1733.95 ms/it, loss 0.127919, accuracy 96.714 %
Finished training it 464/16004 of epoch 0, 1722.25 ms/it, loss 0.128480, accuracy 96.698 %
Finished training it 464/16004 of epoch 0, 1723.48 ms/it, loss 0.128170, accuracy 96.703 %
Finished training it 464/16004 of epoch 0, 1733.57 ms/it, loss 0.128138, accuracy 96.696 %
Finished training it 480/16004 of epoch 0, 1741.04 ms/it, loss 0.129579, accuracy 96.679 %
Finished training it 480/16004 of epoch 0, 1746.10 ms/it, loss 0.129109, accuracy 96.688 %
Finished training it 480/16004 of epoch 0, 1740.60 ms/it, loss 0.127173, accuracy 96.755 %
Finished training it 480/16004 of epoch 0, 1751.16 ms/it, loss 0.127937, accuracy 96.712 %
Finished training it 480/16004 of epoch 0, 1749.55 ms/it, loss 0.127299, accuracy 96.733 %
Finished training it 480/16004 of epoch 0, 1744.15 ms/it, loss 0.127302, accuracy 96.745 %
Finished training it 480/16004 of epoch 0, 1741.05 ms/it, loss 0.127739, accuracy 96.738 %
Finished training it 480/16004 of epoch 0, 1753.62 ms/it, loss 0.128161, accuracy 96.703 %
Finished training it 496/16004 of epoch 0, 1731.13 ms/it, loss 0.128780, accuracy 96.682 %
Finished training it 496/16004 of epoch 0, 1732.79 ms/it, loss 0.128010, accuracy 96.716 %
Finished training it 496/16004 of epoch 0, 1729.40 ms/it, loss 0.128532, accuracy 96.684 %
Finished training it 496/16004 of epoch 0, 1732.98 ms/it, loss 0.128909, accuracy 96.684 %
Finished training it 496/16004 of epoch 0, 1733.40 ms/it, loss 0.128134, accuracy 96.694 %
Finished training it 496/16004 of epoch 0, 1733.95 ms/it, loss 0.128729, accuracy 96.690 %
Finished training it 496/16004 of epoch 0, 1733.27 ms/it, loss 0.128602, accuracy 96.701 %
Finished training it 496/16004 of epoch 0, 1731.84 ms/it, loss 0.128342, accuracy 96.695 %
```

Optimized Model

```
Finished training it 464/16004 of epoch 0, 613.90 ms/it, loss 0.128640, accuracy 96.676 %
Finished training it 464/16004 of epoch 0, 615.41 ms/it, loss 0.126504, accuracy 96.756 %
Finished training it 464/16004 of epoch 0, 621.91 ms/it, loss 0.127110, accuracy 96.724 %
Finished training it 464/16004 of epoch 0, 624.91 ms/it, loss 0.128911, accuracy 96.691 %
Finished training it 464/16004 of epoch 0, 623.17 ms/it, loss 0.128898, accuracy 96.679 %
Finished training it 464/16004 of epoch 0, 613.39 ms/it, loss 0.128860, accuracy 96.669 %
Finished training it 464/16004 of epoch 0, 619.47 ms/it, loss 0.128149, accuracy 96.705 %
Finished training it 464/16004 of epoch 0, 623.94 ms/it, loss 0.127570, accuracy 96.704 %
Finished training it 480/16004 of epoch 0, 617.31 ms/it, loss 0.127326, accuracy 96.713 %
Finished training it 480/16004 of epoch 0, 612.40 ms/it, loss 0.128756, accuracy 96.670 %
Finished training it 480/16004 of epoch 0, 613.76 ms/it, loss 0.127805, accuracy 96.708 %
Finished training it 480/16004 of epoch 0, 615.16 ms/it, loss 0.127408, accuracy 96.720 %
Finished training it 480/16004 of epoch 0, 614.36 ms/it, loss 0.128632, accuracy 96.687 %
Finished training it 480/16004 of epoch 0, 613.90 ms/it, loss 0.129496, accuracy 96.648 %
Finished training it 480/16004 of epoch 0, 613.16 ms/it, loss 0.128469, accuracy 96.685 %
Finished training it 480/16004 of epoch 0, 613.93 ms/it, loss 0.128865, accuracy 96.685 %
Finished training it 496/16004 of epoch 0, 663.25 ms/it, loss 0.127287, accuracy 96.718 %
Finished training it 496/16004 of epoch 0, 664.08 ms/it, loss 0.127821, accuracy 96.692 %
Finished training it 496/16004 of epoch 0, 662.62 ms/it, loss 0.128212, accuracy 96.676 %
Finished training it 496/16004 of epoch 0, 660.25 ms/it, loss 0.127742, accuracy 96.699 %
Finished training it 496/16004 of epoch 0, 666.27 ms/it, loss 0.127530, accuracy 96.702 %
Finished training it 496/16004 of epoch 0, 662.56 ms/it, loss 0.128617, accuracy 96.675 %
Finished training it 496/16004 of epoch 0, 666.47 ms/it, loss 0.127428, accuracy 96.708 %
```

