Hummingbird: A Tensor Compiler for Unified Machine Learning Prediction Serving

Supun Nakandala[u], Karla Saur[m], Gyeong-In Yu[s], Konstantinos Karanasos[m], Carlo Curino[m], Markus Weimer[m], Matteo Interlandi[m]

Machine Learning Prediction Serving

ML prediction serving has emerged as an important systems problem.

High throughput, low latency, engineering concerns (e.g., maintainability)

Machine Learning Prediction Serving

ML prediction serving has emerged as an important systems problem.

High throughput, low latency, engineering concerns (e.g., maintainability)

Responsible for 45%-65% of the total cost of ownership of ML solutions.

source: “The Total Cost of Ownership of Amazon SageMaker”

Machine Learning Prediction Serving

ML prediction serving has emerged as an important systems problem.

High throughput, low latency, engineering concerns (e.g., maintainability)

Responsible for 45%-65% of the total cost of ownership of ML solutions.

source: “The Total Cost of Ownership of Amazon SageMaker”

Specialized systems have been developed.

tvm TensorRT ONNX RUNTIME

Machine Learning Prediction Serving

ML prediction serving has emerged as an important systems problem.

High throughput, low latency, engineering concerns (e.g., maintainability)

Responsible for 45%-65% of the total cost of ownership of ML solutions.

source: “The Total Cost of Ownership of Amazon SageMaker”

Specialized systems have been developed.

Focus: Deep Learning (DL)

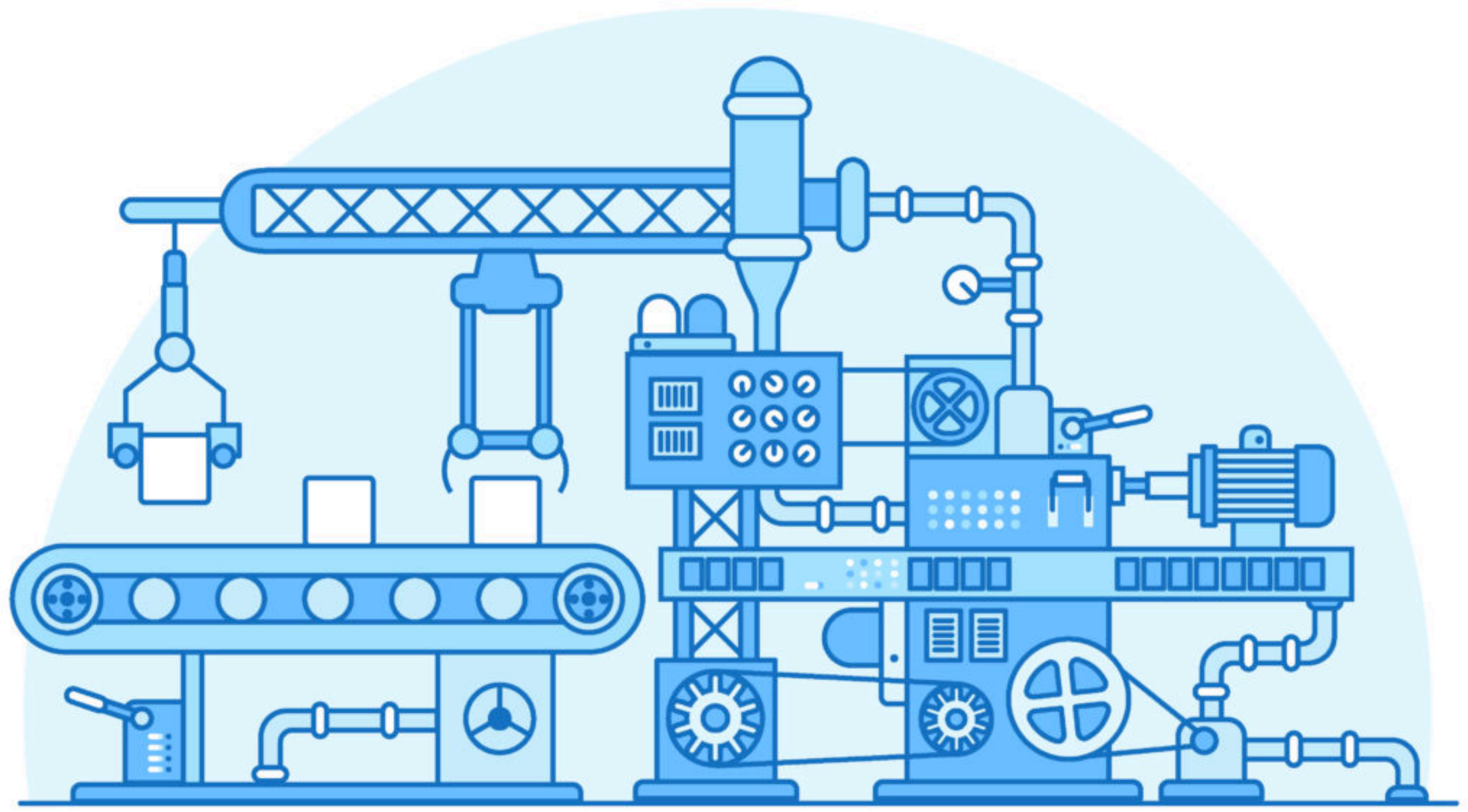
tvm TensorRT ONNX RUNTIME

Traditional Machine Learning in the Enterprises

Traditional Machine Learning in the Enterprises

Powered By: Traditional Machine Learning

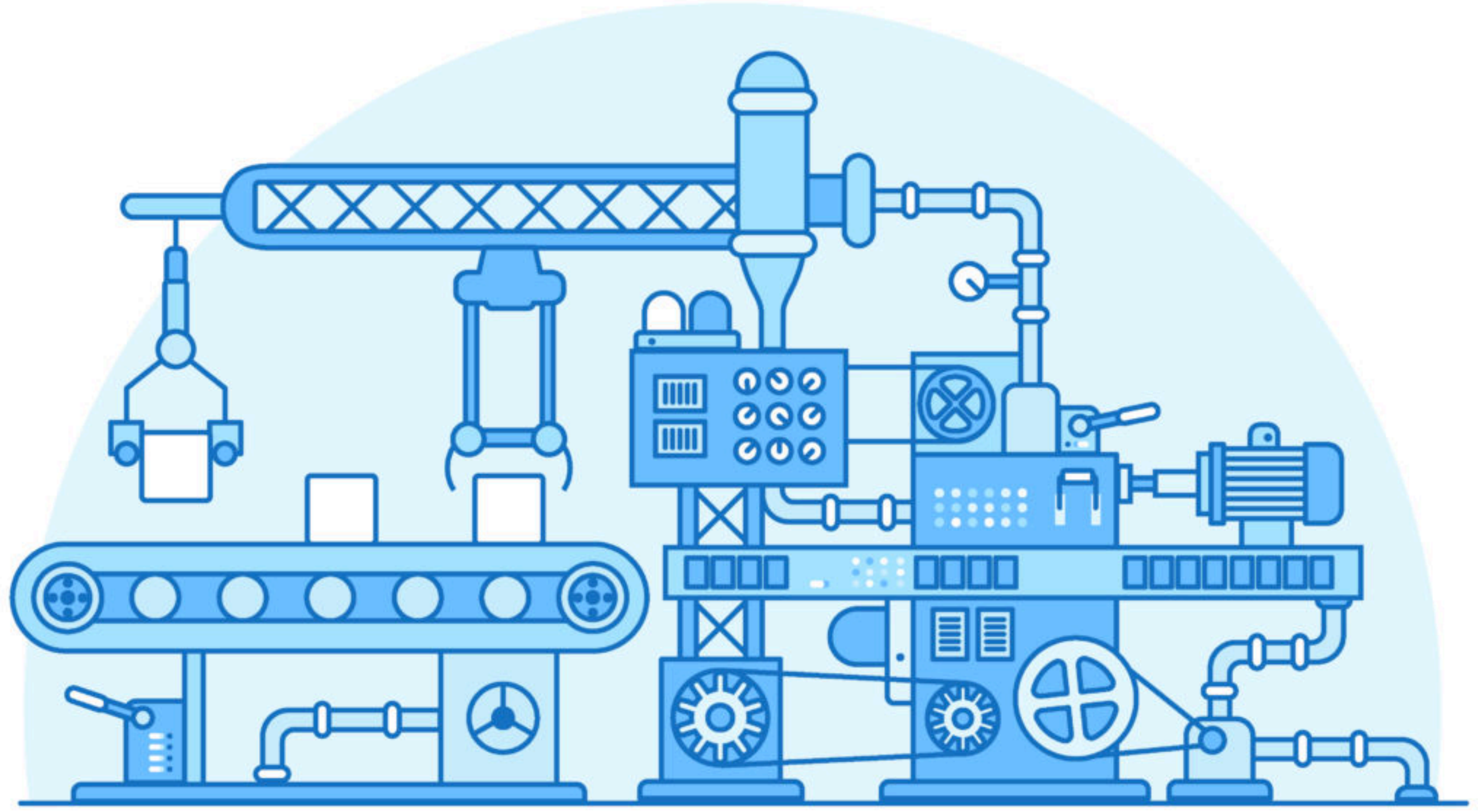
Traditional Machine Learning in the Enterprises



Predictive Maintenance

Powered By: Traditional Machine Learning

Traditional Machine Learning in the Enterprises

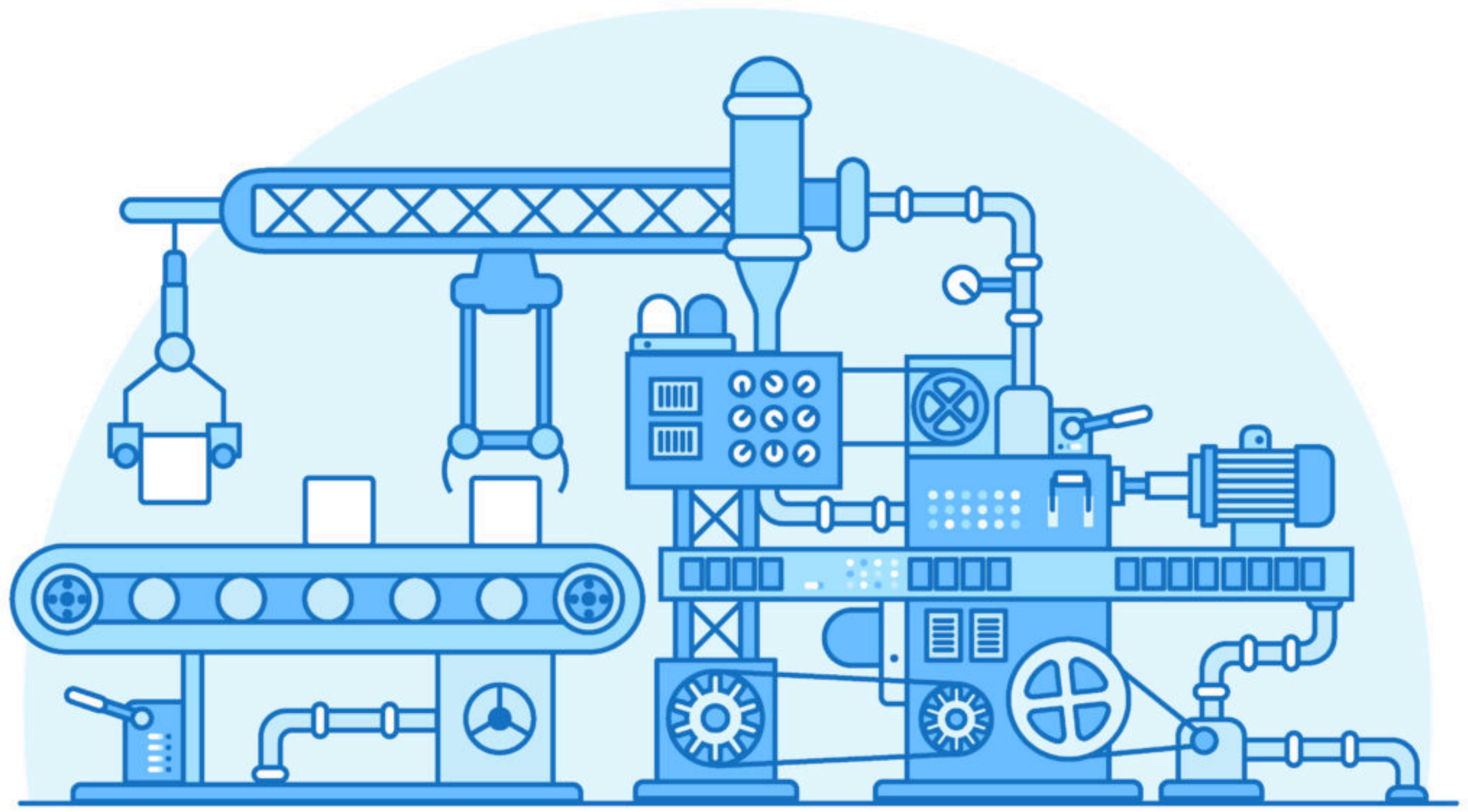


Predictive Maintenance

Supply-chain Optimizations

Powered By: Traditional Machine Learning

Traditional Machine Learning in the Enterprises



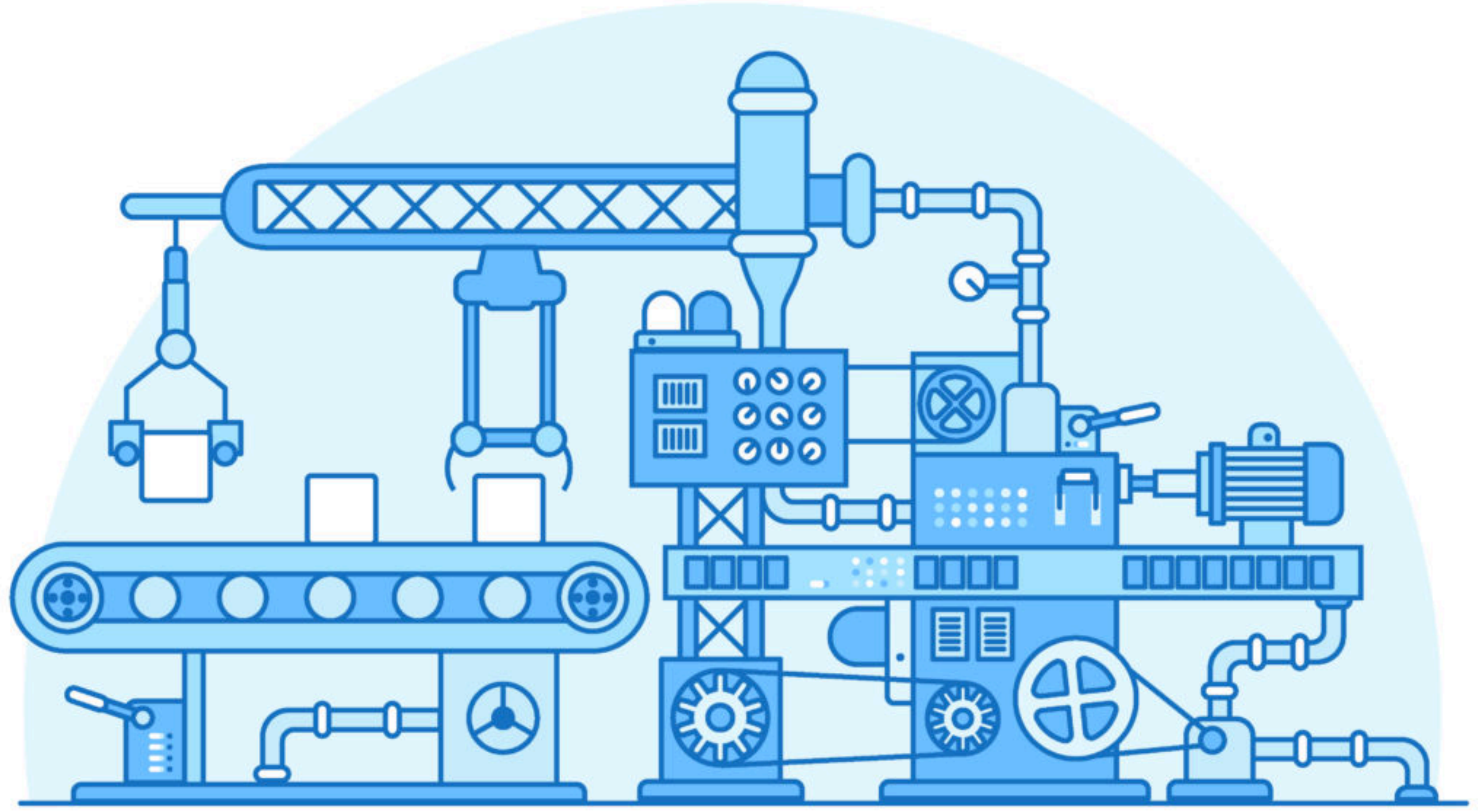
Predictive Maintenance

Supply-chain Optimizations

Customer Churn Prediction

Powered By: Traditional Machine Learning

Traditional Machine Learning in the Enterprises



Predictive Maintenance

Supply-chain Optimizations

Customer Churn Prediction

Powered By: Traditional Machine Learning

50%-95% of all ML applications in an organization are based on Traditional ML

source: “The Total Cost of Ownership of Amazon SageMaker”

Problem: Lack of Optimized Systems for Traditional ML Serving

Systems for training traditional ML models are not optimized for serving.

Problem: Lack of Optimized Systems for Traditional ML Serving

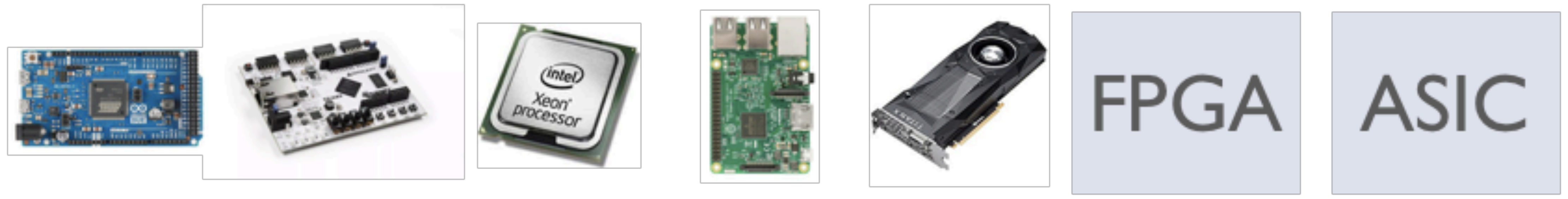
Systems for training traditional ML models are not optimized for serving.

Traditional ML models are expressed using **imperative code** in an ad-hoc fashion, not using a **shared logical abstraction**.

Problem: Lack of Optimized Systems for Traditional ML Serving

Systems for training traditional ML models are not optimized for serving.

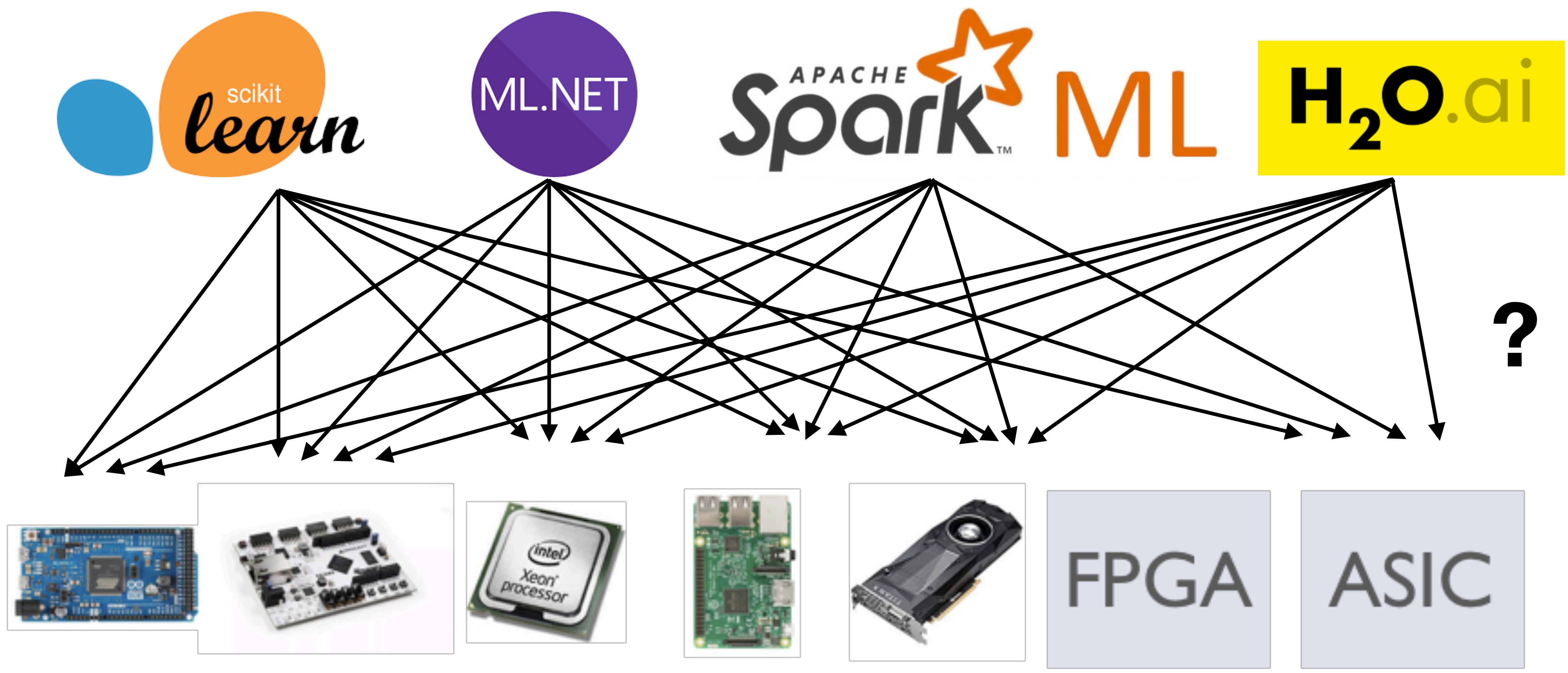
Traditional ML models are expressed using **imperative code** in an ad-hoc fashion, not using a **shared logical abstraction.**



Problem: Lack of Optimized Systems for Traditional ML Serving

Systems for training traditional ML models are not optimized for serving.

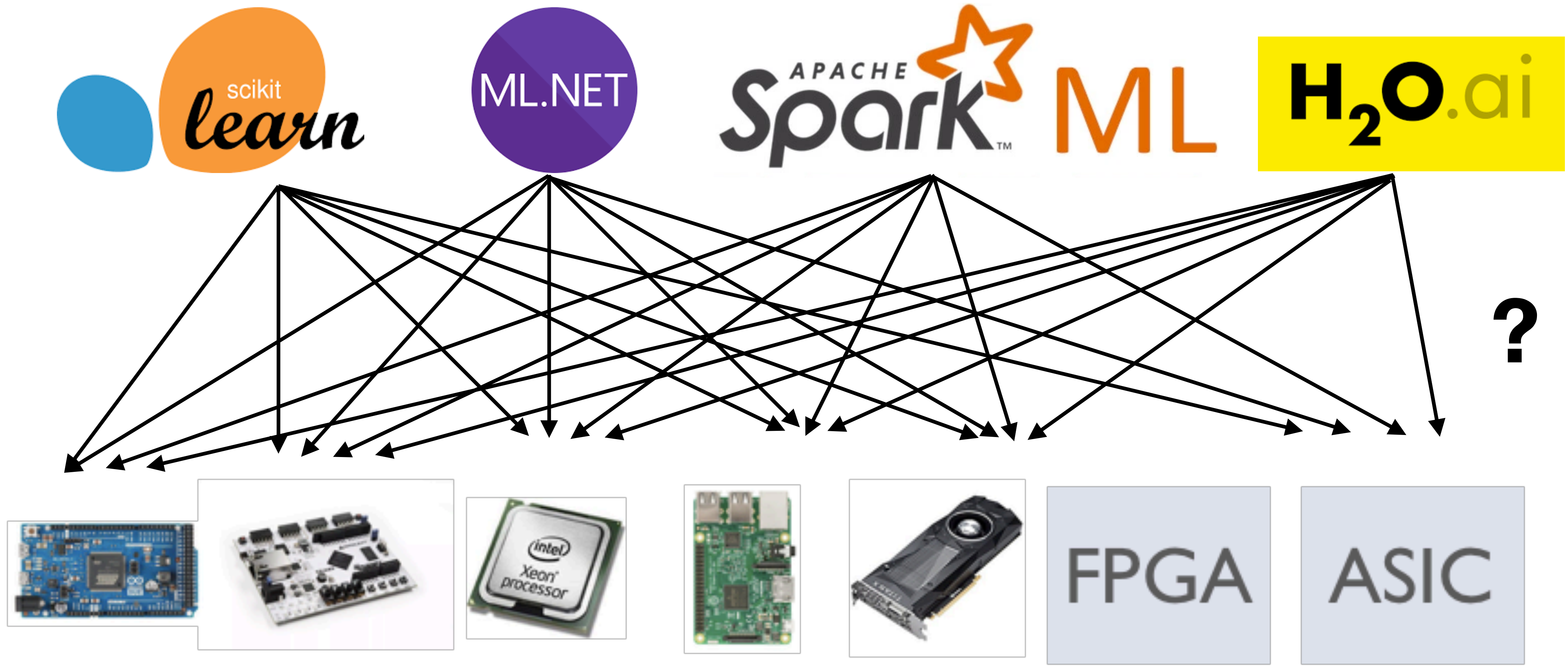
Traditional ML models are expressed using **imperative code** in an ad-hoc fashion, not using a **shared logical abstraction.**



Problem: Lack of Optimized Systems for Traditional ML Serving

Systems for training traditional ML models are not optimized for serving.

Traditional ML models are expressed using **imperative code** in an ad-hoc fashion, not using a **shared logical abstraction.**



Highly complex solutions, amplified engineering costs, and reduced operational performance.

Proposed Approach: Reuse DL Prediction Serving Systems for Traditional ML Serving

Hummingbird, a system that can execute traditional ML models on DL prediction serving systems.

Proposed Approach: Reuse DL Prediction Serving Systems for Traditional ML Serving

Hummingbird, a system that can execute traditional ML models on DL prediction serving systems.

Benefits: Up to **1200x speedups** for predictive pipelines against state-of-the-art frameworks.

Proposed Approach: Reuse DL Prediction Serving Systems for Traditional ML Serving

Hummingbird, a system that can execute traditional ML models on DL prediction serving systems.

Benefits:

Up to **1200x speedups** for predictive pipelines against state-of-the-art frameworks.

Seamless hardware acceleration w/ up to **3x speedups** compared hand-crafted GPU kernels.

Proposed Approach: Reuse DL Prediction Serving Systems for Traditional ML Serving

Hummingbird, a system that can execute traditional ML models on DL prediction serving systems.

Benefits:

Up to **1200x speedups** for predictive pipelines against state-of-the-art frameworks.

Seamless hardware acceleration w/ up to **3x speedups** compared hand-crafted GPU kernels.

Significantly reduced engineering efforts and software complexity. Increased Portability.

Outline

1. Background

   Traditional Machine Learning

   Deep Learning (DL) and Systems for DL Prediction Serving

2. Our System: Hummingbird

3. Experimental Evaluation

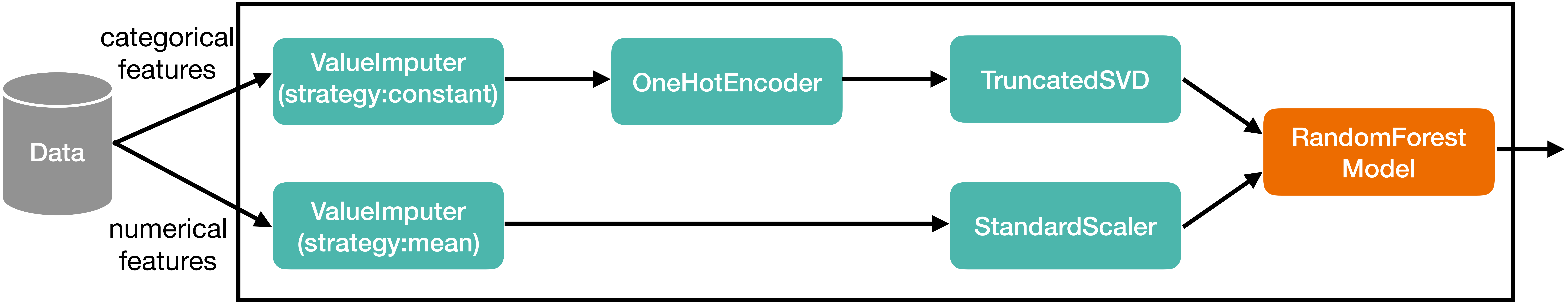
Traditional Machine Learning

Traditional Machine Learning

Predictive pipelines (DAGs) composed of ***featurization*** and ***model*** operators.

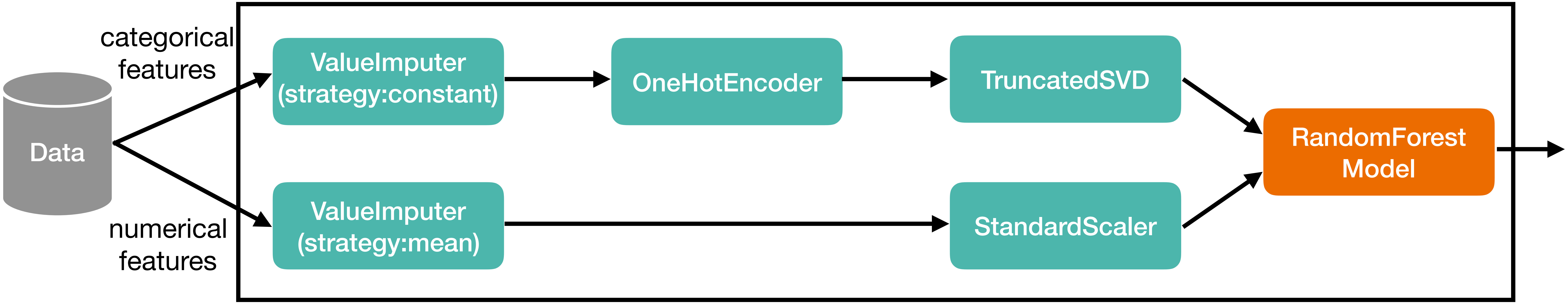
Traditional Machine Learning

Predictive pipelines (DAGs) composed of ***featurization*** and ***model*** operators.



Traditional Machine Learning

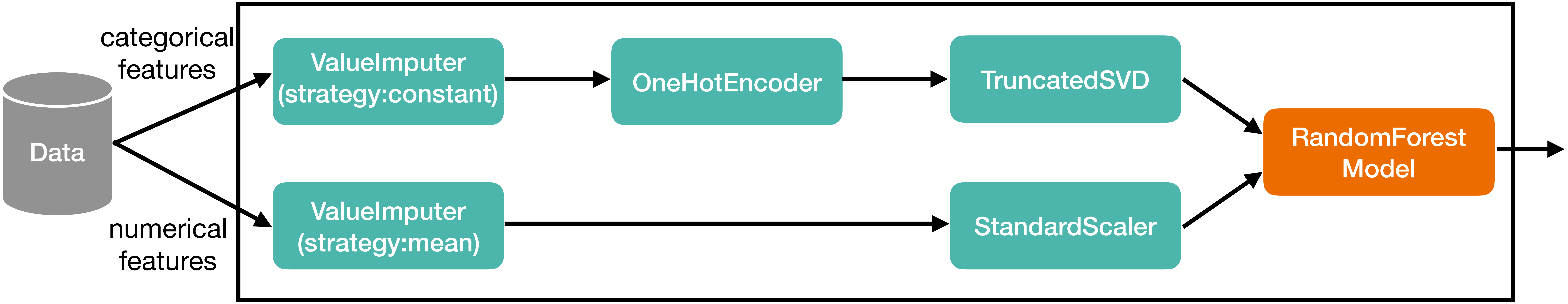
Predictive pipelines (DAGs) composed of ***featurization*** and ***model*** operators.



Operators are expressed using **imperative code**.

Traditional Machine Learning

Predictive pipelines (DAGs) composed of ***featurization*** and ***model*** operators.



Operators are expressed using **imperative code.**

Can contain 10s of operators selected from 100s of potential featurization and model operators.

Outline

1. Background

    Traditional Machine Learning

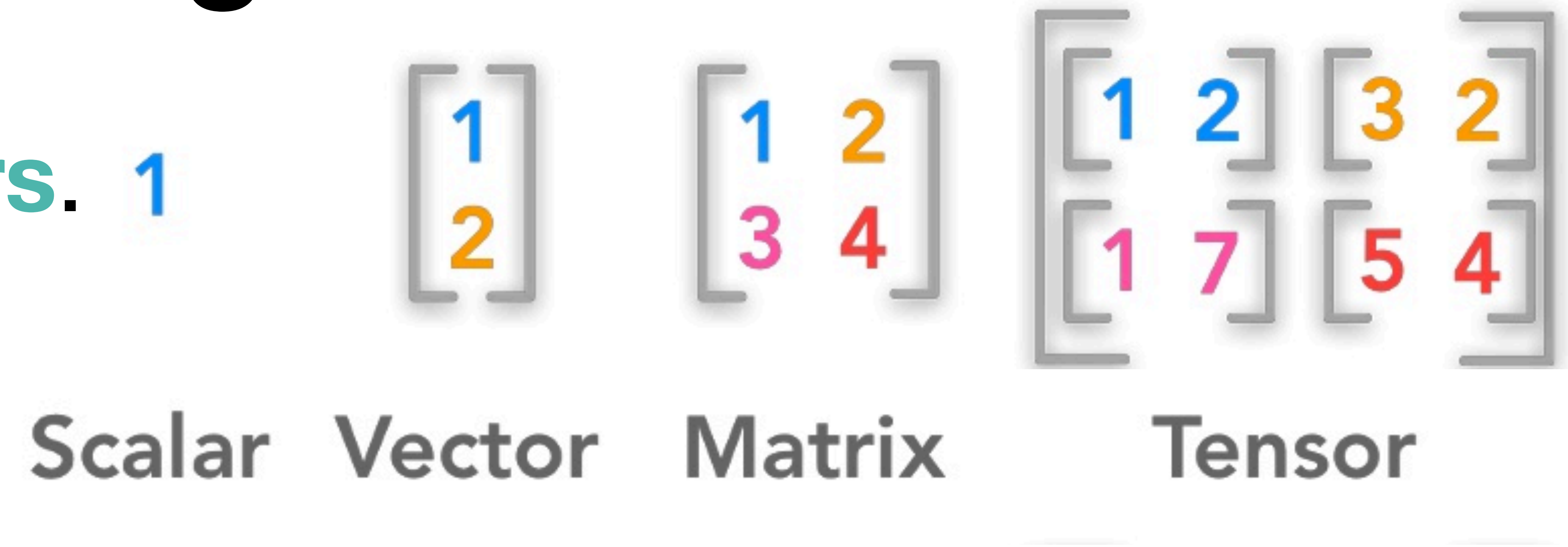
    Deep Learning (DL) and Systems for DL Prediction Serving

2. Our System: Hummingbird

3. Experimental Evaluation

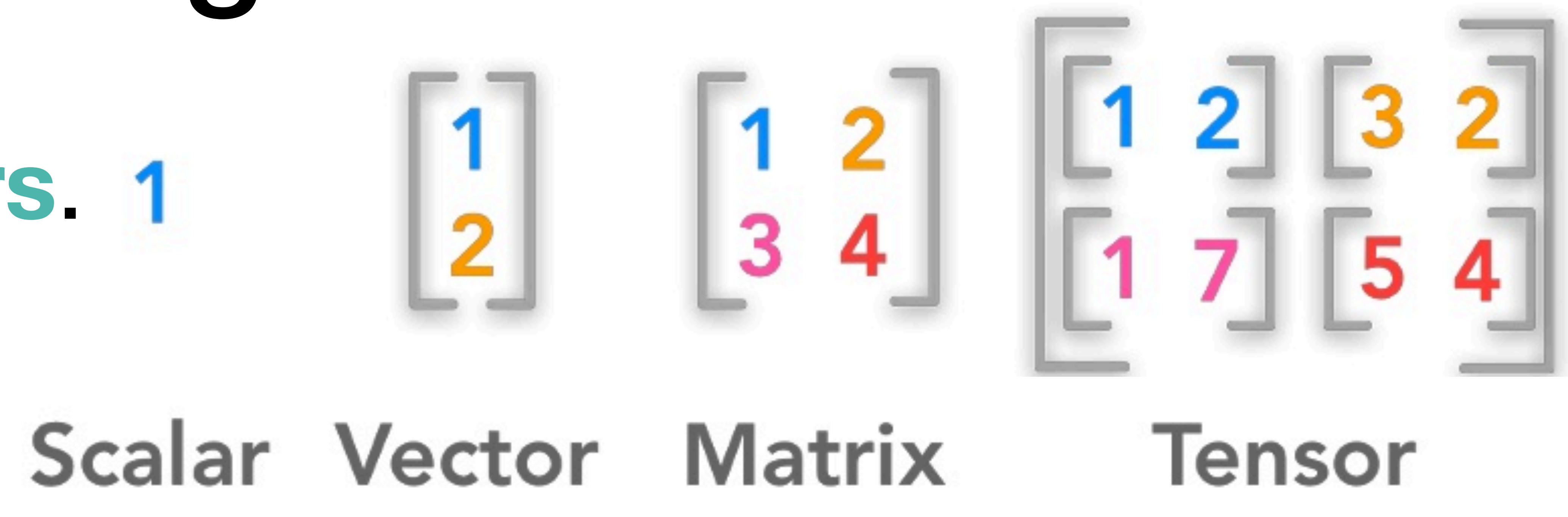
Deep Learning

Primarily relies on the abstraction of **tensors**.

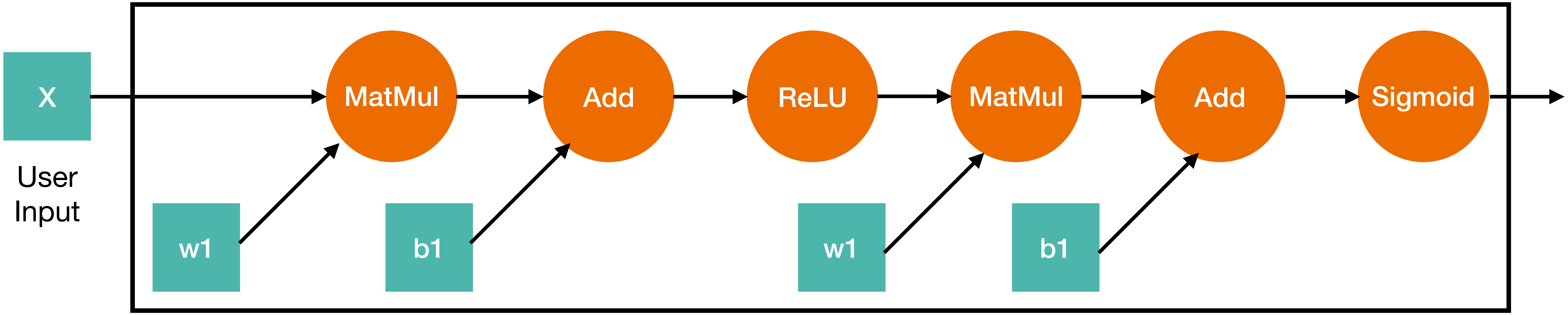


Deep Learning

Primarily relies on the abstraction of **tensors**.

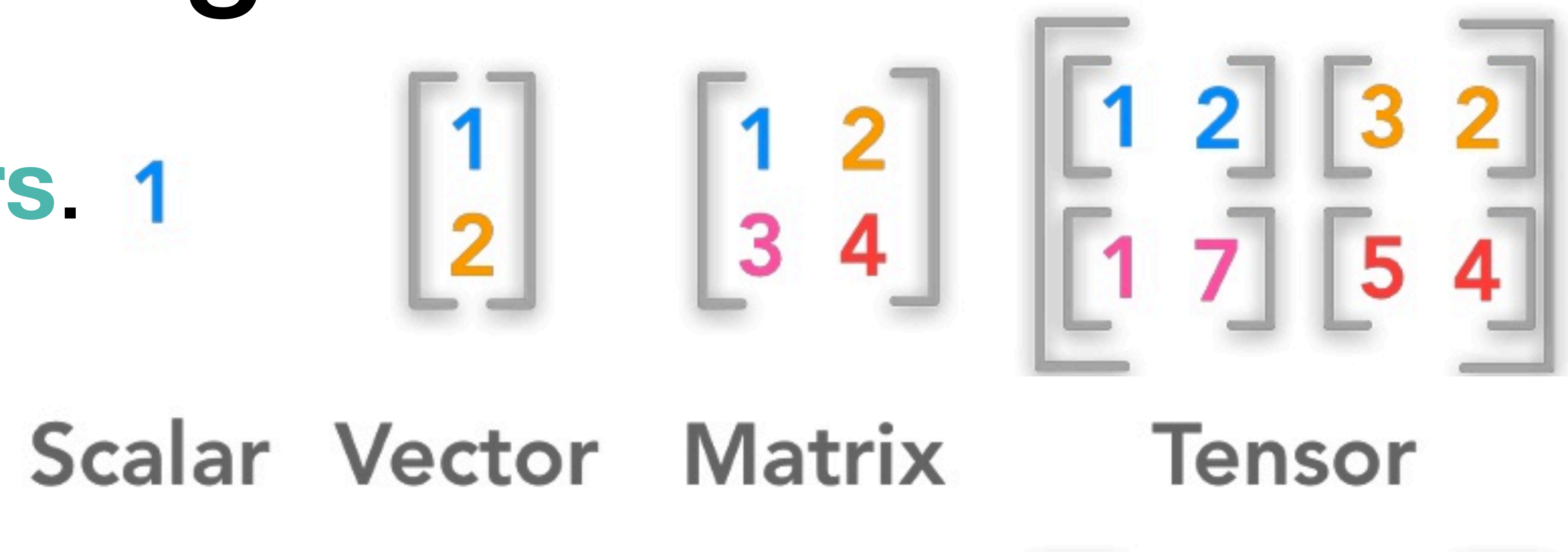


DL models are expressed as a DAG of **tensor operators**.

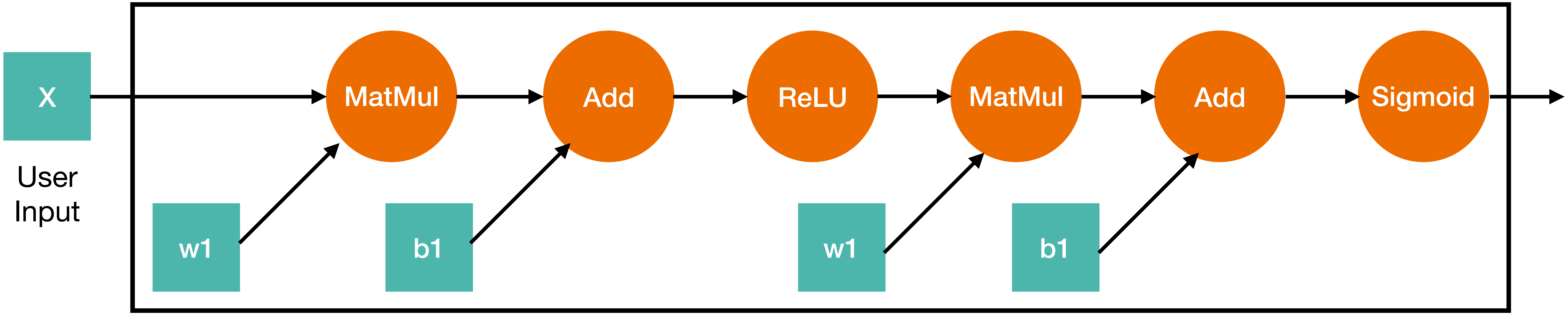


Deep Learning

Primarily relies on the abstraction of **tensors**.



DL models are expressed as a DAG of **tensor operators**.



Can contain 100s of operators with often 10s of unique operator types.

Systems for DL Prediction Serving

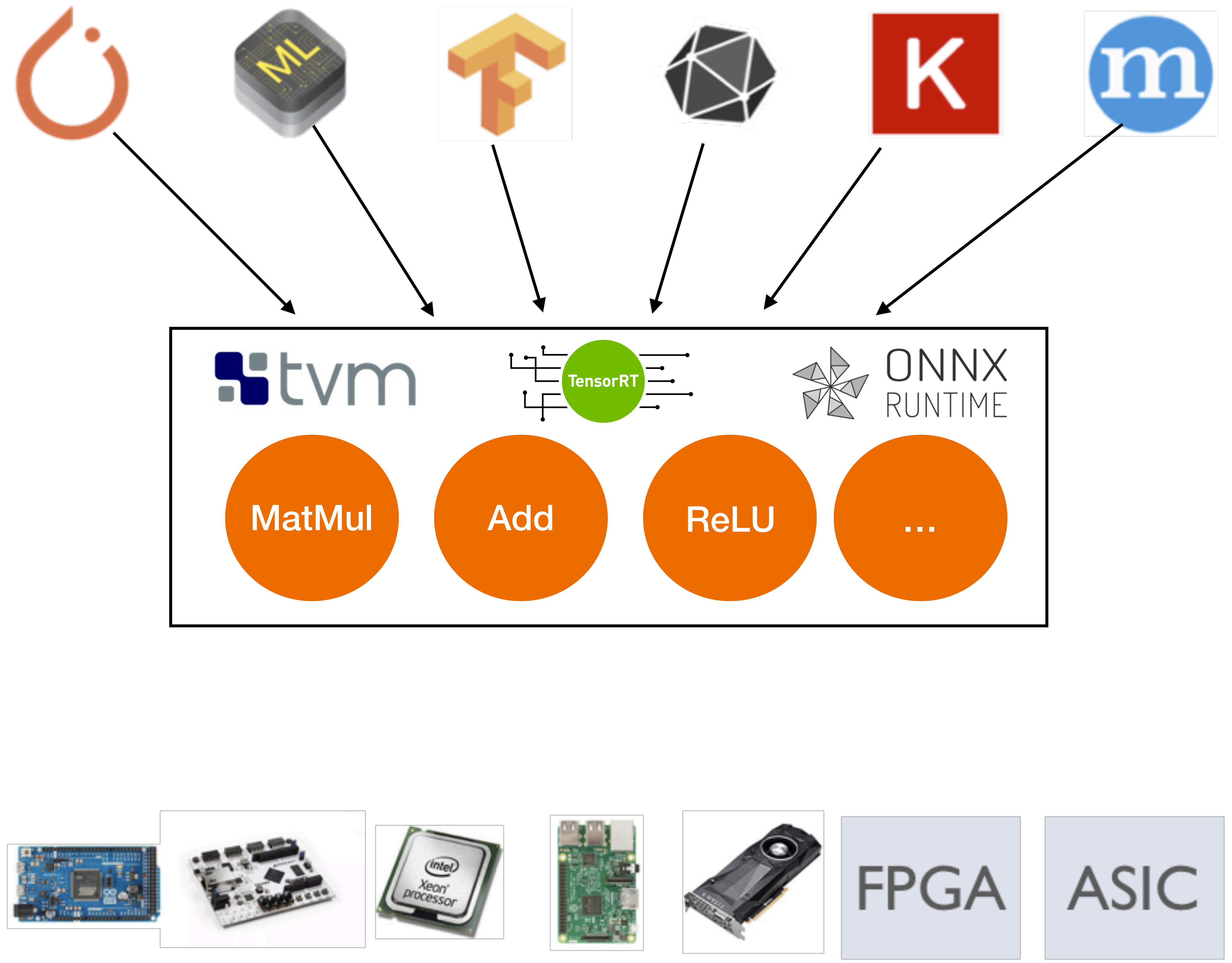
Exploit the abstraction of tensor operations to support multiple DL frameworks on multiple target environments.

Systems for DL Prediction Serving

Exploit the abstraction of tensor operations to support multiple DL frameworks on multiple target environments.

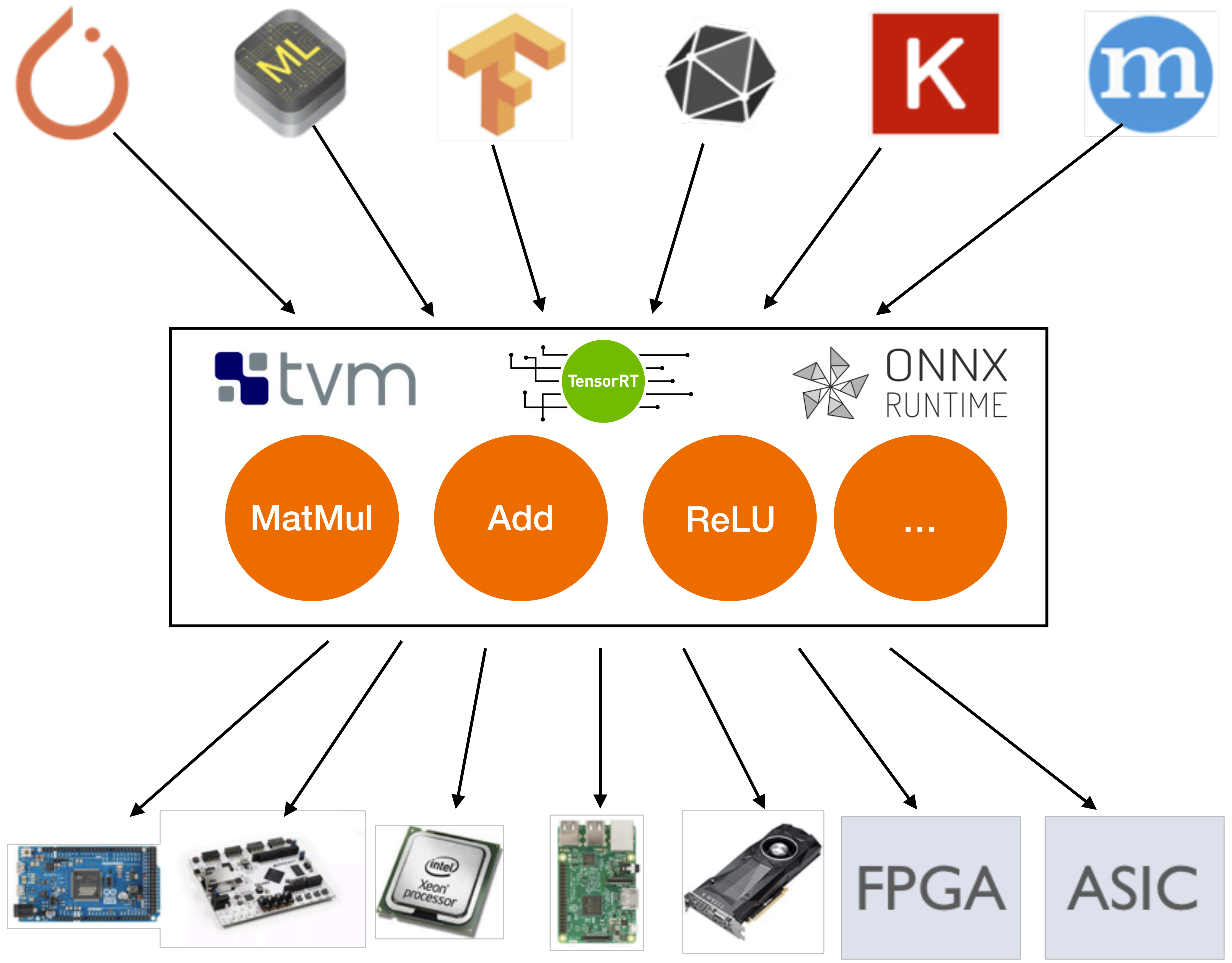
Systems for DL Prediction Serving

Exploit the abstraction of tensor operations to support multiple DL frameworks on multiple target environments.



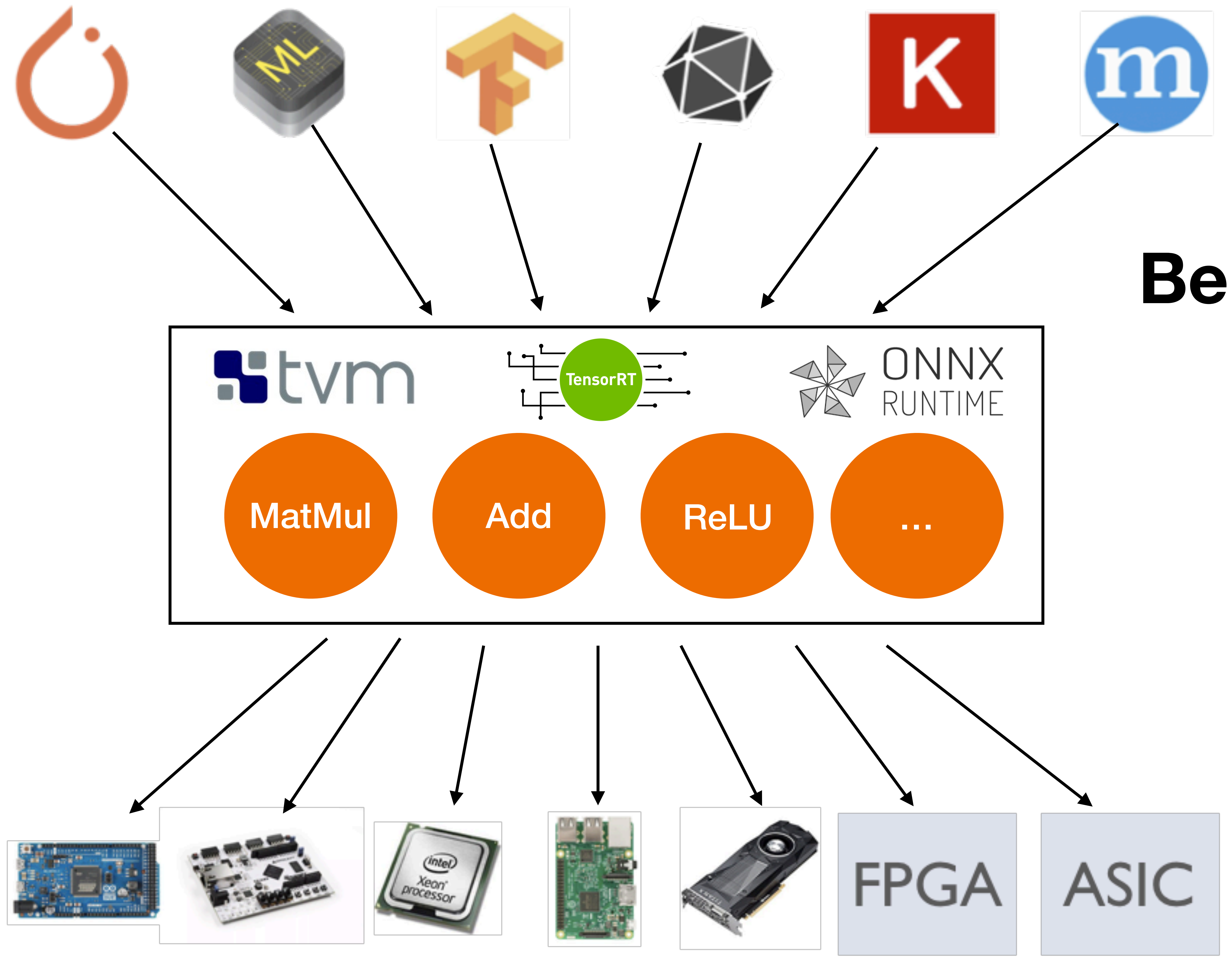
Systems for DL Prediction Serving

Exploit the abstraction of tensor operations to support multiple DL frameworks on multiple target environments.



Systems for DL Prediction Serving

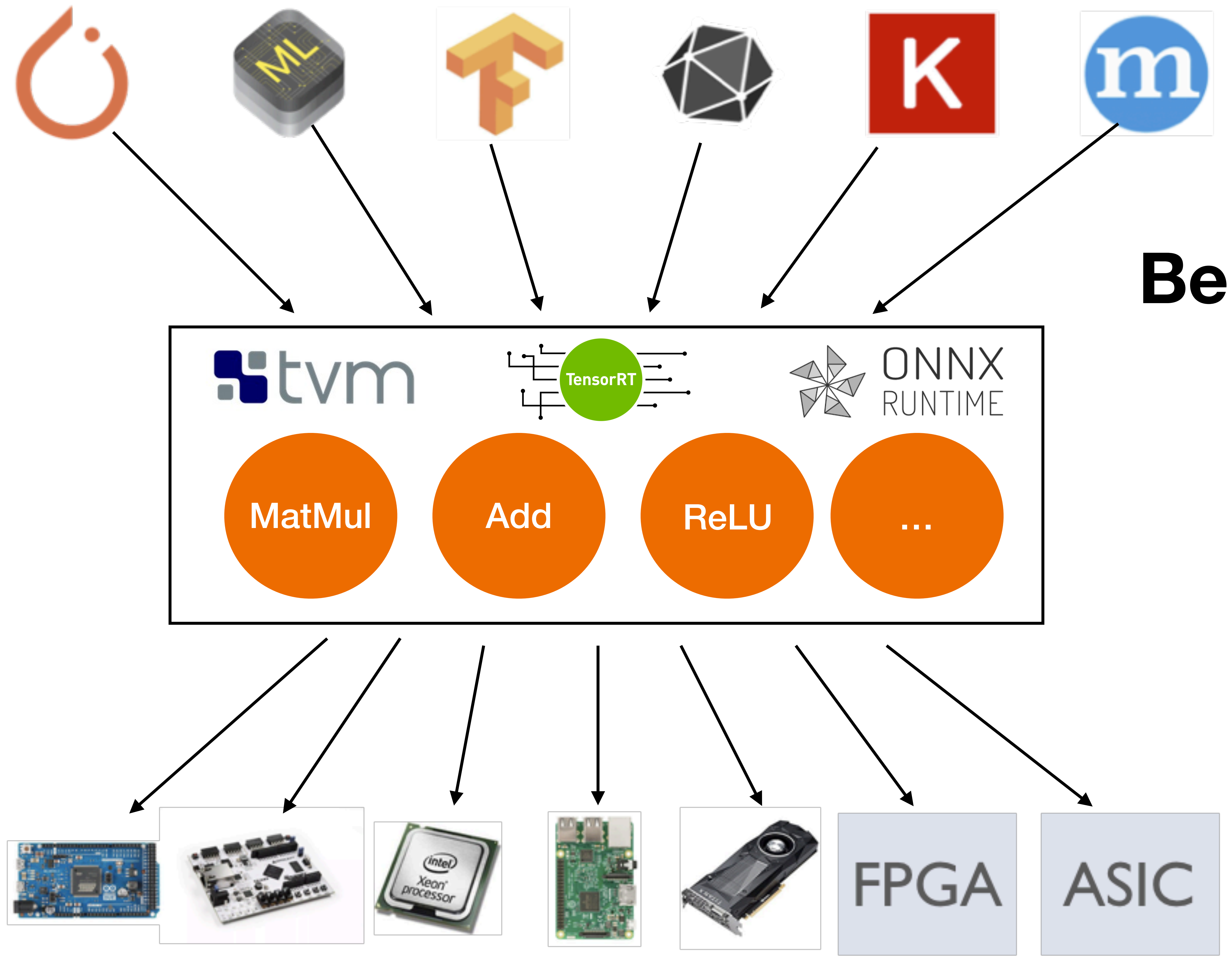
Exploit the abstraction of tensor operations to support multiple DL frameworks on multiple target environments.



Benefits: Target-independent and target-dependent optimization in a single place.

Systems for DL Prediction Serving

Exploit the abstraction of tensor operations to support multiple DL frameworks on multiple target environments.



Benefits:

Target-independent and target-dependent optimization in a single place.

Significantly reduced engineering efforts. Increased Portability.

Outline

1. Background

2. Our System: Hummingbird

   Main Idea: Compile Traditional ML Operators into Tensor Operators

   Example: Compiling Decision Tree-based Models

   High-level System Architecture

3. Experimental Evaluation

Main Idea: Compile Traditional ML Operators into Tensor Operators

Main Idea: Compile Traditional ML Operators into Tensor Operators

Focus: Traditional ML pipelines trained on structured data.

Main Idea: Compile Traditional ML Operators into Tensor Operators

Focus: Traditional ML pipelines trained on structured data.

Observation: Once trained, each operator can be represented as a transformation function that transforms input features into output features/score.

Main Idea: Compile Traditional ML Operators into Tensor Operators

Focus: Traditional ML pipelines trained on structured data.

Observation: Once trained, each operator can be represented as a transformation function that transforms input features into output features/score.

Often much simpler than the algorithm used during training.

Main Idea: Compile Traditional ML Operators into Tensor Operators

Focus: Traditional ML pipelines trained on structured data.

Observation: Once trained, each operator can be represented as a transformation function that transforms input features into output features/score.

Often much simpler than the algorithm used during training.

Algebraic Operations: E.g., Linear Regression $\boxed{\mathbf{Y = wX + b}}$

Algorithmic Operations: E.g., RandomForest, OneHotEncoder

Main Idea: Compile Traditional ML Operators into Tensor Operators

Focus: Traditional ML pipelines trained on structured data.

Observation: Once trained, each operator can be represented as a transformation function that transforms input features into output features/score.

Often much simpler than the algorithm used during training.

✓ Algebraic Operations: E.g., Linear Regression $\boxed{\mathbf{Y = wX + b}}$

Algorithmic Operations: E.g., RandomForest, OneHotEncoder

Main Idea: Compile Traditional ML Operators into Tensor Operators

Focus: Traditional ML pipelines trained on structured data.

Observation: Once trained, each operator can be represented as a transformation function that transforms input features into output features/score.

Often much simpler than the algorithm used during training.

Algebraic Operations: E.g., Linear Regression $\boxed{\mathbf{Y = wX + b}}$

Algorithmic Operations: E.g., RandomForest, OneHotEncoder

Complex data access patterns and control-flow patterns!

Main Idea: Compile Traditional ML Operators into Tensor Operators

Problem: How to compile algorithmic operators into tensor operators?

Main Idea: Compile Traditional ML Operators into Tensor Operators

Problem: How to compile algorithmic operators into tensor operators?

Our Solution: Introduce **redundancies**, both **computational** and **storage**, and make the data access patterns and control flow uniform for all inputs.

Main Idea: Compile Traditional ML Operators into Tensor Operators

Problem: How to compile algorithmic operators into tensor operators?

Our Solution: Introduce **redundancies**, both **computational** and **storage**, and make the data access patterns and control flow uniform for all inputs.

Depending on the level of redundancy introduced there can be more than one potential compilation approach.

Hummingbird picks the one that works best for the target setting.

Outline

1. Background

2. Our System: Hummingbird

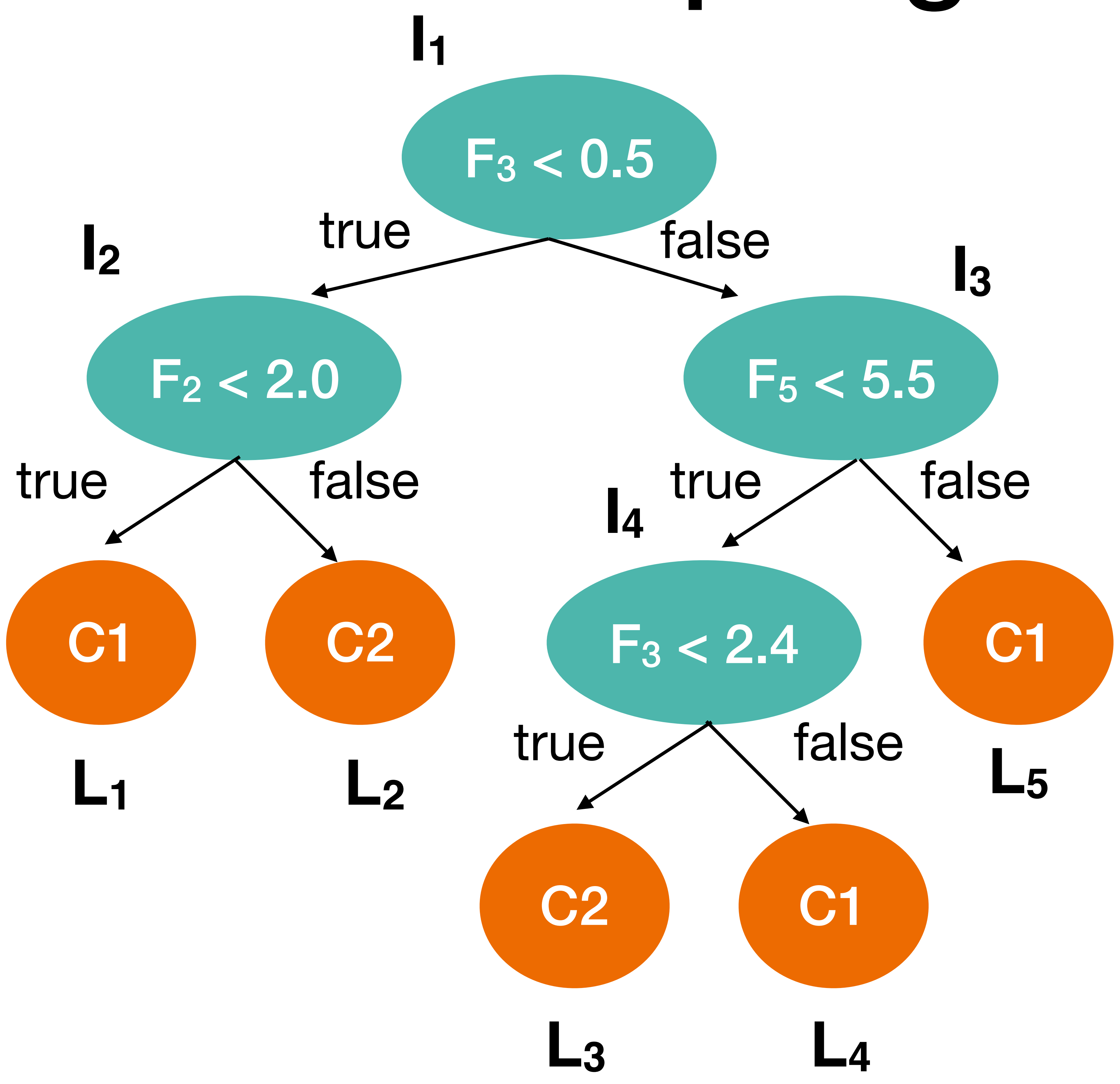
   Main Idea: Compile Traditional ML Operators into Tensor Operators

   Example: Compiling Decision Tree-based Models

   High-level System Architecture

3. Experimental Evaluation

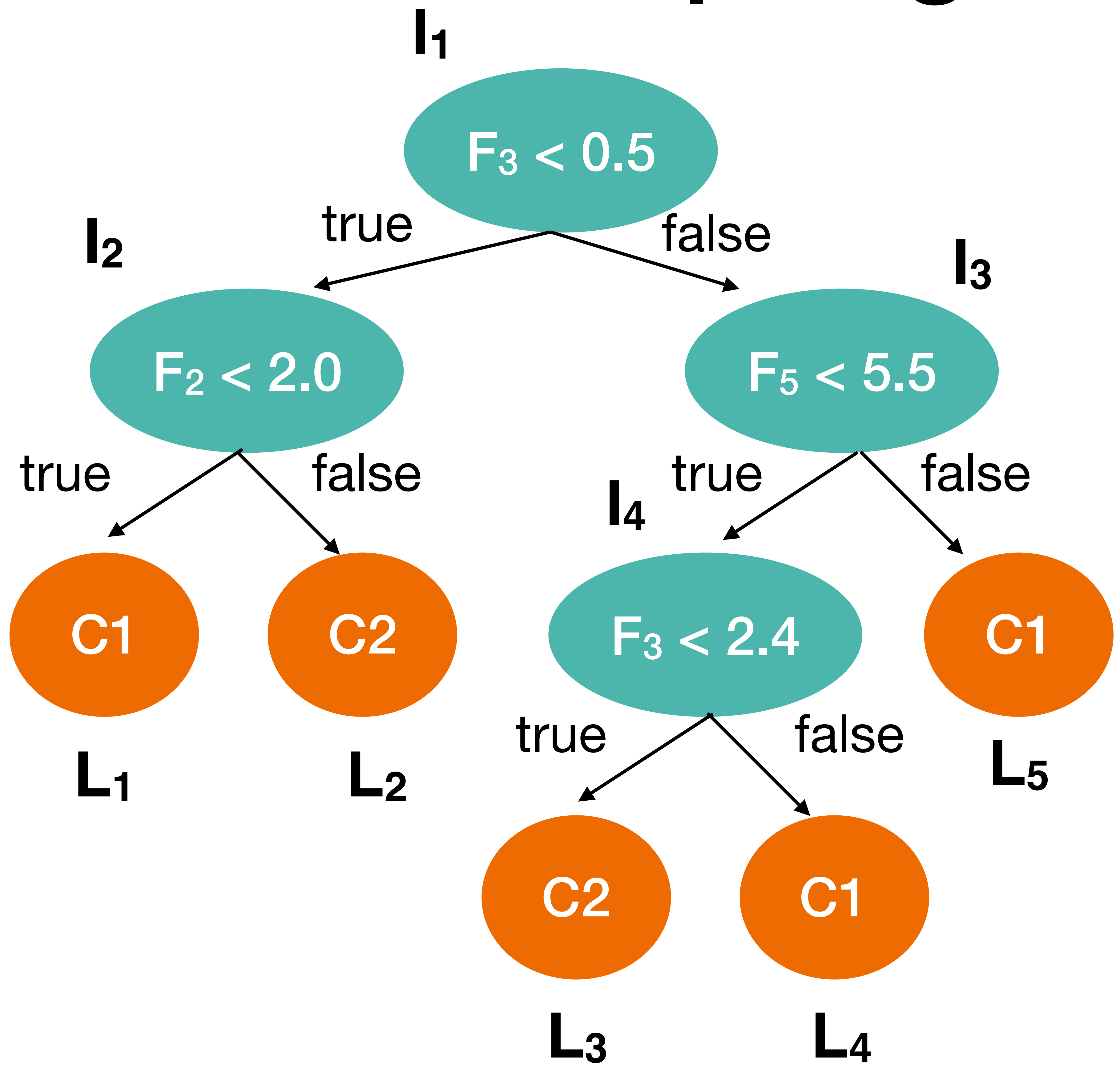
Compiling Decision Tree-based Models



F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

Compiling Decision Tree-based Models



F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

A

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

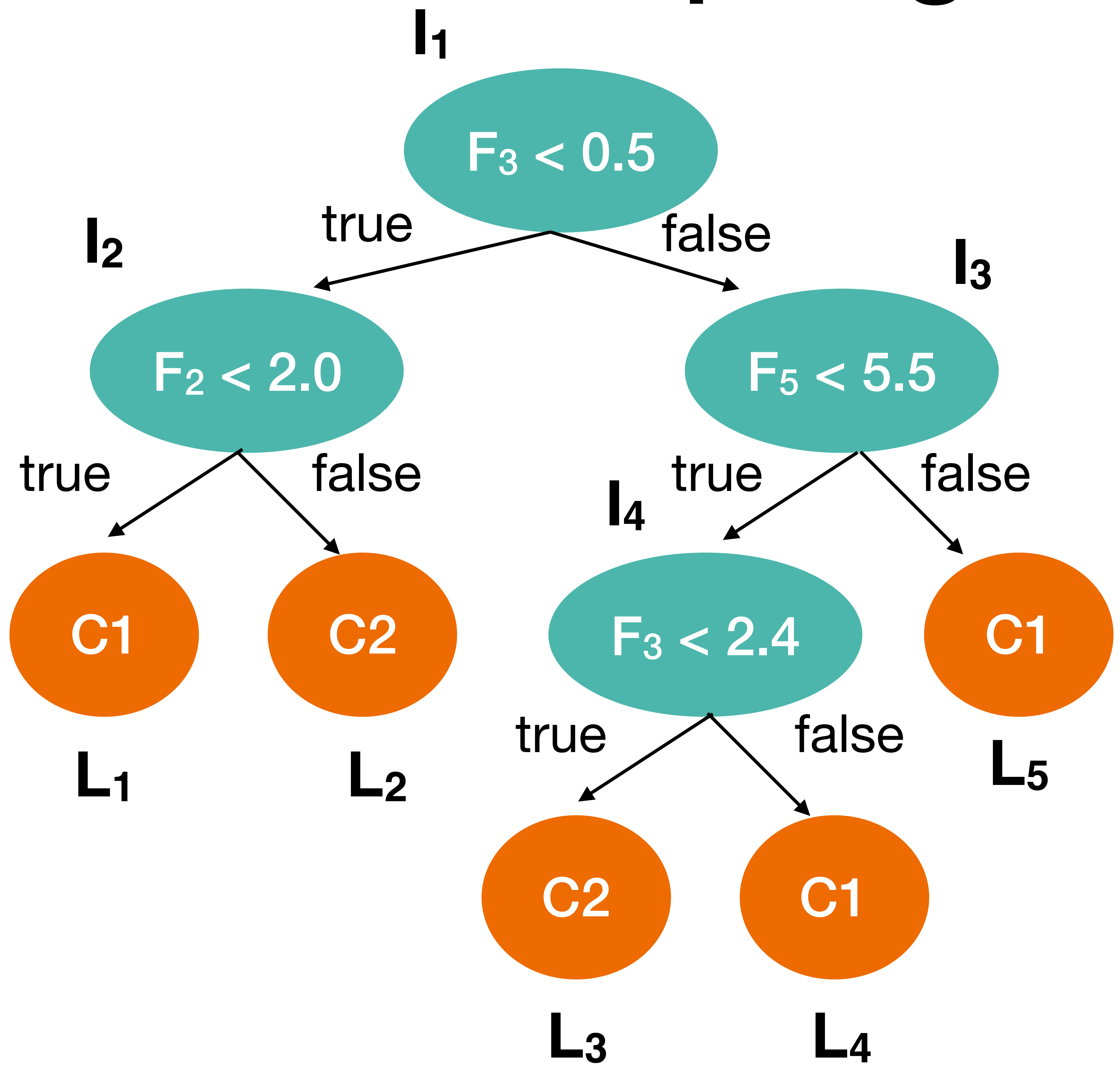
B

$$\begin{bmatrix} 0.5 & 2.0 & 5.5 & 2.4 \end{bmatrix}$$

$$A \in \mathbb{R}^{|F| \times |I|}$$

$$A_{i,j} = \begin{cases} 1, & I_j \textit{ evaluates } F_i \\ 0, & \textit{otherwise} \end{cases}$$

Compiling Decision Tree-based Models



F1	F2	F3	F4	F5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

A

0	0	0	0
0	1	0	0
1	0	0	1
0	0	0	0
0	0	1	0

$$A \in \mathbb{R}^{|F| \times |I|}$$

$$A_{i,j} = \begin{cases} 1, & I_j \ evaluates \ F_i \\ 0, & otherwise \end{cases}$$

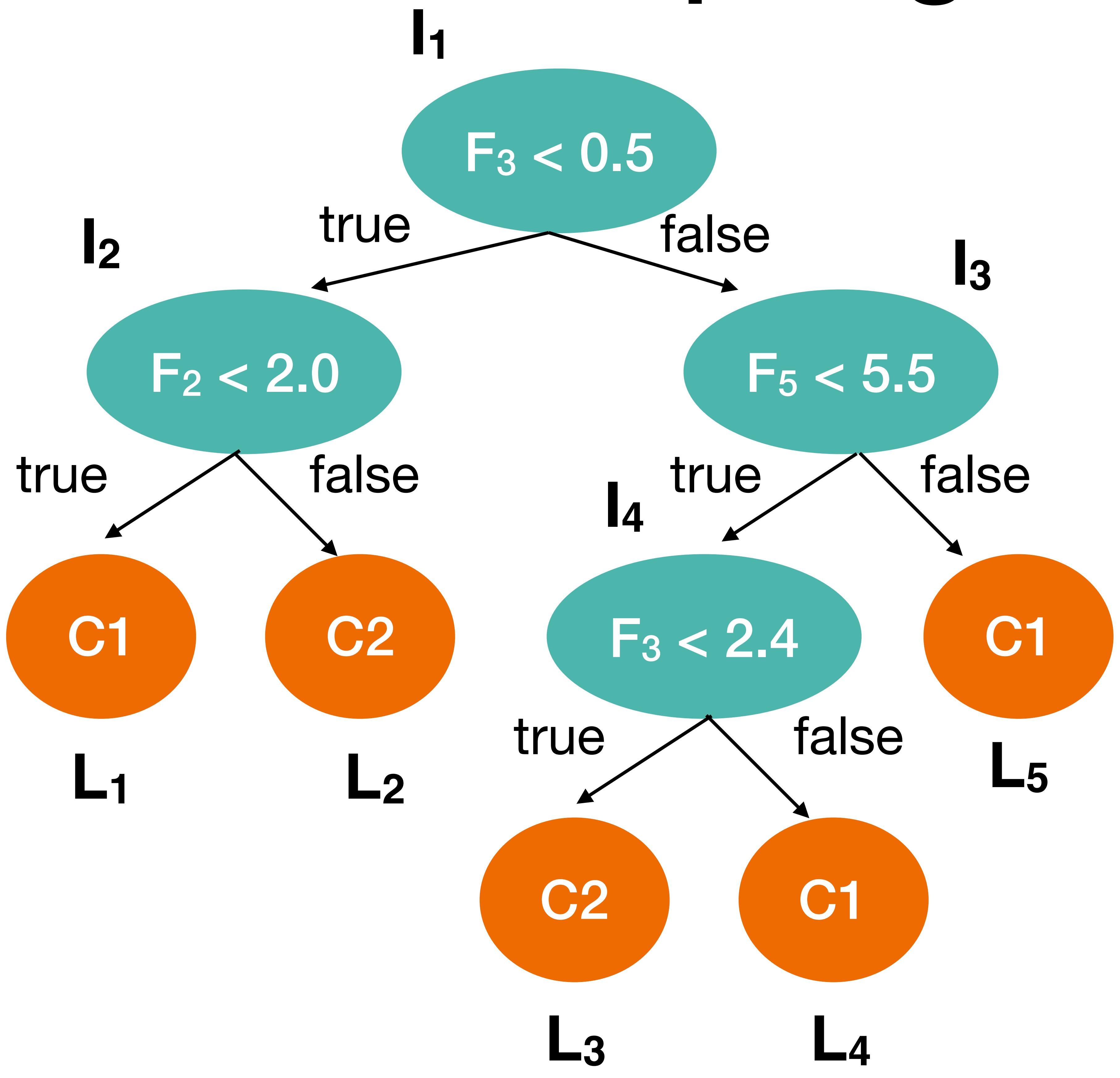
B

0.5	2.0	5.5	2.4

$$B \in \mathbb{R}^{|I|}$$

$$B_j = ThresholdValue(I_j)$$

Compiling Decision Tree-based Models



F1	F2	F3	F4	F5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

C

1	1	-1	-1	-1
1	-1	0	0	0
0	0	1	1	-1
0	0	1	-1	0

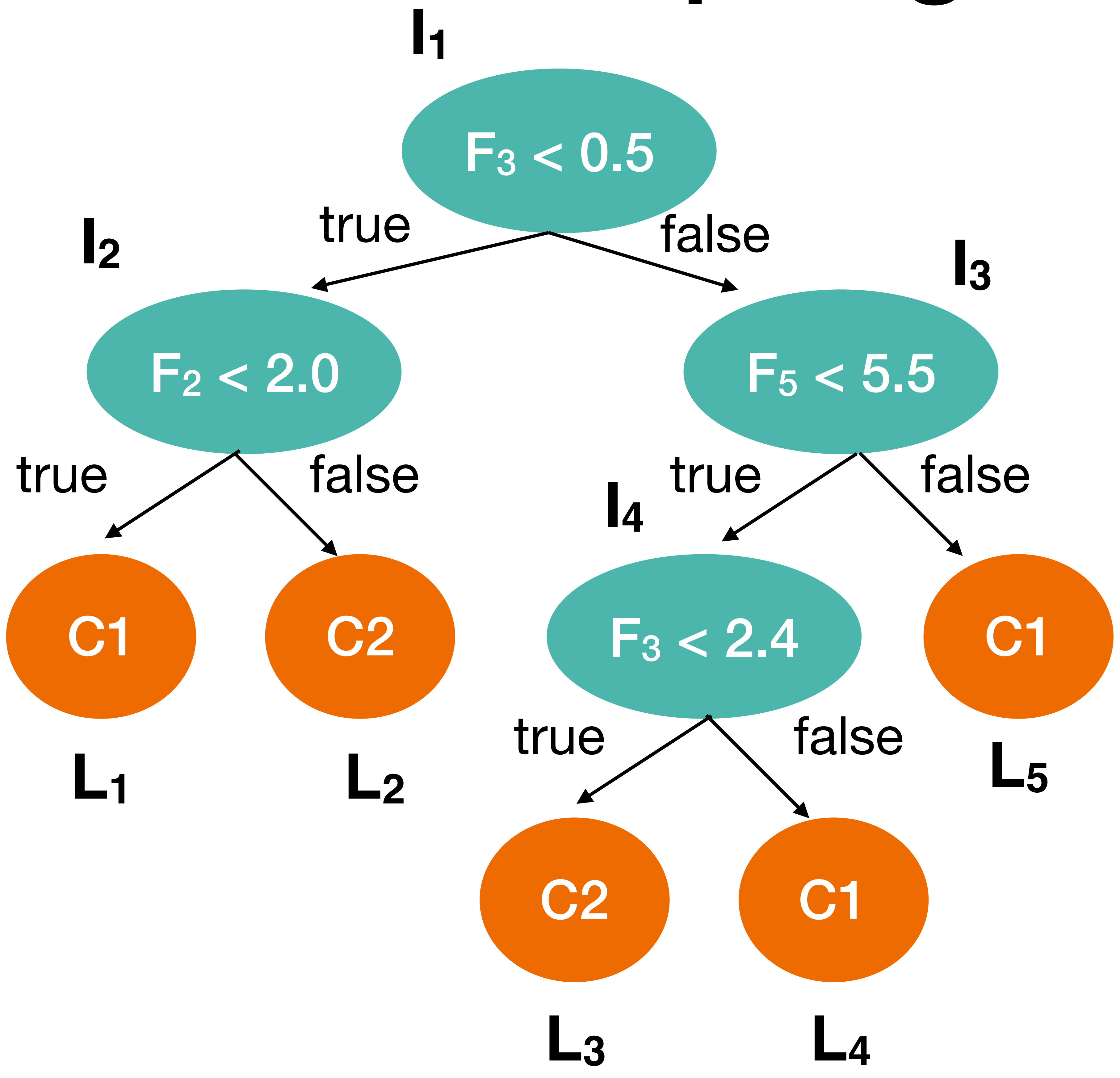
$$C \in \mathbb{R}^{|I|\times|L|}$$

$$C_{i,j} = \begin{cases} -1, \ L_j \in \ RightSubTree(I_i) \\ 1, \ L_j \in \ LeftSubTree(I_i) \\ 0, \ otherwise \end{cases}$$

D

2	1	2	1	0

Compiling Decision Tree-based Models



F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

C

1	1	-1	-1	-1
1	-1	0	0	0
0	0	1	1	-1
0	0	1	-1	0

$$C \in \mathbb{R}^{|I| \times |L|}$$

$$C_{i,j} = \begin{cases} -1, \ L_j \in \ RightSubTree(I_i) \\ 1, \ L_j \in \ LeftSubTree(I_i) \\ 0, \ otherwise \end{cases}$$

D

2	1	2	1	0

$$D \in \mathbb{R}^{|L|}$$

$$D_j = \sum_{k \in L_j \xrightarrow{path} Root} \mathbf{1}(k == \ LeftChild(Parent(k)))$$

Compiling Decision Tree-based Models

F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

X

A

0	0	0	0
0	1	0	0
1	0	0	1
0	0	0	0
0	0	1	0

Compiling Decision Tree-based Models

F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

x

A

0	0	0	0
0	1	0	0
1	0	0	1
0	0	0	0
0	0	1	0

=

1.9	4.6	0.1	10.1

Compiling Decision Tree-based Models

F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

x

A

0	0	0	0
0	1	0	0
1	0	0	1
0	0	0	0
0	0	1	0

=

1.9	4.6	0.1	10.1

1.9	4.6	0.1	10.1

==

B

0.5	2.0	5.5	2.4

Compiling Decision Tree-based Models

F_1	F_2	F_3	F_4	F_5
0.1	4.5	1.9	10.1	3.5

F (Feature Vector)

x

A

0	0	0	0
0	1	0	0
1	0	0	1
0	0	0	0
0	0	1	0

=

1.9	4.6	0.1	10.1

1.9	4.6	0.1	10.1

==

B

0.5	2.0	5.5	2.4

=

I_1	I_2	I_3	I_4
0	0	1	1

Compiling Decision Tree-based Models

$$
\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}
\times
\begin{bmatrix} 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix}
$$

I_1	I_2	I_3	I_4
0	0	1	1

x

C

1	1	-1	-1	-1
1	-1	0	0	0
0	0	1	1	-1
0	0	1	-1	0

Compiling Decision Tree-based Models

$$
\begin{array}{cccc} I_1 & I_2 & I_3 & I_4 \\ \hline 0 & 0 & 1 & 1 \end{array}
\times
\overset{C}{\begin{bmatrix} 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix}}
=
\begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix}
$$

Compiling Decision Tree-based Models

$$\begin{array}{cccc} I_1 & I_2 & I_3 & I_4 \end{array}$$

$$\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} \times \underbrace{\begin{bmatrix} 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix}}_{\text{ }}^{} = \begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix}$$

C

D

$$\begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix} == \begin{bmatrix} 2 & 1 & 2 & 1 & 0 \end{bmatrix}$$

Compiling Decision Tree-based Models

$$
\begin{array}{cccc} I_1 & I_2 & I_3 & I_4 \\ \end{array}
\qquad
C
$$

$$
\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}
\times
\begin{bmatrix} 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix}
=
\begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix}
$$

$$
\begin{array}{c} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix} \end{array}
==
\begin{array}{c} D \\ \begin{bmatrix} 2 & 1 & 2 & 1 & 0 \end{bmatrix} \end{array}
$$

$$
= \begin{array}{ccccc} L_1 & L_2 & L_3 & L_4 & L_5 \\ \hline 0 & 0 & 0 & 1 & 0 \end{array}
$$

Compiling Decision Tree-based Models

$$
\begin{array}{cccc} I_1 & I_2 & I_3 & I_4 \\ \hline 0 & 0 & 1 & 1 \end{array}
\;\times\;
\overset{\textbf{C}}{\begin{bmatrix} 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix}}
\;=\;
\begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix}
$$

$$
\begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix}
\;==\;
\overset{\textbf{D}}{\begin{bmatrix} 2 & 1 & 2 & 1 & 0 \end{bmatrix}}
$$

$$
= \begin{array}{ccccc} L_1 & L_2 & L_3 & L_4 & L_5 \\ \hline 0 & 0 & 0 & 1 & 0 \end{array}
$$

This technique can be easily adopted for tree-ensembles by batching individual tensors for each tree.

Compiling Decision Tree-based Models

Above approach (**GEMM** approach) essentially evaluates all paths in a decision tree model: **computation redundancy**.

Compiling Decision Tree-based Models

Above approach (**GEMM** approach) essentially evaluates all paths in a decision tree model: **computation redundancy**.

Encoding tree structure using tensors introduce **storage redundancy**.

Compiling Decision Tree-based Models

Above approach (**GEMM** approach) essentially evaluates all paths in a decision tree model: **computation redundancy**.

Encoding tree structure using tensors introduce **storage redundancy**.

Works surprisingly well on modern hardware for many cases!

Compiling Decision Tree-based Models

Above approach (**GEMM** approach) essentially evaluates all paths in a decision tree model: **computation redundancy**.

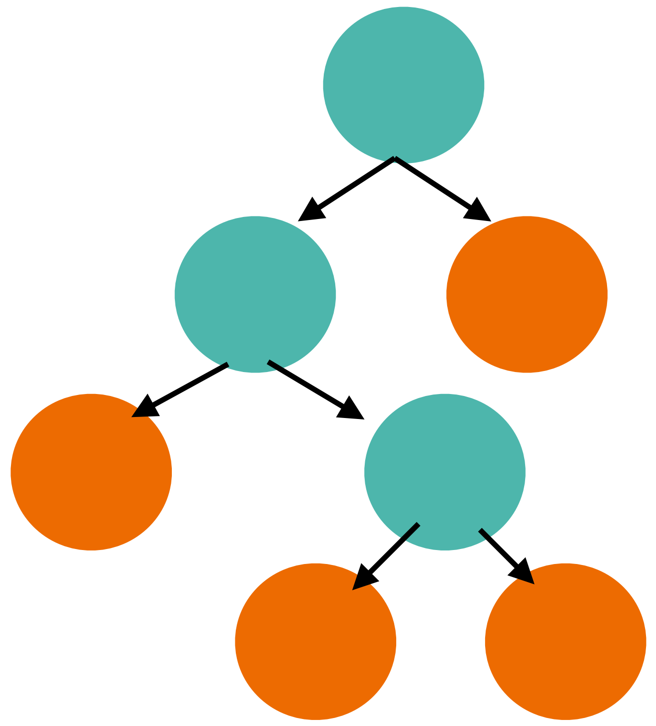
Encoding tree structure using tensors introduce **storage redundancy**.

Works surprisingly well on modern hardware for many cases!

Two other tree traversal-based methods that exploit the tree structure.

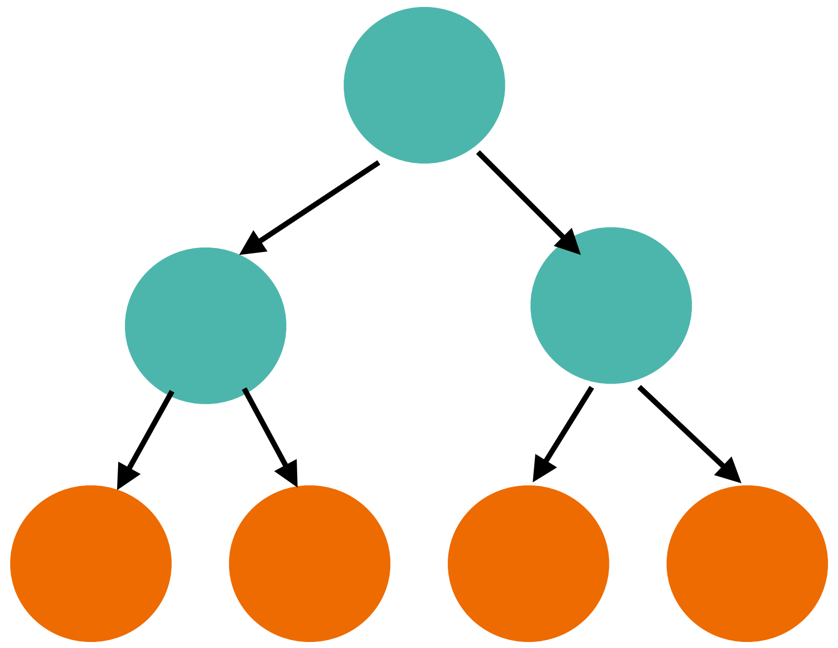
TreeTraversal

For tall trees (e.g., LightGBM)



PerfectTreeTraversal

For bushy trees (e.g., XGBoost)



Compiling Decision Tree-based Models

Above approach (**GEMM** approach) essentially evaluates all paths in a decision tree model: **computation redundancy**.

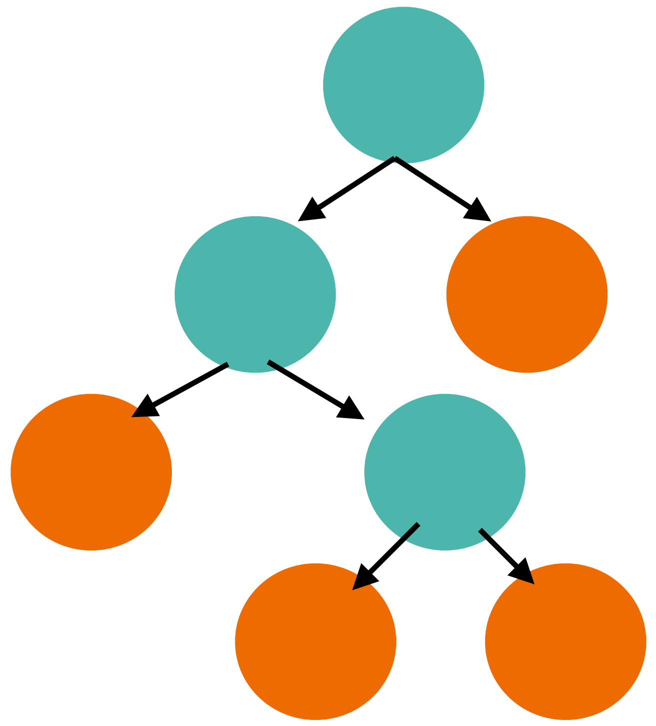
Encoding tree structure using tensors introduce **storage redundancy**.

Works surprisingly well on modern hardware for many cases!

Two other tree traversal-based methods that exploit the tree structure.

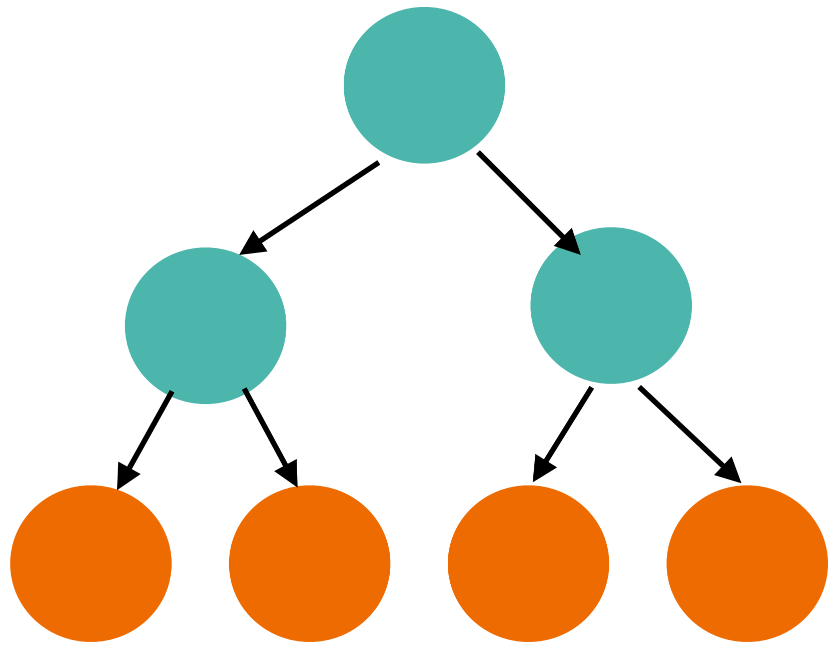
TreeTraversal

For tall trees (e.g., LightGBM)



PerfectTreeTraversal

For bushy trees (e.g., XGBoost)



More details about these methods and a summary of techniques used to compile 40+ Scikit-Learn ops can be found in our paper.

Outline

1. Background

2. Our System: Hummingbird

    Main Idea: Compile Traditional ML Operators into Tensor Operators

    Example: Compiling Decision Tree-based Models

    High-level System Architecture

3. Experimental Evaluation

High-level System Architecture

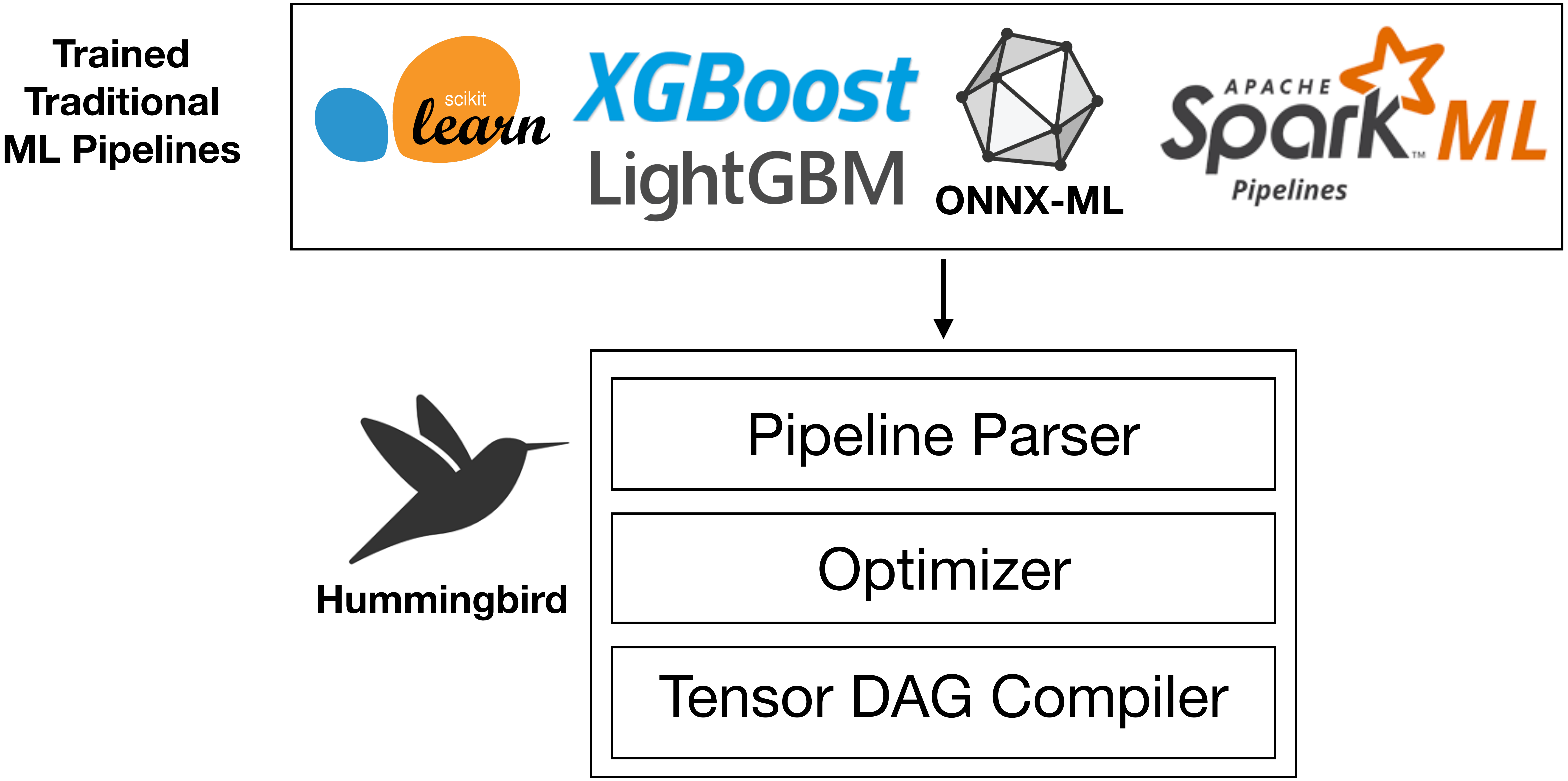
Hummingbird

Pipeline Parser

Optimizer

Tensor DAG Compiler

High-level System Architecture



High-level System Architecture

Trained Traditional ML Pipelines

scikit learn | XGBoost | LightGBM | ONNX-ML | APACHE Spark ML Pipelines

↓

Hummingbird

- Pipeline Parser
- Optimizer
- Tensor DAG Compiler

Optimizations:

Heuristics-based strategy selection

Feature selection push-down

Algebraic rewrites

Batching stacked models

(More details in our paper)

High-level System Architecture

Trained Traditional ML Pipelines

scikit learn | XGBoost | LightGBM | ONNX-ML | APACHE Spark ML Pipelines

↓

Hummingbird

- Pipeline Parser
- Optimizer
- Tensor DAG Compiler

↓

DL Prediction Serving Systems

tvm | ONNX RUNTIME | PyTorch/TorchScript

Optimizations:

Heuristics-based strategy selection

Feature selection push-down

Algebraic rewrites

Batching stacked models

(More details in our paper)

High-level System Architecture

Trained Traditional ML Pipelines

scikit learn · XGBoost · LightGBM · ONNX-ML · APACHE Spark ML Pipelines · ML.NET · H_2O.ai

↓

Hummingbird

- Pipeline Parser
- Optimizer
- Tensor DAG Compiler

↓

DL Prediction Serving Systems

tvm · ONNX RUNTIME · PyTorch/TorchScript · TensorRT · TF XLA

Optimizations:

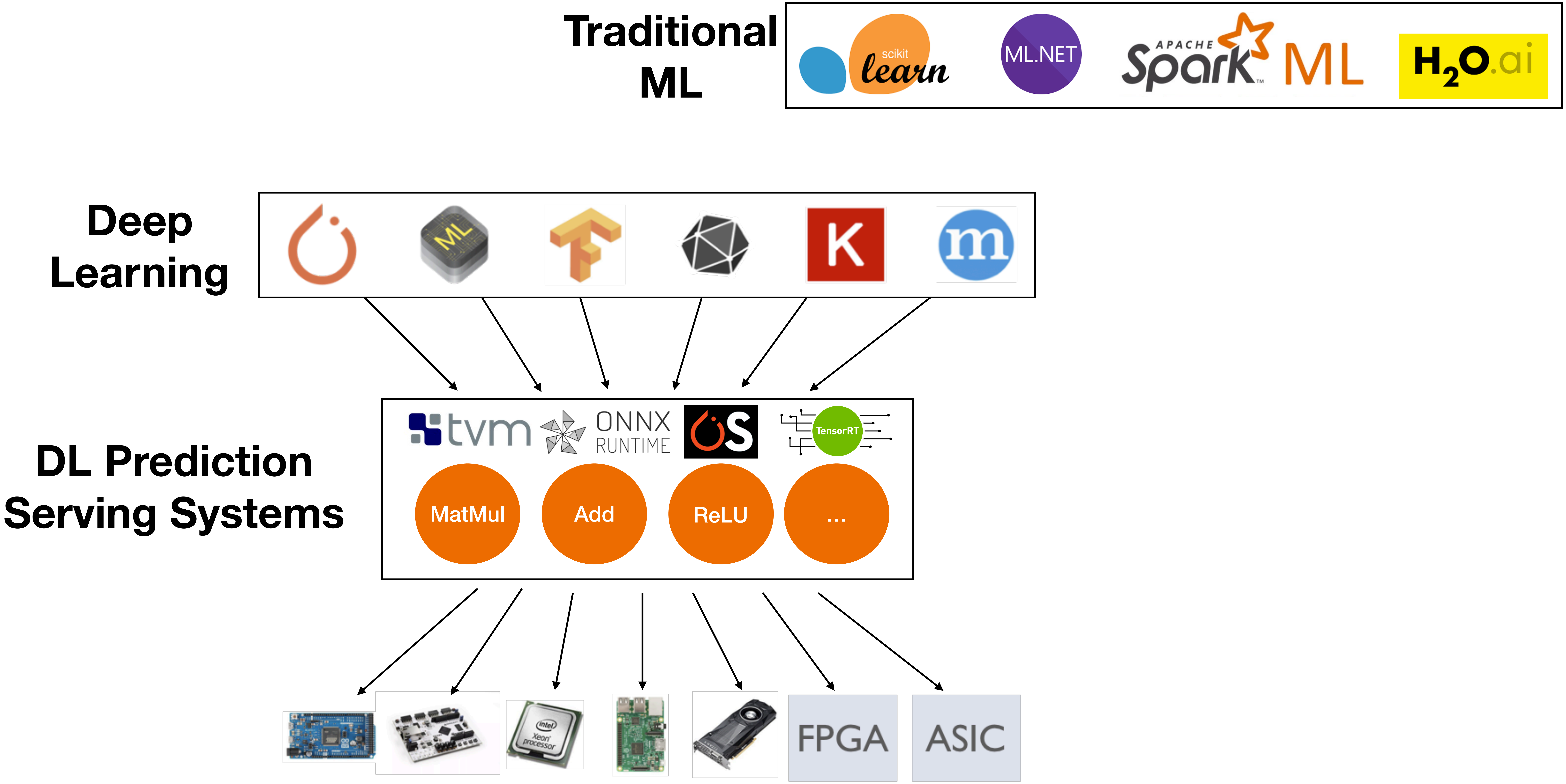
Heuristics-based strategy selection

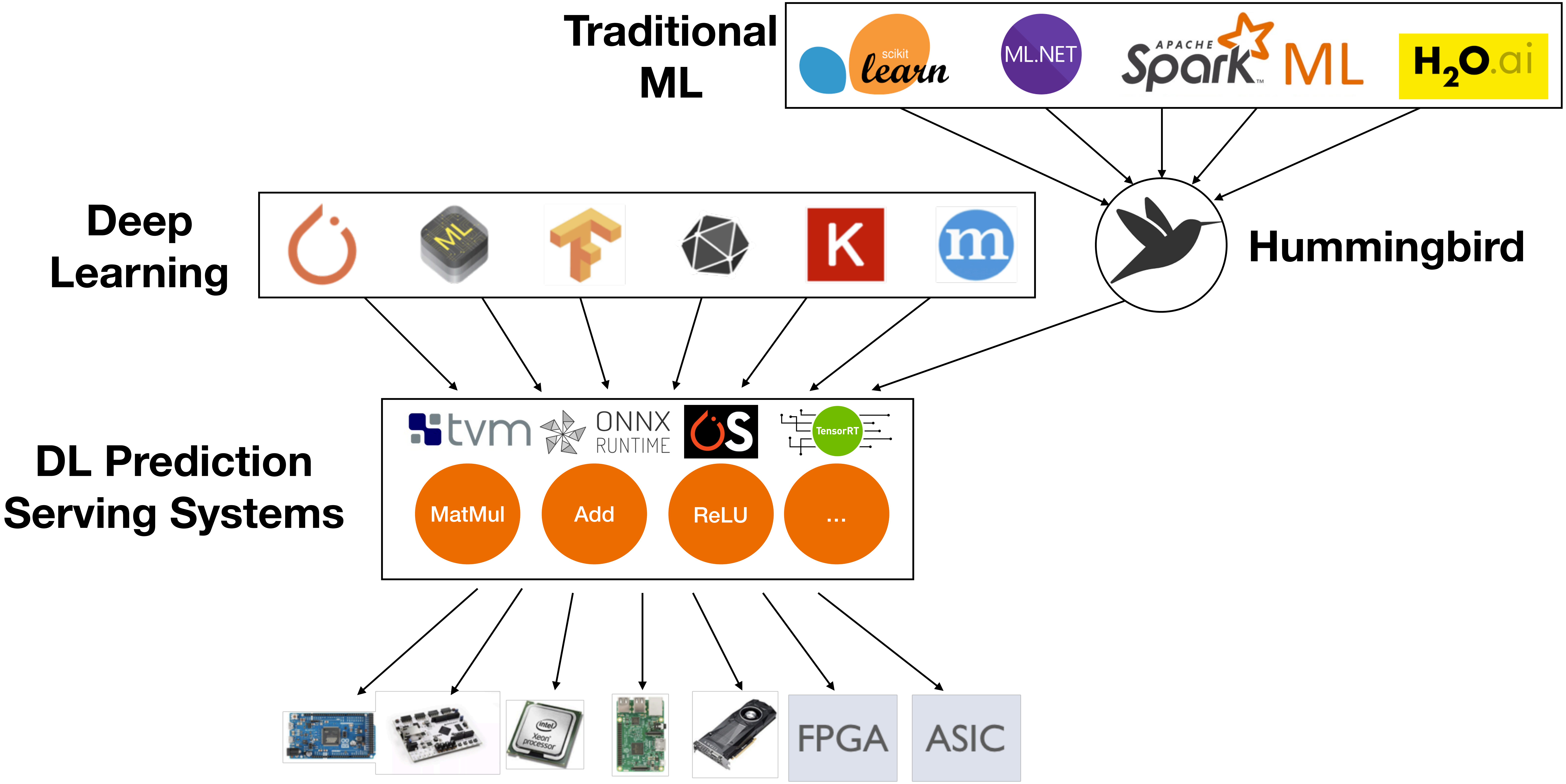
Feature selection push-down

Algebraic rewrites

Batching stacked models

(More details in our paper)


Traditional ML
scikit learn
ML.NET
APACHE Spark ML
H2O.ai
Deep Learning
DL Prediction Serving Systems
tvm
ONNX RUNTIME
TensorRT
MatMul
Add
ReLU
...
FPGA
ASIC


Traditional ML
scikit learn
ML.NET
APACHE Spark ML
H2O.ai
Deep Learning
Hummingbird
DL Prediction Serving Systems
tvm
ONNX RUNTIME
TensorRT
MatMul
Add
ReLU
...
FPGA
ASIC

Outline

1. Background

2. Our System: Hummingbird

3. Experimental Evaluation

End-to-End Pipeline Evaluation

Hardware Setup

Azure NC6 v2 machine

Intel Xeon E5-2690 v4@ 2.6GHz (6 cores)

112 GB RAM

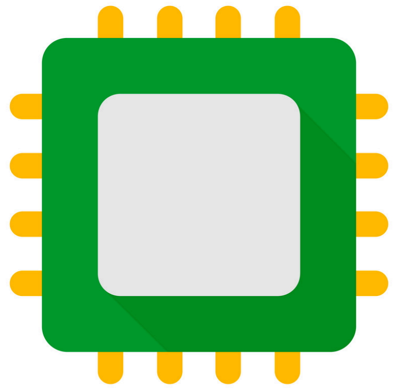
Nvidia P100

Ubuntu 18.04,
PyTorch 1.3, TVM 0.6, CUDA 10,
RAPIDS 0.9

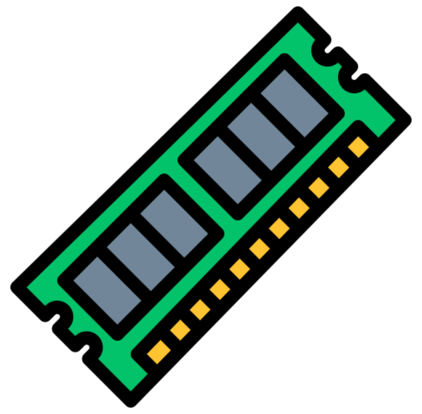
End-to-End Pipeline Evaluation

Hardware Setup

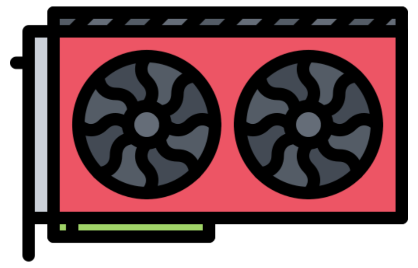
Azure NC6 v2 machine



Intel Xeon E5-2690
v4@ 2.6GHz (6 cores)



112 GB RAM



Nvidia P100

Ubuntu 18.04,
PyTorch 1.3, TVM 0.6, CUDA 10,
RAPIDS 0.9

Experimental Workload

Scikit-Learn pipelines for OpenML-CC18 benchmark which has 72 datasets.

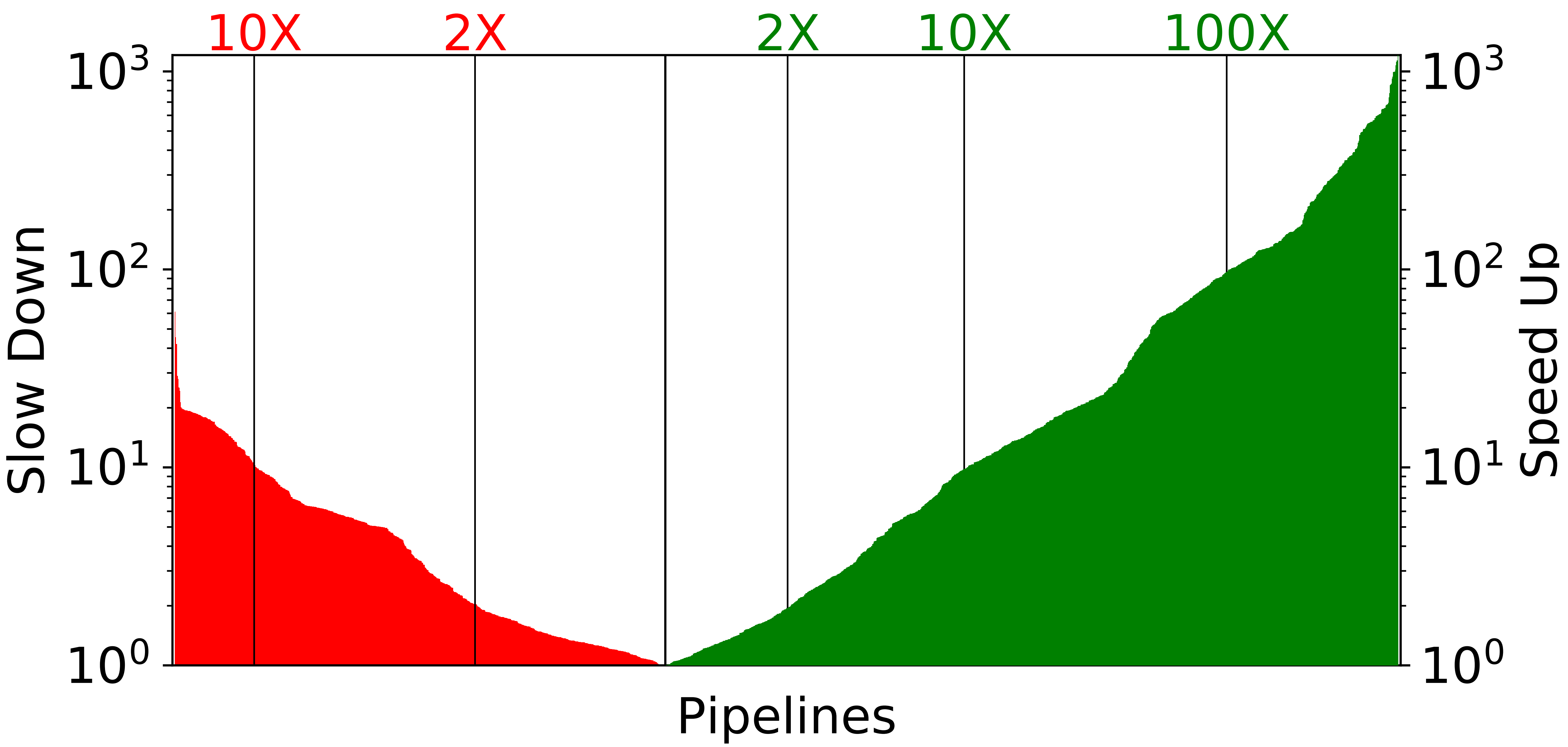
Hummingbird can translate 2328 pipelines (88%).

Perform inference on 20% of the dataset.

TorchScript as the backend for Hummingbird.

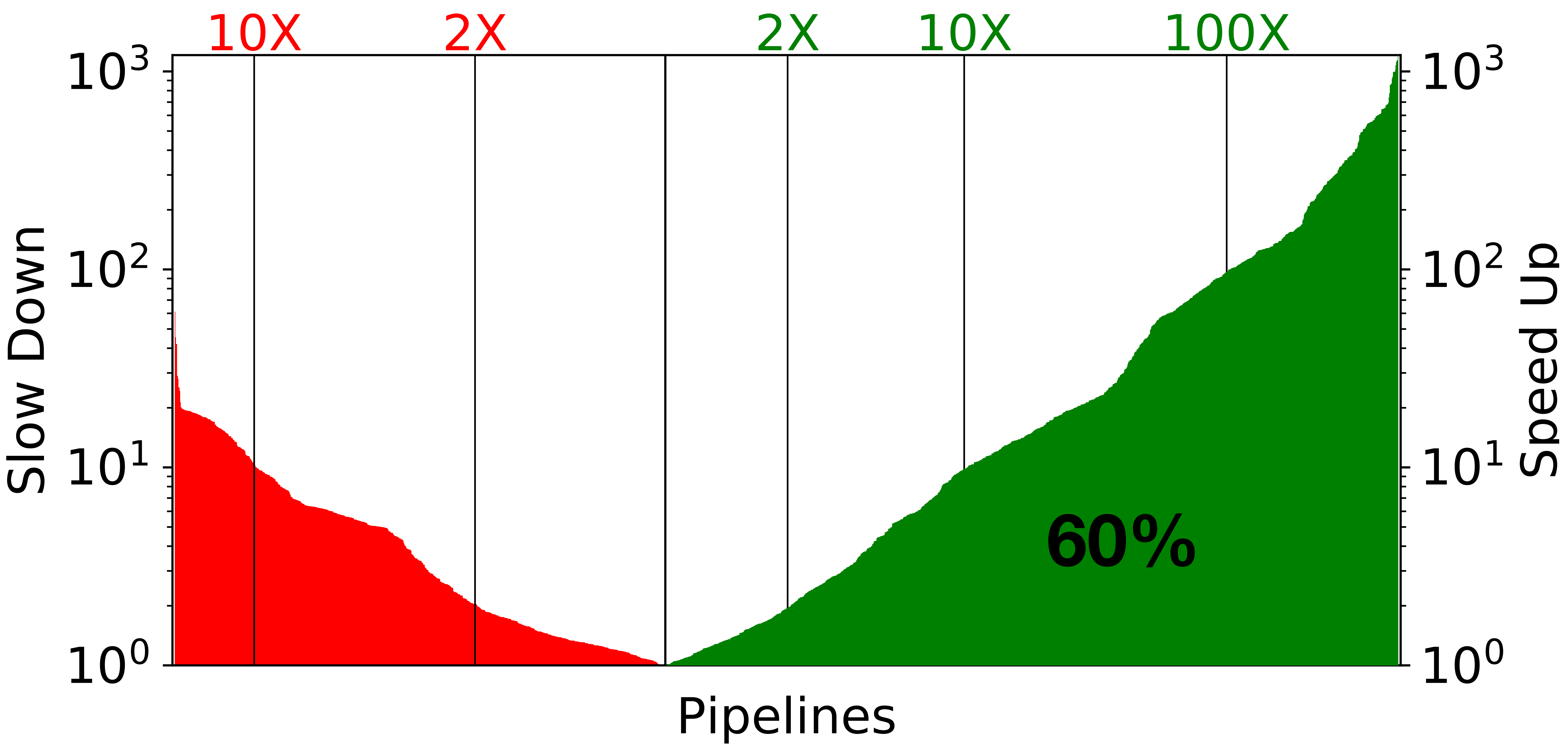
End-to-End Pipeline Evaluation

CPU

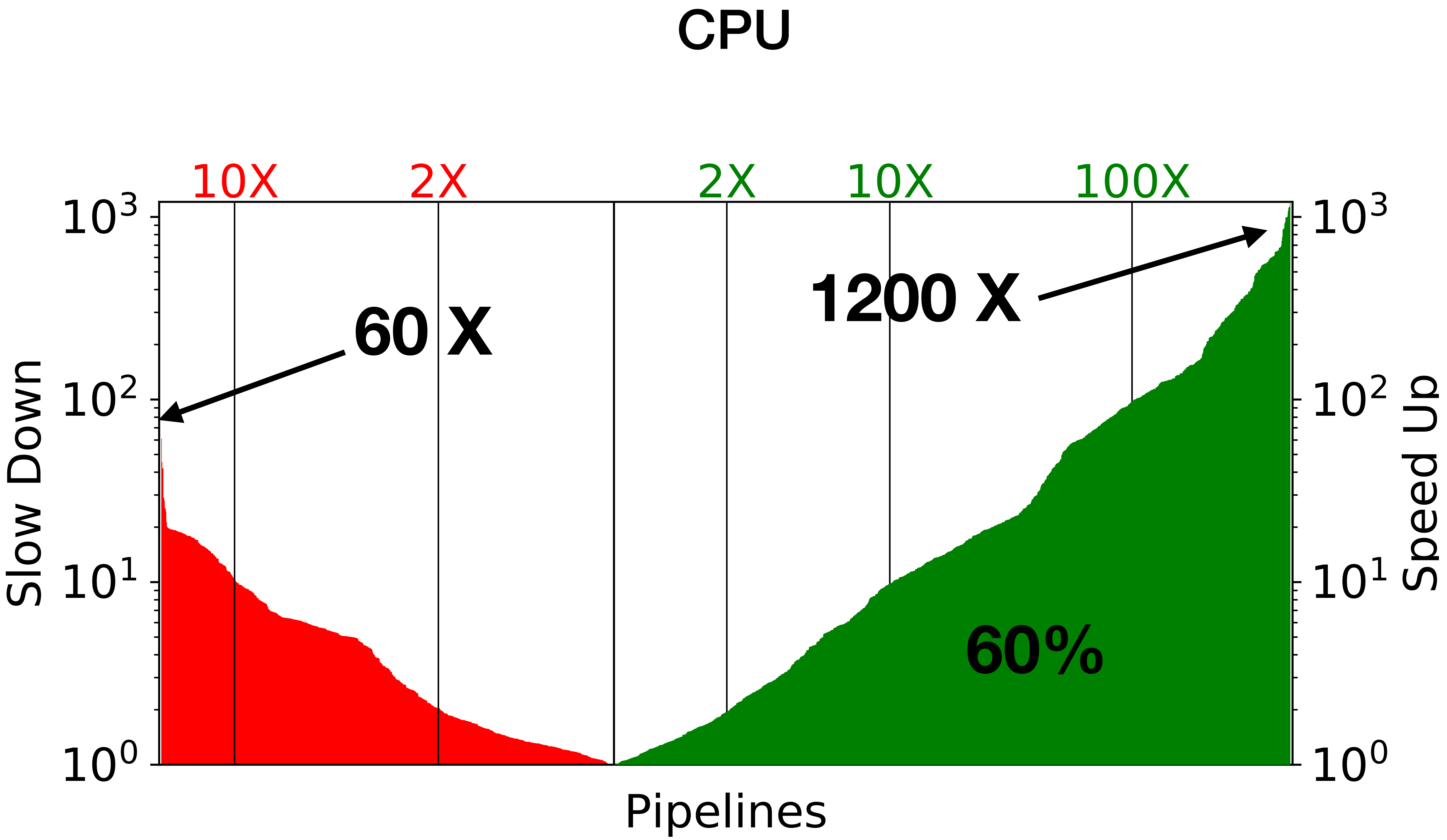


End-to-End Pipeline Evaluation

CPU

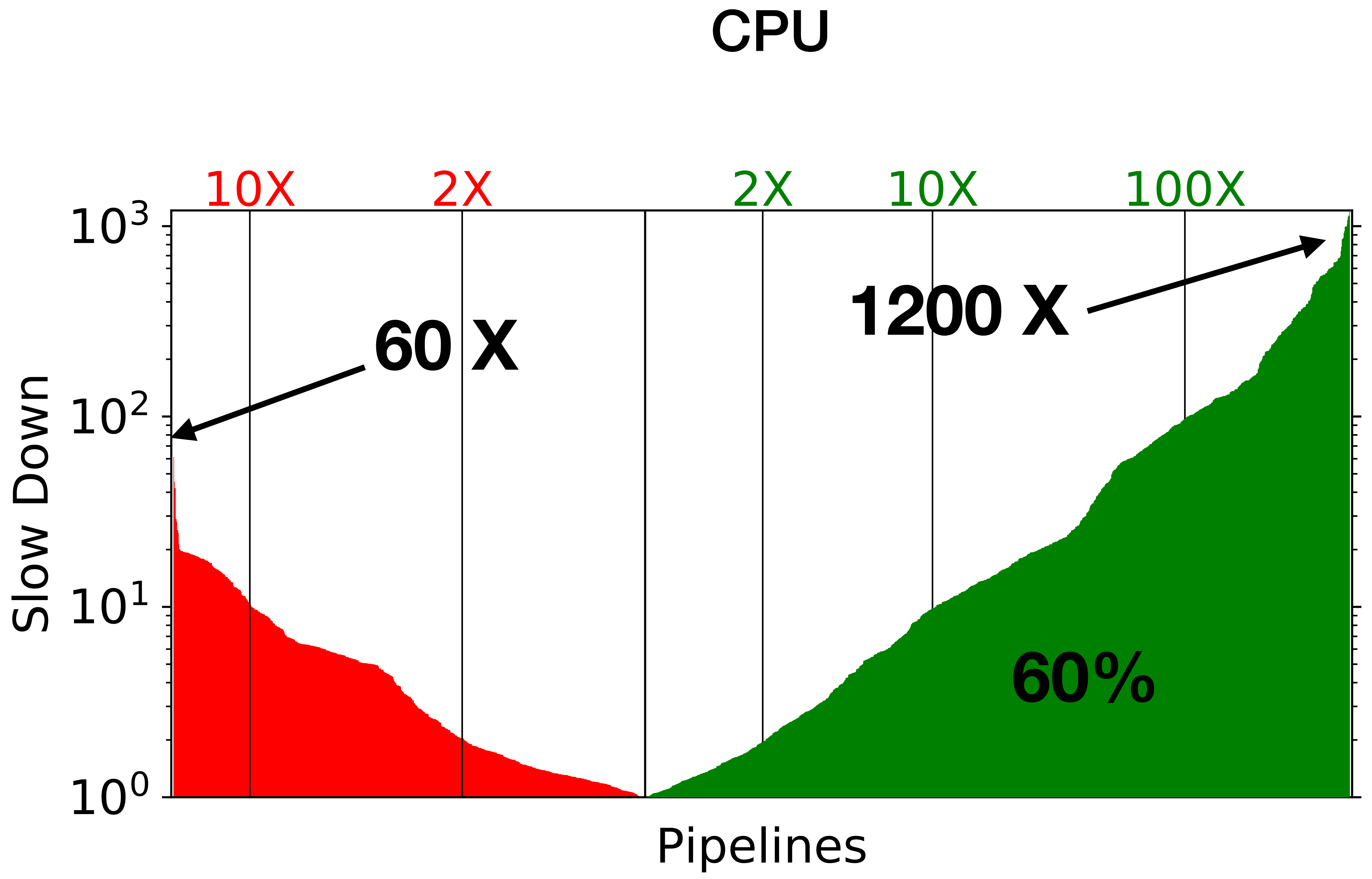


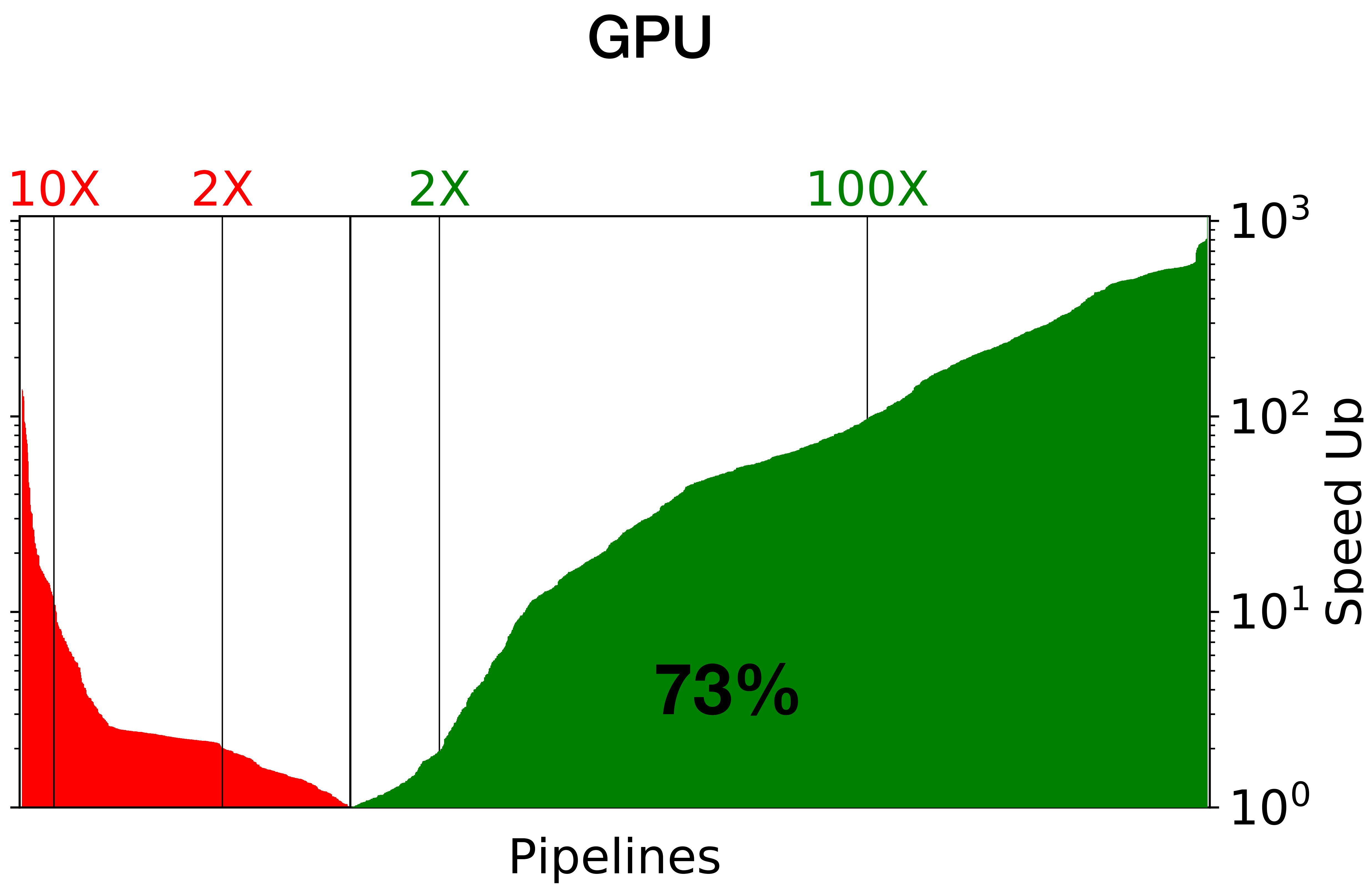
End-to-End Pipeline Evaluation



End-to-End Pipeline Evaluation

End-to-End Pipeline Evaluation





End-to-End Pipeline Evaluation

End-to-End Pipeline Evaluation

Main reasons for slowdowns: Sparse input data, small inference datasets.

Tree-Models Microbenchmark

Experimental Workload: Nvidia Gradient Boosting Algorithm Benchmark*

Dataset	Rows	#Features	Task
Fraud	285k	28	BinaryClass
Year	512k	90	Regression
Covtype	581k	54	MultiClass
Epsilon	500k	2000	BinaryClass

(* https://github.com/NVIDIA/gbm-bench)

Tree-Models Microbenchmark

Experimental Workload: Nvidia Gradient Boosting Algorithm Benchmark*

Dataset	Rows	#Features	Task
Fraud	285k	28	BinaryClass
Year	512k	90	Regression
Covtype	581k	54	MultiClass
Epsilon	500k	2000	BinaryClass

3 Models: RandomForest, XGBoost, LightGBM.

80/20 train/test split.

Batch inference (batch size 10k w/ and w/o GPU.

(* https://github.com/NVIDIA/gbm-bench)

Tree-Models Microbenchmark

Algorithm	Dataset	Sklearn (CPU Baseline)	Hummingbird (CPU)		RAPIDS (GPU Baseline)	Hummingbird (GPU)	
			TorchScript	TVM		TorchScript	TVM
Rand. Forest	Fraud						
	Year						
	Covtype						
	Epsilon						
LightGBM	Fraud						
	Year						
	Covtype						
	Epsilon						
XGBoost	Fraud						
	Year						
	Covtype						
	Epsilon						

(All runtimes are reported in seconds. More datasets and experimental results in the paper.)

Tree-Models Microbenchmark

Algorithm	Dataset	Sklearn (CPU Baseline)	Hummingbird (CPU)		RAPIDS (GPU Baseline)	Hummingbird (GPU)	
			TorchScript	TVM		TorchScript	TVM
Rand. Forest	Fraud	**2.5**	7.8	3.0			
	Year	1.9	7.7	**1.4**			
	Covtype	**5.9**	16.5	6.8			
	Epsilon	9.8	13.9	**6.6**			
LightGBM	Fraud	3.4	7.6	**1.7**			
	Year	5.0	7.6	**1.6**			
	Covtype	51.1	79.5	**27.2**			
	Epsilon	10.5	14.5	**4.0**			
XGBoost	Fraud	1.9	7.6	**1.6**			
	Year	3.1	7.6	**1.6**			
	Covtype	42.3	79.0	**26.4**			
	Epsilon	7.6	14.8	**4.2**			

(All runtimes are reported in seconds. More datasets and experimental results in the paper.)

Tree-Models Microbenchmark

Algorithm	Dataset	Sklearn (CPU Baseline)	Hummingbird (CPU)		RAPIDS (GPU Baseline)	Hummingbird (GPU)	
			TorchScript	TVM		TorchScript	TVM
Rand. Forest	Fraud	**2.5**	7.8	3.0			
	Year	1.9	7.7	**1.4**			
	Covtype	**5.9**	16.5	6.8			
	Epsilon	9.8	13.9	**6.6**			
LightGBM	Fraud	3.4	7.6	**1.7**			
	Year	5.0	7.6	**1.6**			
	Covtype	51.1	79.5	**27.2**			
	Epsilon	10.5	14.5	**4.0**			
XGBoost	Fraud	1.9	7.6	**1.6**			
	Year	3.1	7.6	**1.6**			
	Covtype	42.3	79.0	**26.4**			
	Epsilon	7.6	14.8	**4.2**			

(All runtimes are reported in seconds. More datasets and experimental results in the paper.)

Tree-Models Microbenchmark

Algorithm	Dataset	Sklearn (CPU Baseline)	Hummingbird (CPU)		RAPIDS (GPU Baseline)	Hummingbird (GPU)	
			TorchScript	TVM		TorchScript	TVM
Rand. Forest	Fraud	**2.5**	7.8	3.0			
	Year	1.9	7.7	**1.4**			
	Covtype	**5.9**	16.5	6.8			
	Epsilon	9.8	13.9	**6.6**			
LightGBM	Fraud	3.4	7.6	**1.7**			
	Year	5.0	7.6	**1.6**			
	Covtype	51.1	79.5	**27.2**			
	Epsilon	10.5	14.5	**4.0**			
XGBoost	Fraud	1.9	7.6	**1.6**			
	Year	3.1	7.6	**1.6**			
	Covtype	42.3	79.0	**26.4**			
	Epsilon	7.6	14.8	**4.2**			

(All runtimes are reported in seconds. More datasets and experimental results in the paper.)

Tree-Models Microbenchmark

Algorithm	Dataset	Sklearn (CPU Baseline)	Hummingbird (CPU) TorchScript	Hummingbird (CPU) TVM	RAPIDS (GPU Baseline)	Hummingbird (GPU) TorchScript	Hummingbird (GPU) TVM
Rand. Forest	Fraud	**2.5**	7.8	3.0	!SUPPORTED	0.044	**0.015**
	Year	1.9	7.7	**1.4**	!SUPPORTED	0.045	**0.026**
	Covtype	**5.9**	16.5	6.8	!SUPPORTED	0.110	**0.047**
	Epsilon	9.8	13.9	**6.6**	!SUPPORTED	**0.130**	**0.13**
LightGBM	Fraud	3.4	7.6	**1.7**	**0.014**	0.044	**0.014**
	Year	5.0	7.6	**1.6**	**0.023**	0.045	0.025
	Covtype	51.1	79.5	**27.2**	!SUPPORTED	0.620	**0.250**
	Epsilon	10.5	14.5	**4.0**	0.150	0.130	**0.120**
XGBoost	Fraud	1.9	7.6	**1.6**	**0.013**	0.044	0.015
	Year	3.1	7.6	**1.6**	**0.022**	0.045	0.026
	Covtype	42.3	79.0	**26.4**	!SUPPORTED	0.620	**0.250**
	Epsilon	7.6	14.8	**4.2**	0.150	0.130	**0.120**

(All runtimes are reported in seconds. More datasets and experimental results in the paper.)

Tree-Models Microbenchmark

Algorithm	Dataset	Sklearn (CPU Baseline)	Hummingbird (CPU)		RAPIDS (GPU Baseline)	Hummingbird (GPU)	
			TorchScript	TVM		TorchScript	TVM
Rand. Forest	Fraud	**2.5**	7.8	3.0	!SUPPORTED	0.044	**0.015**
	Year	1.9	7.7	**1.4**	!SUPPORTED	0.045	**0.026**
	Covtype	**5.9**	16.5	6.8	!SUPPORTED	0.110	**0.047**
	Epsilon	9.8	13.9	**6.6**	!SUPPORTED	**0.130**	**0.13**
LightGBM	Fraud	3.4	7.6	**1.7**	**0.014**	0.044	**0.014**
	Year	5.0	7.6	**1.6**	**0.023**	0.045	0.025
	Covtype	51.1	79.5	**27.2**	!SUPPORTED	0.620	**0.250**
	Epsilon	10.5	14.5	**4.0**	0.150	0.130	**0.120**
XGBoost	Fraud	1.9	7.6	**1.6**	**0.013**	0.044	0.015
	Year	3.1	7.6	**1.6**	**0.022**	0.045	0.026
	Covtype	42.3	79.0	**26.4**	!SUPPORTED	0.620	**0.250**
	Epsilon	7.6	14.8	**4.2**	0.150	0.130	**0.120**

(All runtimes are reported in seconds. More datasets and experimental results in the paper.)

Summary

Hummingbird: A Tensor Compiler for Unified Machine Learning Prediction Serving.

Summary

Hummingbird: A Tensor Compiler for Unified Machine Learning Prediction Serving.

Compiles traditional ML pipelines into tensor computations and thereby reuse DL prediction serving systems for traditional ML prediction serving.

Summary

Hummingbird: A Tensor Compiler for Unified Machine Learning Prediction Serving.

Compiles traditional ML pipelines into tensor computations and thereby reuse DL prediction serving systems for traditional ML prediction serving.

Open sourced and part of PyTorch eco-system and ONNX converters. Support for more traditional ML training frameworks and target DL prediction serving runtimes is in the pipeline.

Summary

Hummingbird: A Tensor Compiler for Unified Machine Learning Prediction Serving.

Compiles traditional ML pipelines into tensor computations and thereby reuse DL prediction serving systems for traditional ML prediction serving.

Open sourced and part of PyTorch eco-system and ONNX converters. Support for more traditional ML training frameworks and target DL prediction serving runtimes is in the pipeline.

Thank You!

https://github.com/microsoft/hummingbird

hummingbird-dev@microsoft.com

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED