An Automated End-to-End Optimizing Compiler for Deep Learning

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan,Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, Arvind Krishnamurthy

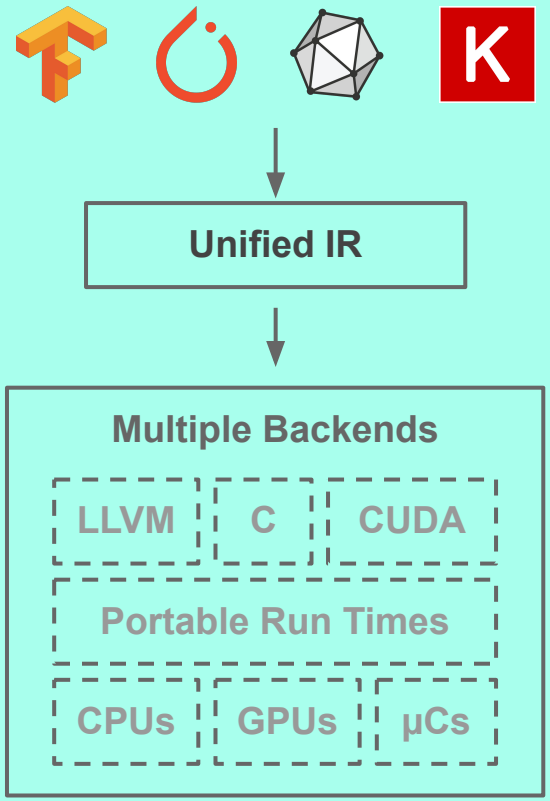
Group 14

Zhenning Yang, Reggie Hsu, Leo Wu, and Dave Yonkers

What Is TVM?

End-to-end framework for neural net compilation

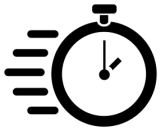
- Imports trained models from ***popular deep-learning frameworks*** (PyTorch, Tensorflow, ONNX, Keras, etc.)
- **Unifies** *imported models* into **custom IR**
- Applies traditional optimization techniques
- **Tunes the model** for the ***target hardware***
- **Compiles linkable output modules** ready for production deployment (C++, Java, Python, etc.)



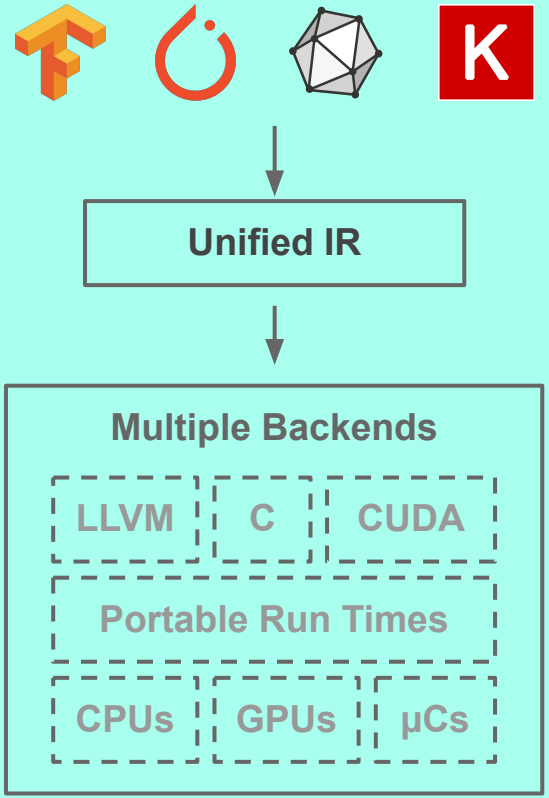
Why use TVM?

Common operations, such as **matrix multiplication**, can be sped up over

18x



Very flexible and nearly platform independent



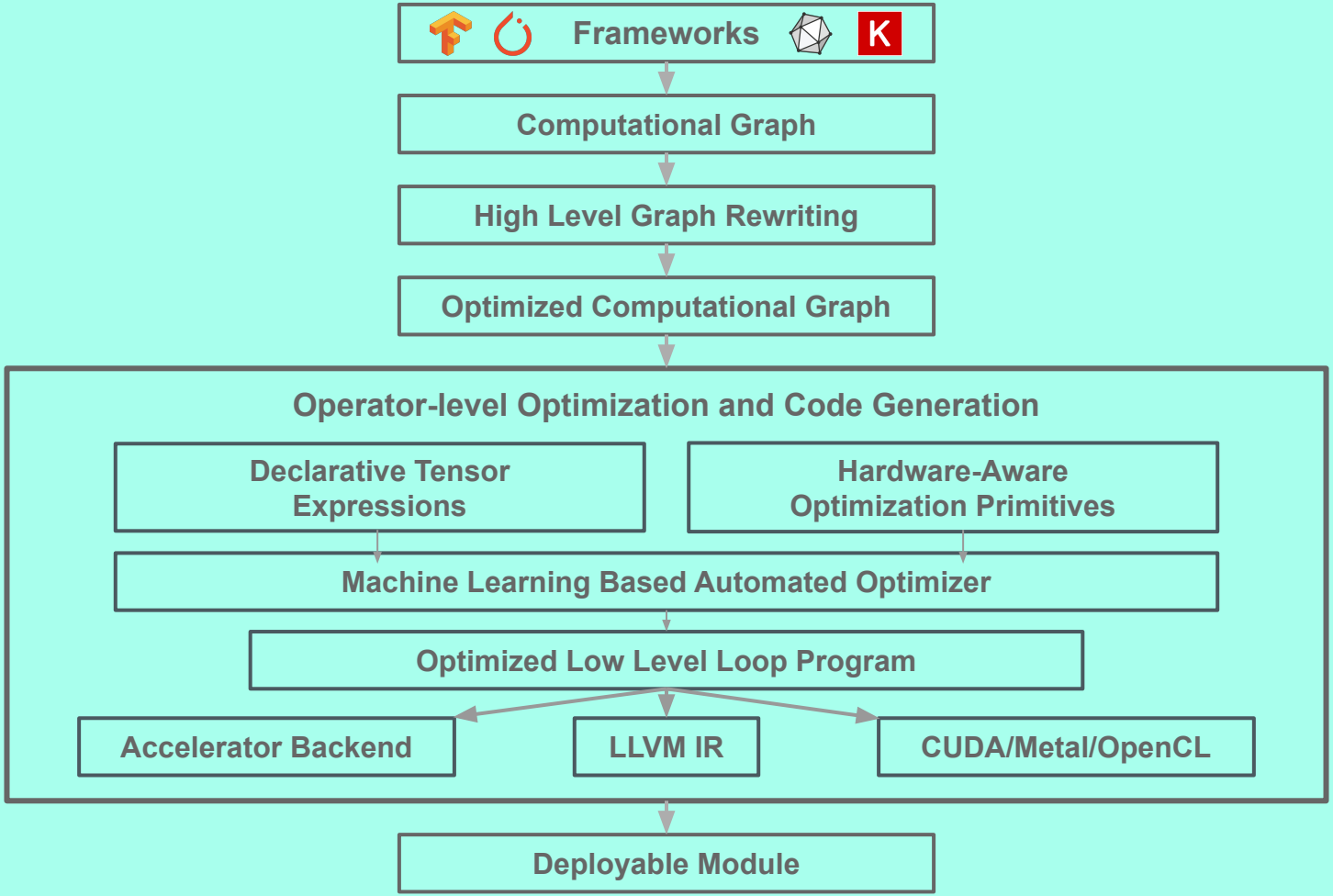
Previously

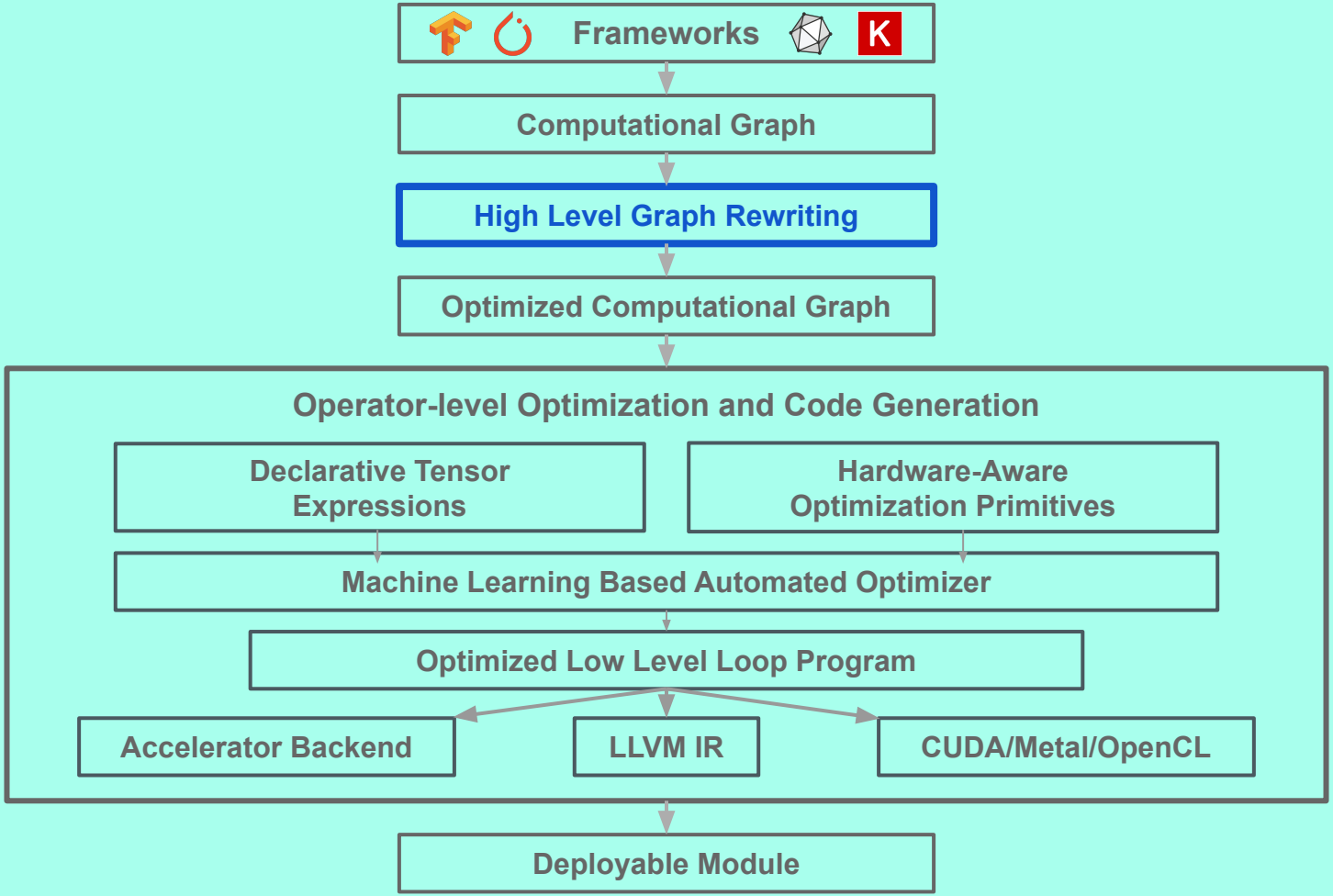
DL frameworks implement graph-level optimizations

- High level
- Not sufficient enough to take advantage of hardware architecture

Hand-tuning was necessary for low-level optimizations

- Expensive
- Time consuming
- Not portable → portability necessary for modern, vast hardware landscape

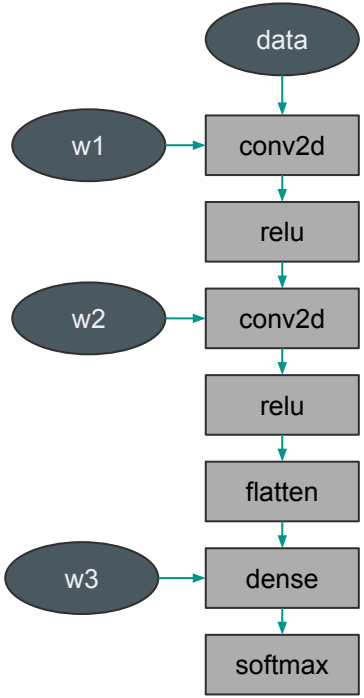

Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module


Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

Optimizing Computational Graph

Computational Graph (CG) vs LLVM IR

- Intermediate data are tensors
- Does not specify how each operator is implemented



Optimizing Computational Graph

CG is used to apply highest-level optimizations

- **Operator fusion**
- **Constant folding**
  - pre-computes graph parts *statically*
- **Static memory planning pass**
- **Data layout transformations**
  - Rearrange data to exploit features like *vectorization for GPUs* or even 4x4 matrix operations for a DL accelerator

Operator Fusion

Injective

Addition

Reduction

Summation

Complex-out-fusable

Cov2d

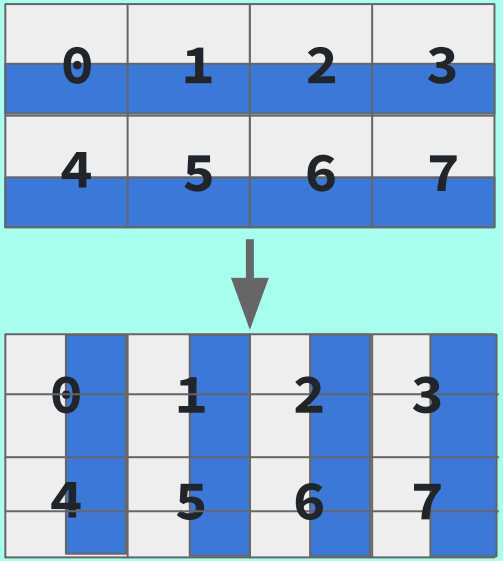
Opaque

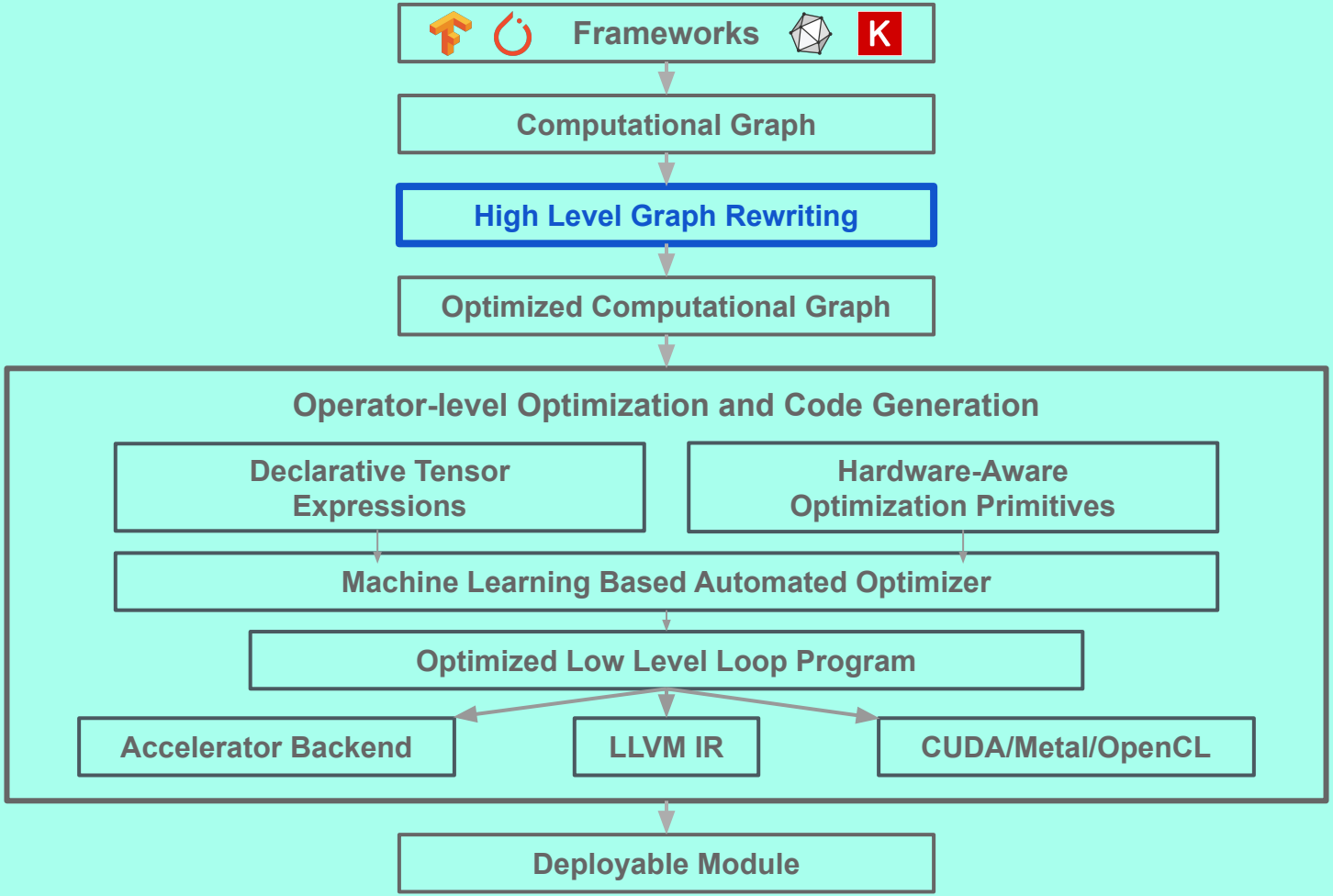
Sort

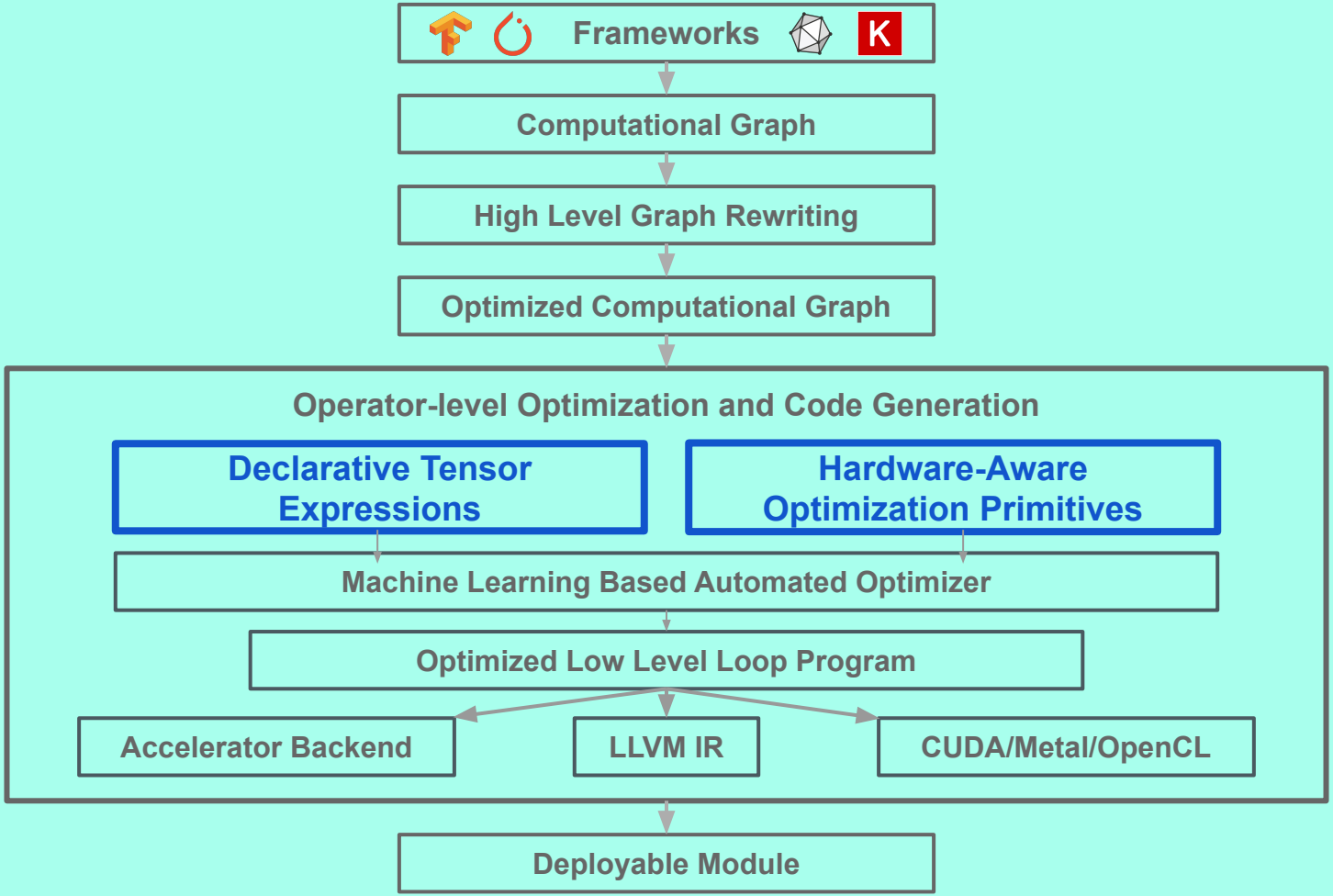
Data Layout Transformation

- Converts a CG into to use ***better internal data layouts***.
- Specify the **preferred data layout** for each operator given their *memory hierarchy constraints*.
- Perform the proper layout transformation between a producer and a consumer if their **preferred data layouts** ***do not match***.

DL accelerators might exploit *n×n matrix operations*, requiring data to be tiled into *n×n chunks* to optimize for access locality.




Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module


Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

Generating Tensor Operations

Tensor Expression and Schedule Space

- Each ***tensor operation*** is described in TVM's own **expression language**, and each operation specifies:
  - The **shape** of the output tensor
  - How to compute ***each individual element*** of the output tensor
- Use schedules to denote a specific mapping from a tensor expression to ***low-level code***

Tensor expression of *MxK* and *KxN* matrix multiplication:

```
k = te.reduce_axis((0, K), "k")
A = te.placeholder((M, K), name="A")
B = te.placeholder((K, N), name="B")
C = te.compute((M, N), lambda x, y:
    te.sum(A[x, k] * B[k, y],
    axis=k), name="C")
```

Generating Tensor Operations

Nested Parallelism

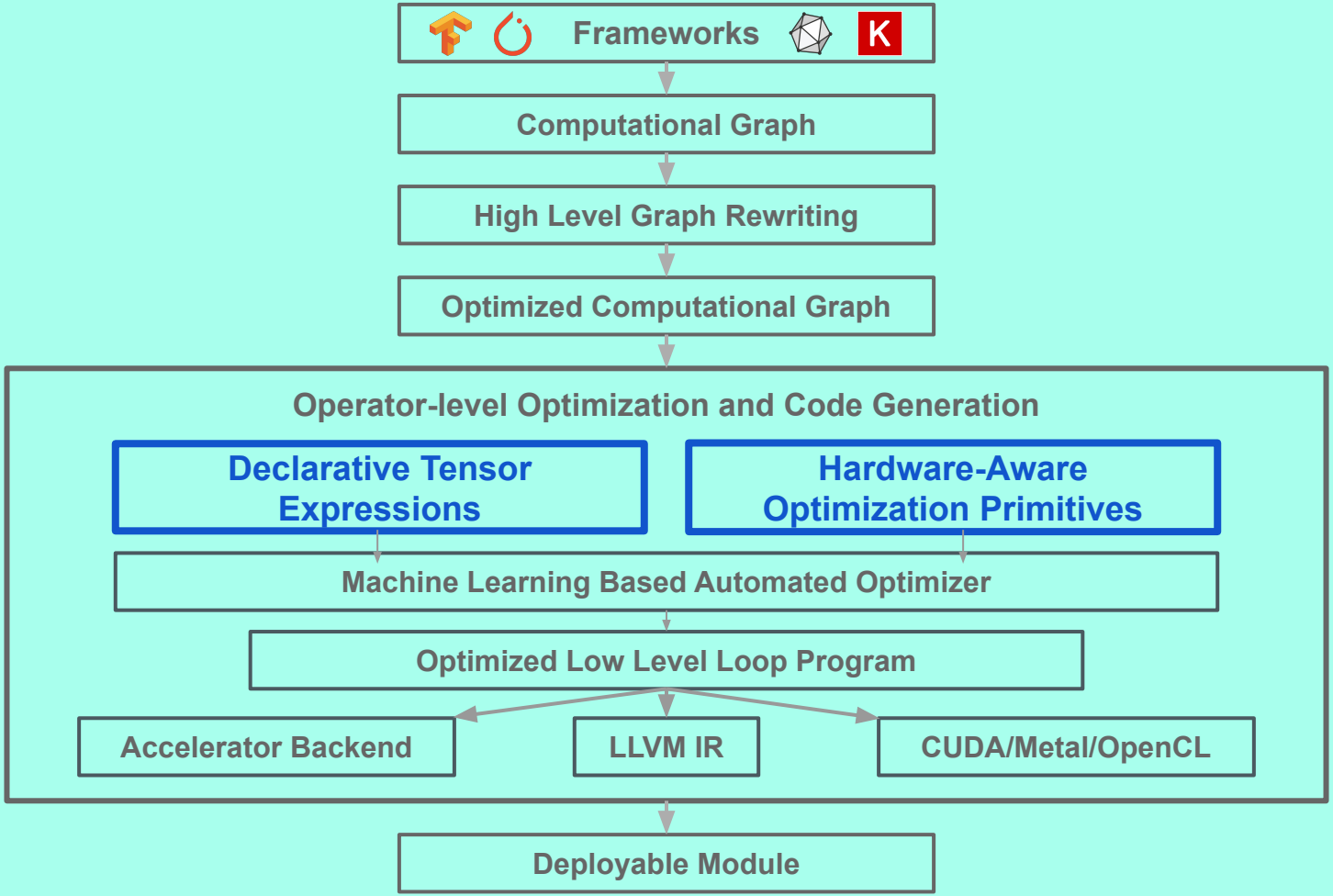
- Each task is ***split into subtasks***
- Subtasks are grouped to ***exploit hardware architecture*** (e.g. **GPU thread groups**)

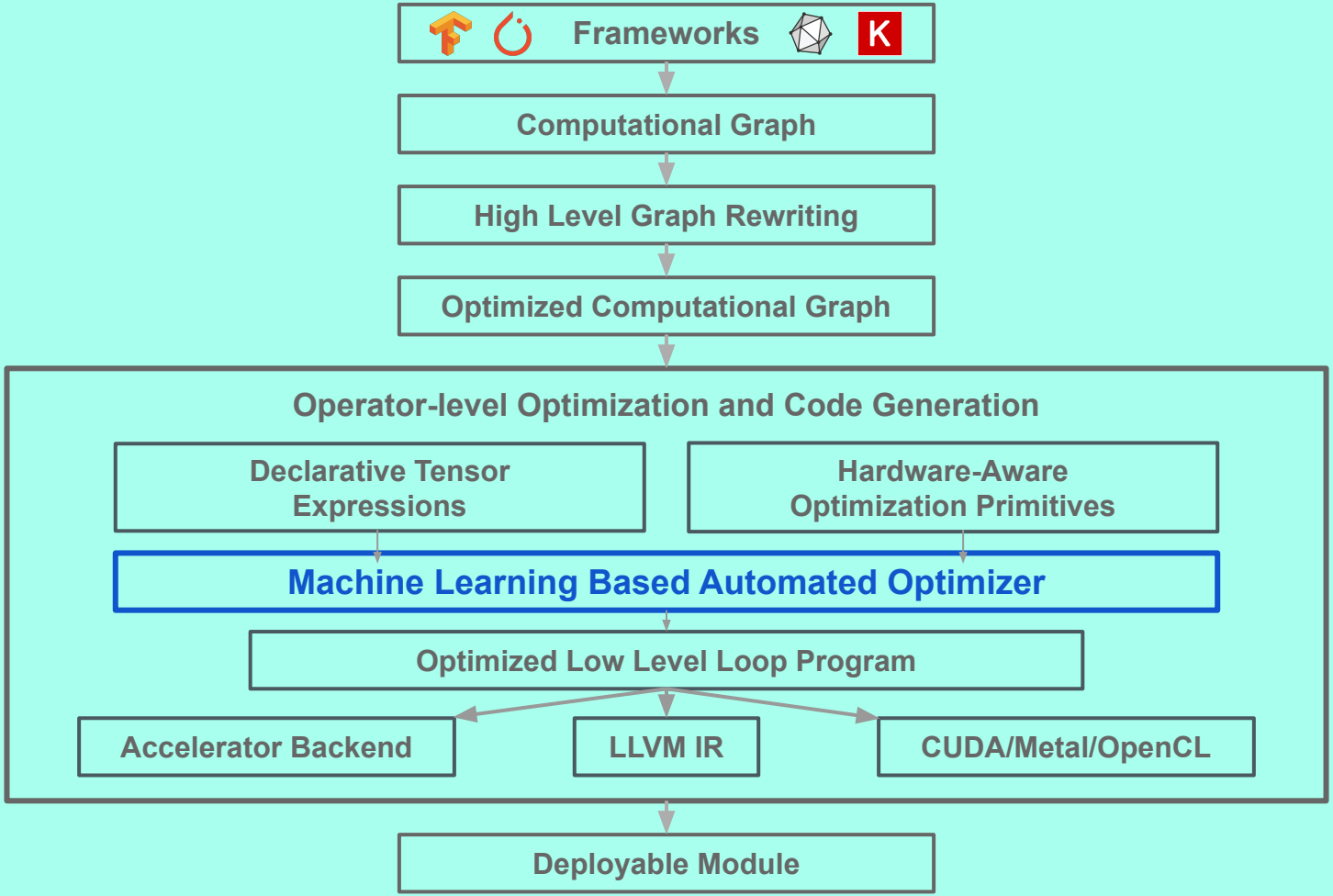
Tensorization

- Decomposes DL workloads into **common tensor operators**
- Allows for hardware compilation to take advantage of *emerging tensor compute primitives*

Explicit Memory Latency Hiding

- Overlapping **memory access** with **computation** to ***maximize utilization***
- Uses *hardware prefetching* and *simultaneous multithreading* on **CPUs**
- Uses *rapid context switching* on **GPUs**
- Uses *DAE* on **TPUs**


Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module


Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

Automating Optimization

Lowering tensor IR operations into machine code presents a number of different factors to consider:

- **Grouping Instructions**
- **Optimizing for Memory Hierarchy**
- **Tiling Size**
- **Modifying Loop Order**
- **Loop Unrolling Factor**

Search space for large models can be *in the billions*

Automating Optimization

Previous

Approach

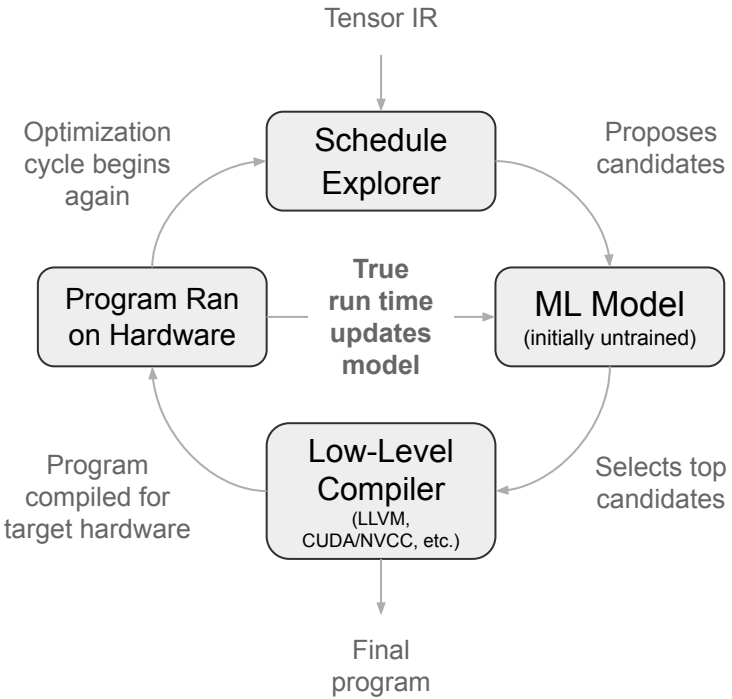
- Random search
- Blackbox auto-tuning
- Predefined cost model

TVM's

Approach

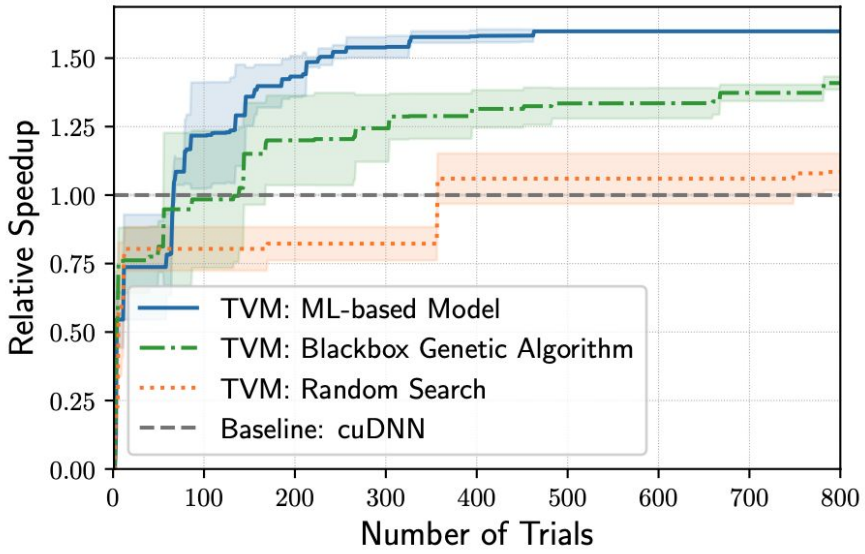
- A ***schedule explorer*** tries out different optimization factors and update an ***ML-based cost model*** to iteratively improve computational efficiency

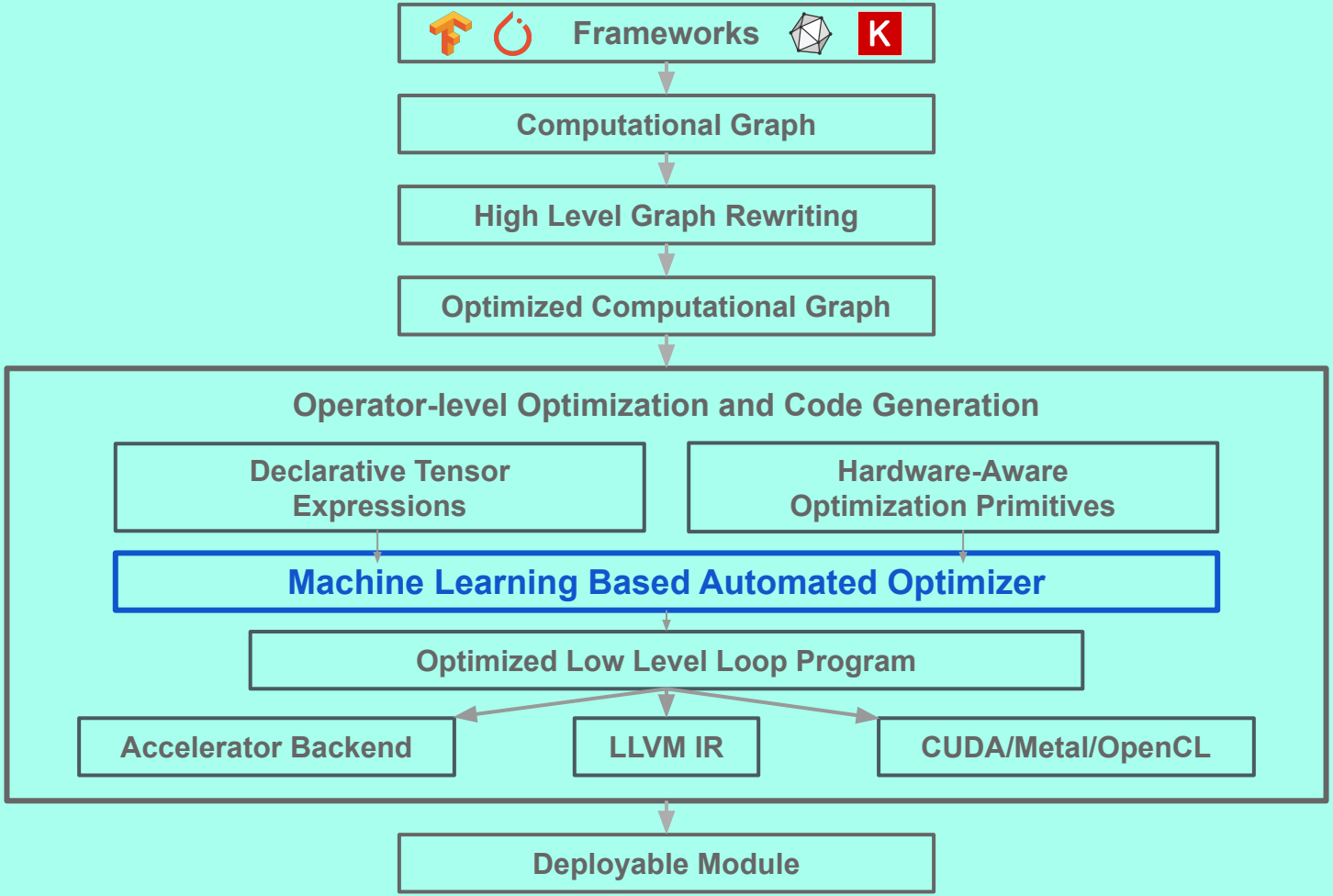
Automating Optimization

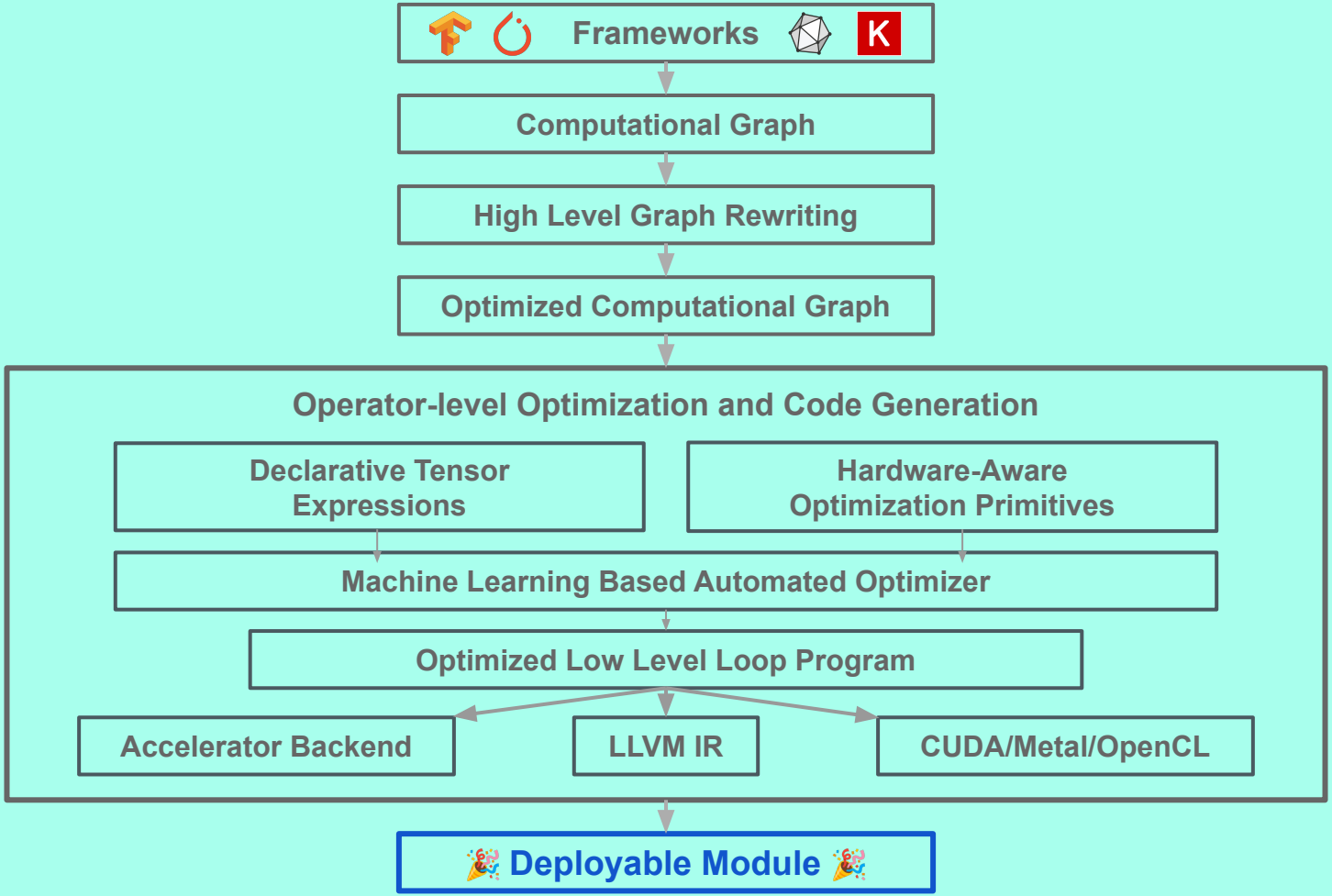


ML Model	Exploration Algorithm
A gradient tree boosting model (based on XGBoost)	Parallel simulated annealing algorithm

Automating Optimization



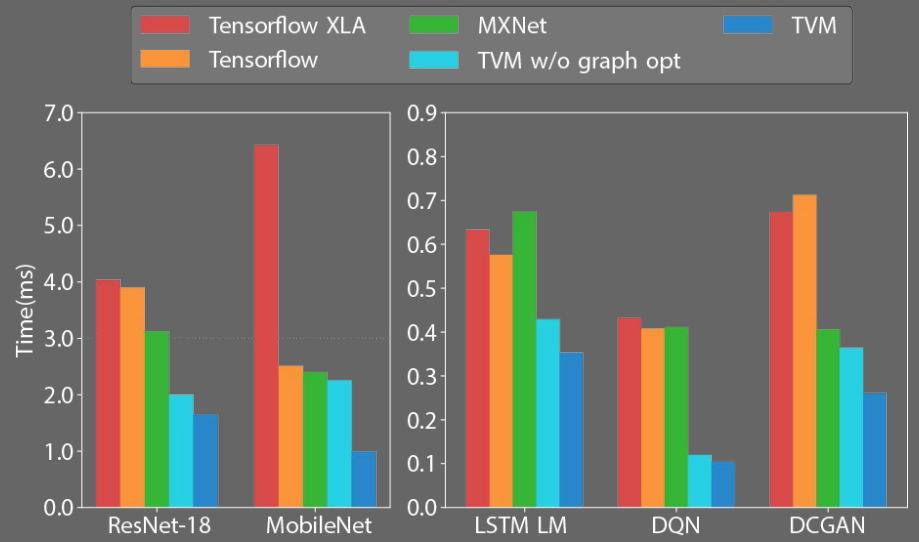

Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module


Frameworks
K
Computational Graph
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
🎉 Deployable Module 🎉

Evaluation: Server Class GPU

- ***Tensorflow*** and ***MXNet*** use standard operator implementations
  - cuDNN v7 for convolutional operators
  - Custom versions for *depthwise convolution*
- TVM *outperforms* standard operators
  - Speedups range from ***1.6x to 3.8x***
  - This is because TVM automatically **finds optimized operators**

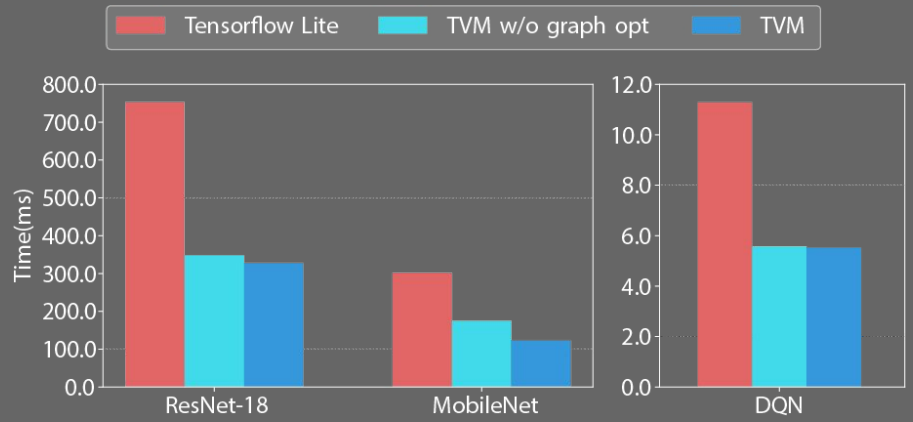
GPU end-to-end evaluation for *TVM*, *MXNet*, *Tensorflow*, and *Tensorflow XLA*. Tested on the NVIDIA Titan X.



Evaluation: Embedded CPU

- TVM *outperforms* standard hand-optimized operators provided by Tensorflow
- Can make workloads more **CPU-friendly**
  - Avoid expensive GPUs

ARM cortex A53 (quad-core 1.2Ghz) end-to-end evaluation of *TVM* and *TFLite*.



Our Thoughts about TVM

Pros

- Fully open-source and *active!*
  - https://github.com/apache/tvm
  - Last commit was only ***hours ago!***
  - TVM 9.3k GitHub stars
  - Accepted neural network frameworks and target platforms are constantly expanding via community contributions
- Extremely flexible
  - Due to the way Relay IR operations are flexibly constructed, optimizations are possible for **any target platform**

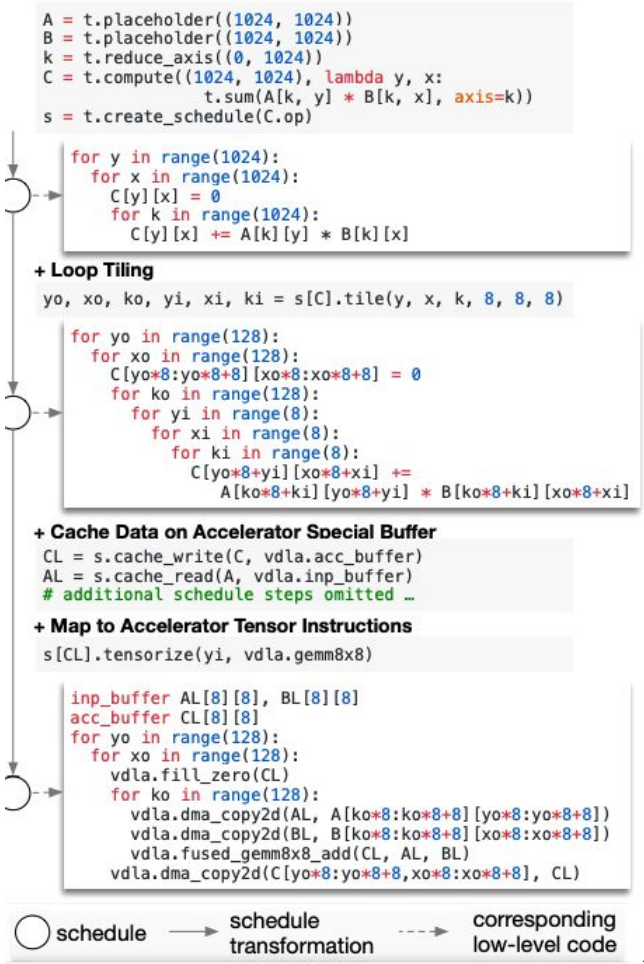
Cons

- Only optimizes for **time**
  - What about energy? Disk space?
  - GPT-4 is ~1 trillion params, other optimizations will need to be considered before any AGI-targeted model is useable
- Optimizations start with an untrained model
  - Production deployments require significant training time
    - CPU: 1,500 iterations
    - GPU: 3,000-4,000 iterations
  - Improvements can usually be seen within ~10 iterations for small models

Q & A

Generating Tensor Operations

- Tensor Expression and Schedule Space
- Each tensor operation is described in TVM's own expression language, and each operation specifies:
  - The shape of the output tensor
  - How to compute each individual element of the output tensor
- Use schedules to denote a specific mapping from a tensor expression to low-level code:
  - Build a schedule by incrementally applying basic transformations (schedule primitives) that preserve the program's logical equivalence



Automating Optimization