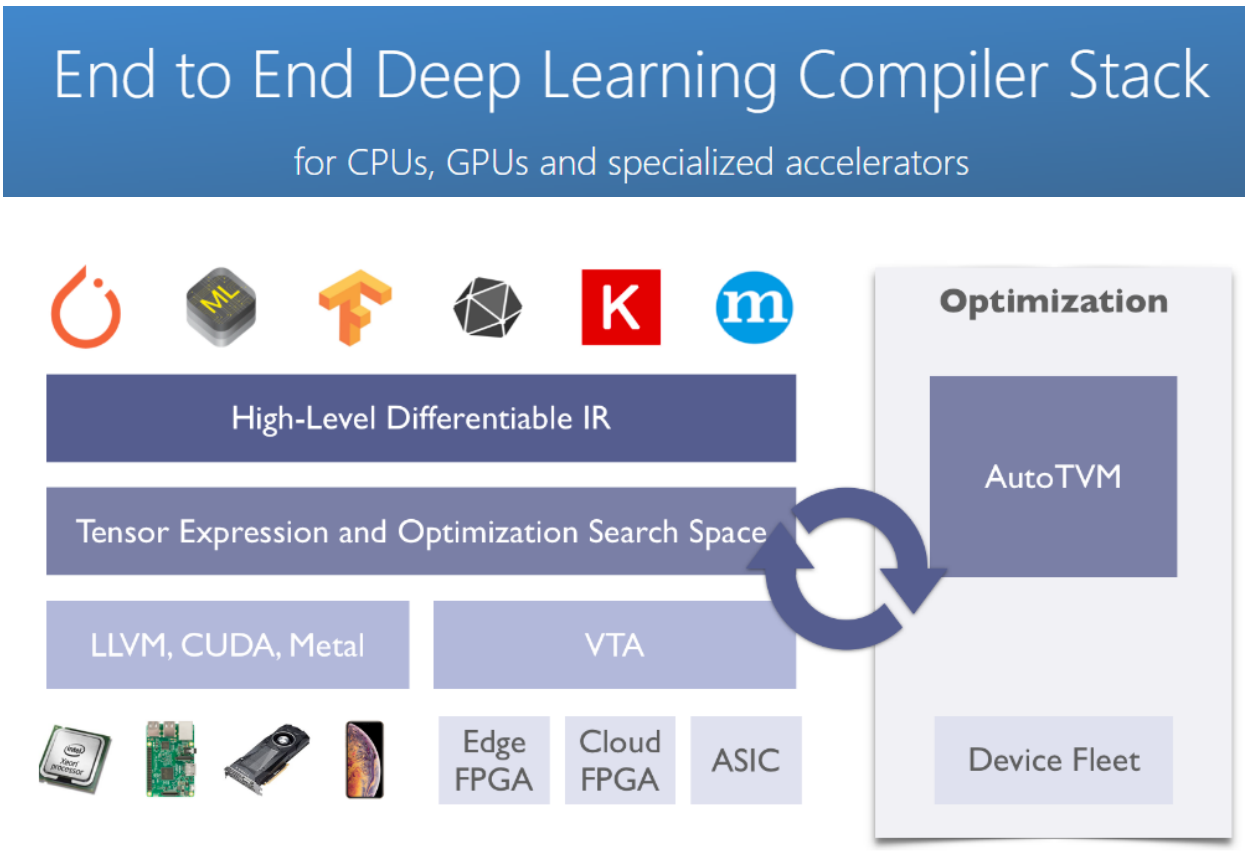
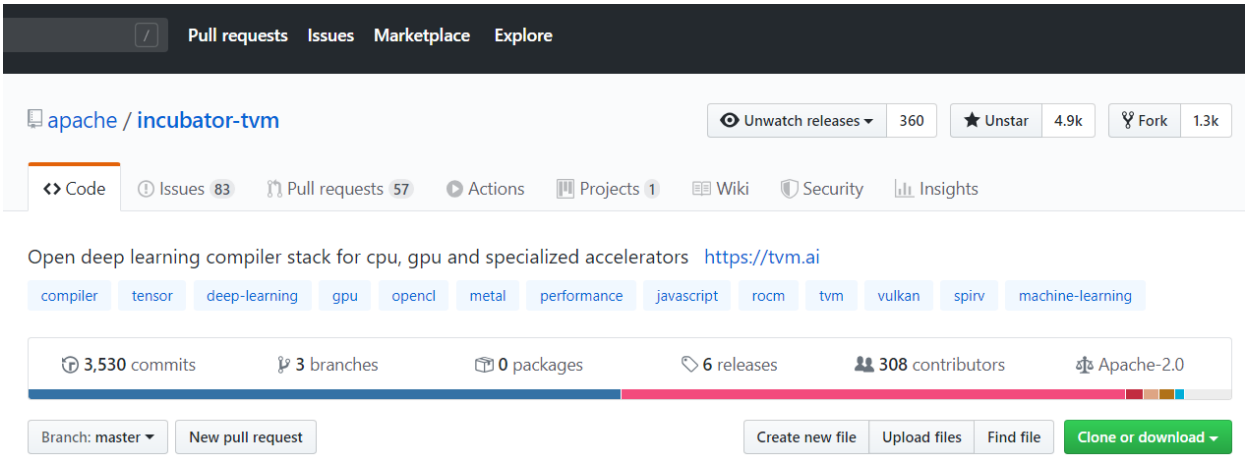


Apache TVM (incubating) is a compiler stack for deep learning systems. It is designed to close the gap between the productivity-focused deep learning frameworks, and the performance- and efficiency-focused hardware backends. TVM works with deep learning frameworks to provide end to end compilation to different backends.



[HTML] TVM: end-to-end optimization stack for deep learning

T Chen, T Moreau, Z Jiang, H Shen, E Yan… - arXiv preprint arXiv …, 2018 - arxiv-vanity.com

Scalable frameworks, such as TensorFlow, MXNet, Caffe, and PyTorch drive the current popularity and utility of deep learning. However, these frameworks are optimized for a narrow range of server-class GPUs and deploying workloads to other platforms such as …

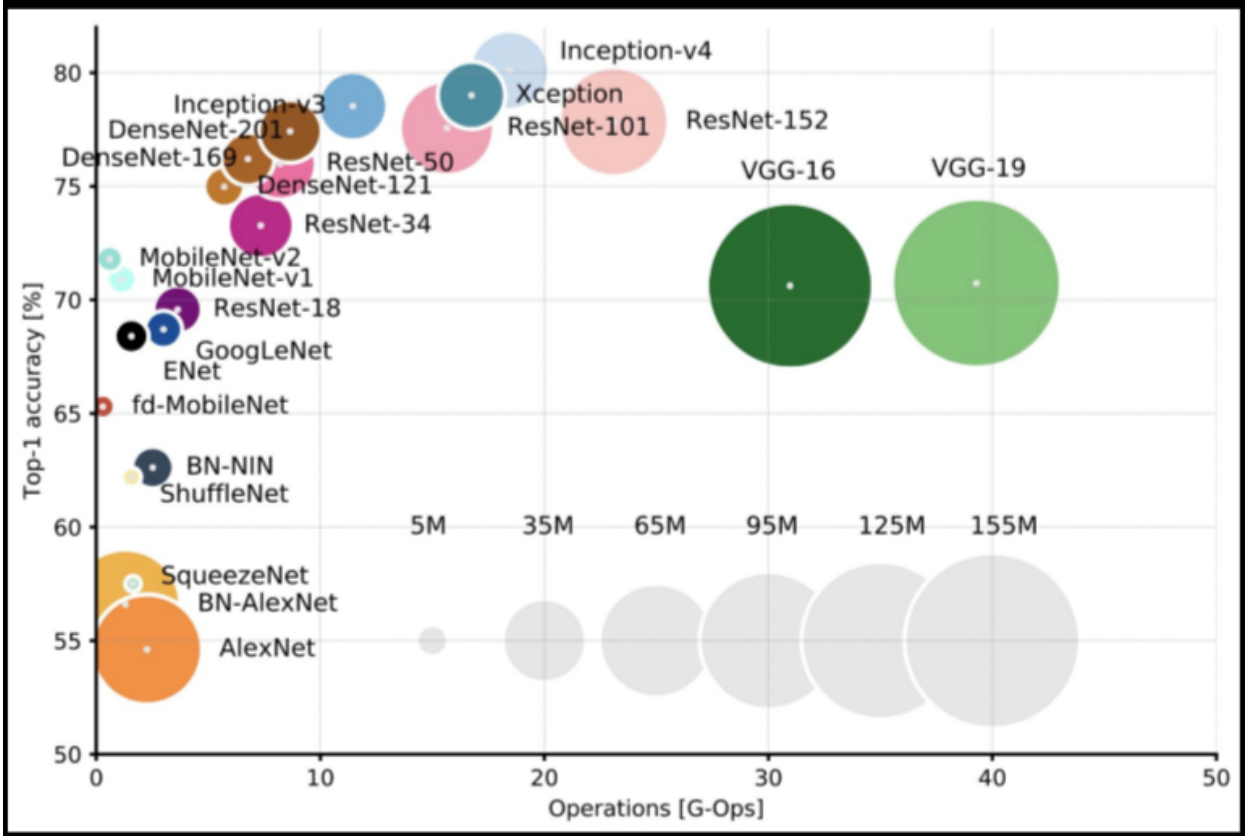
☆ 99 被引用次数：82 相关文章 所有 6 个版本 图书馆搜索 »

Learning to optimize tensor programs
T Chen, L Zheng, E Yan, Z Jiang, T Moreau… - Advances in Neural …, 2018 - papers.nips.cc
Paper accepted and presented at the Neural Information
Processing Systems Conference (http://nips.cc/).
☆ 99 被引用次数：30 相关文章 所有 10 个版本 图书馆搜索 »

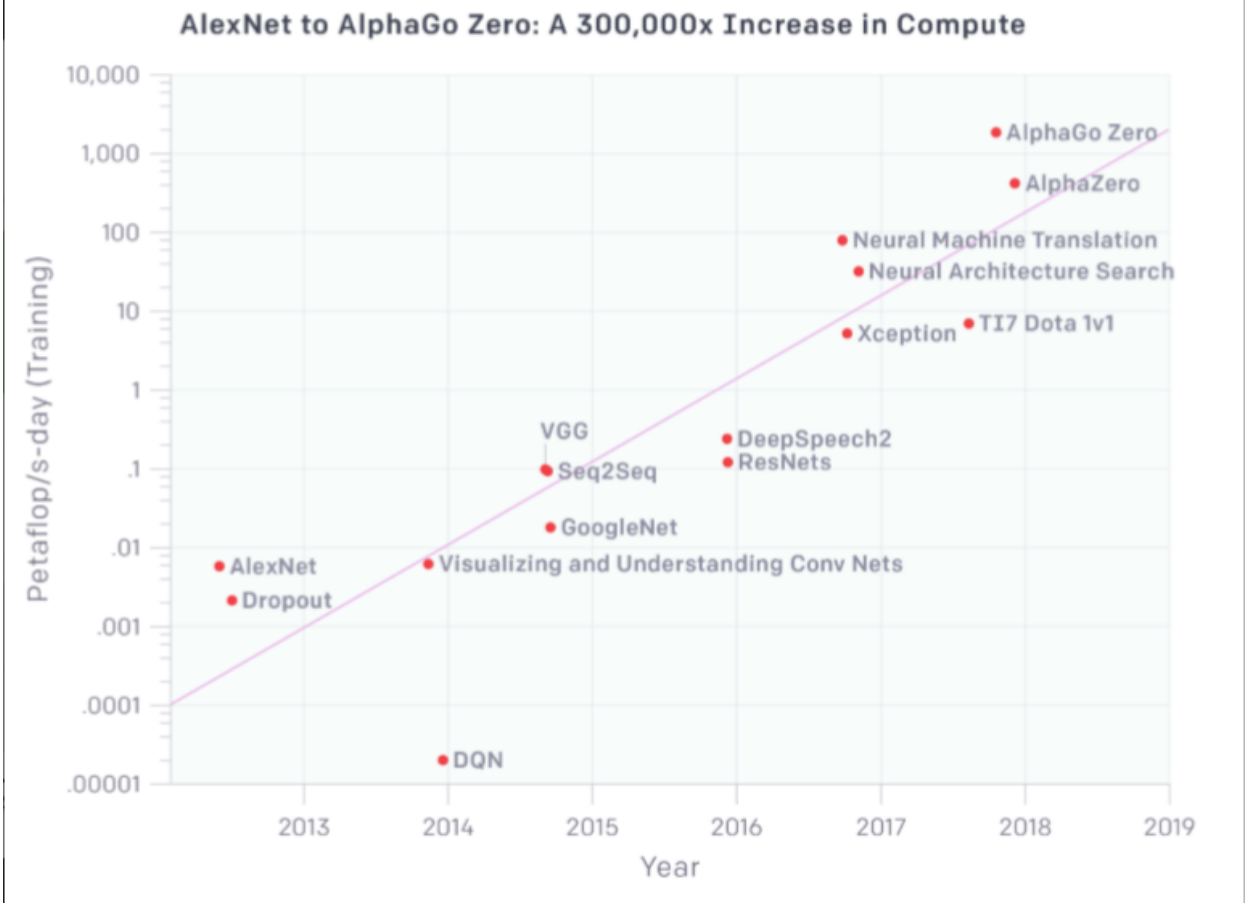
We learned a lot from the following projects when building TVM.

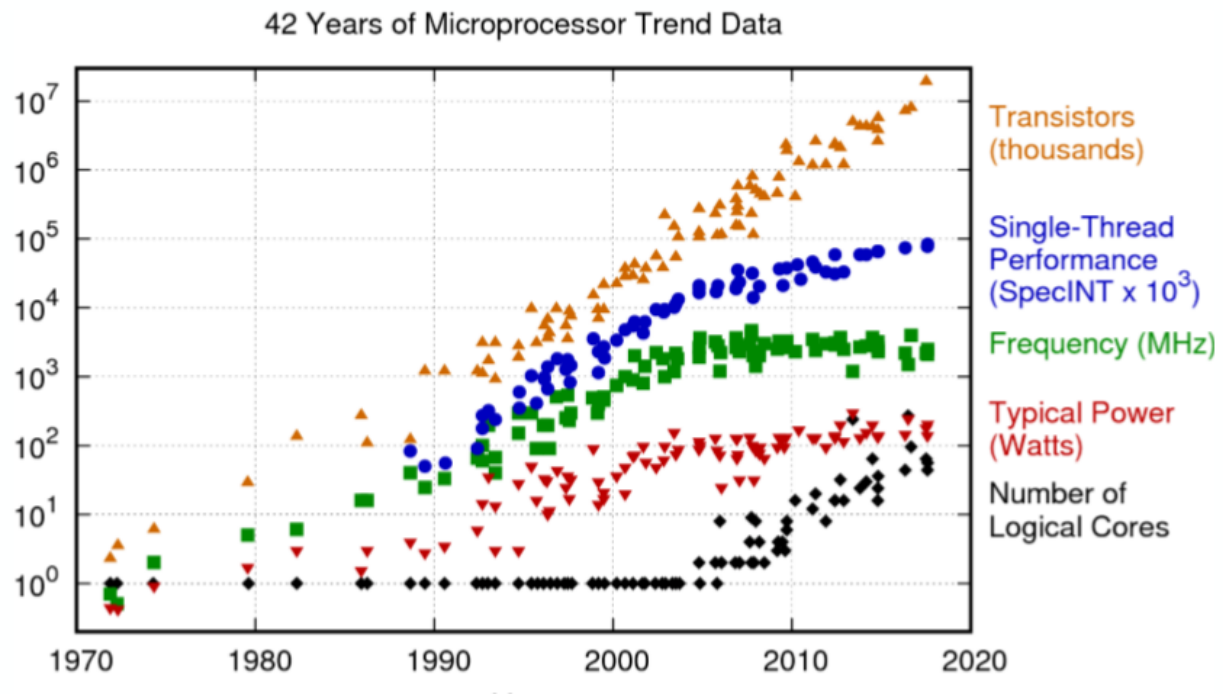
- Halide: TVM uses HalideIR as data structure for arithmetic simplification and low level lowering. We also learned and adapted some part of lowering pipeline from Halide.
- Loopy: use of integer set analysis and its loop transformation primitives.
- Theano: the design inspiration of symbolic scan operator for recurrence.

Model size and compute cost growing fast



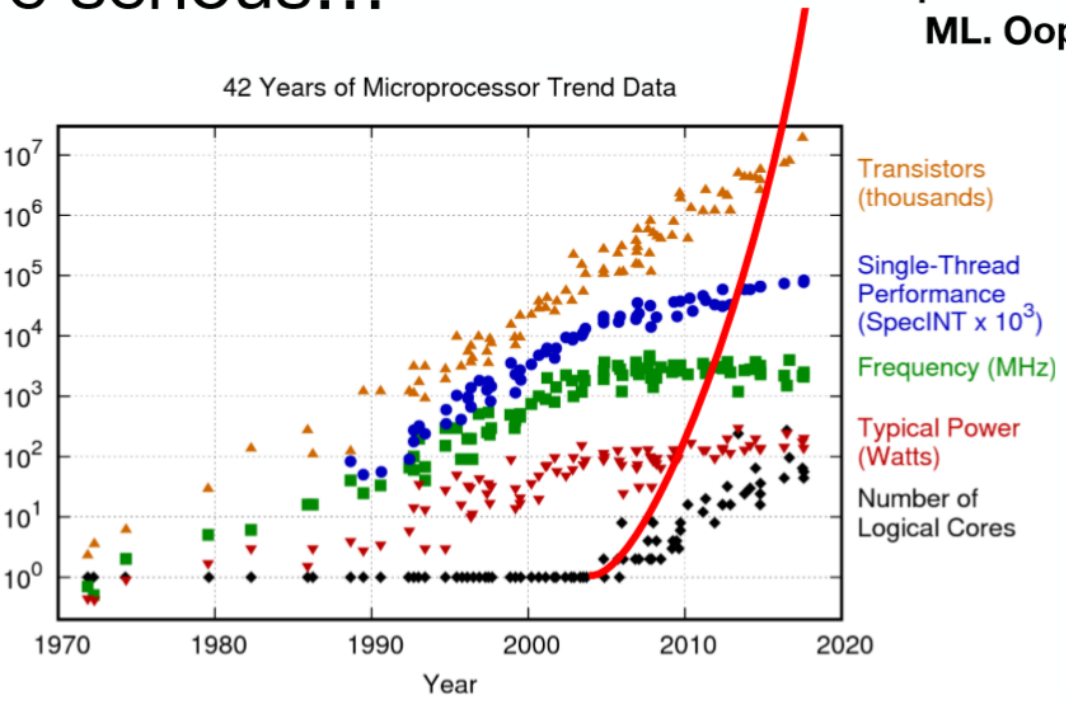
Training costs growing exponentially





’e serious…

Computational cost of ML. Oops. :)



A perfect storm

Growing set of requirements: **cost, latency, power, security & privacy**

Cambrian explosion of models, workloads, and use cases.

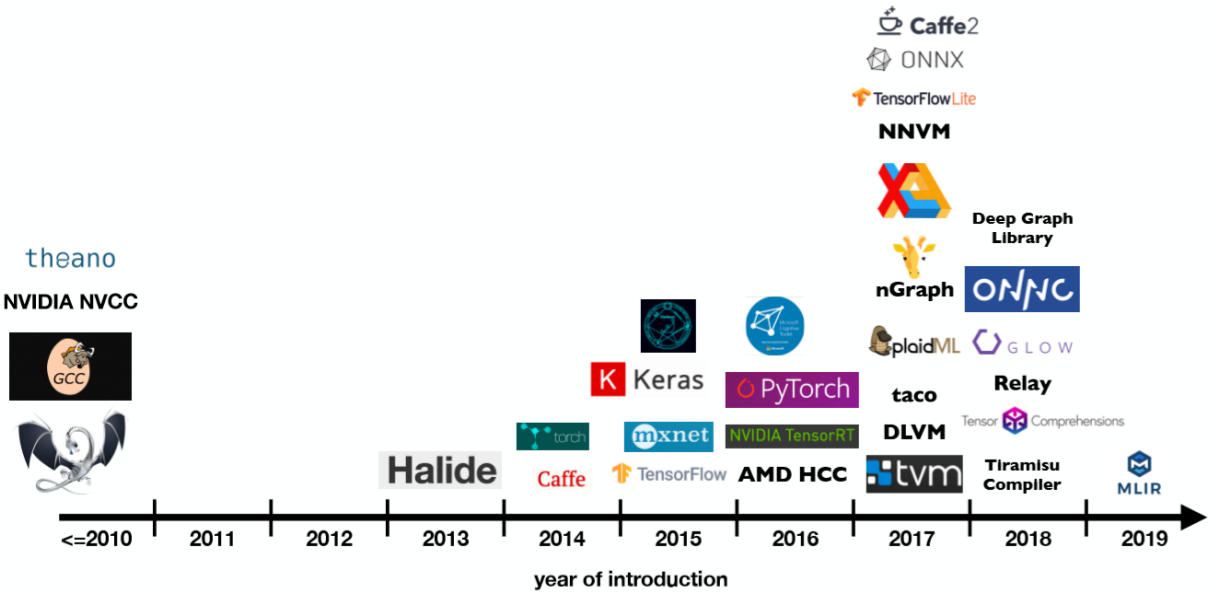
CNN GAN RNN MLP DQNN

Rapidly evolving ML software ecosystem quickly fragmenting

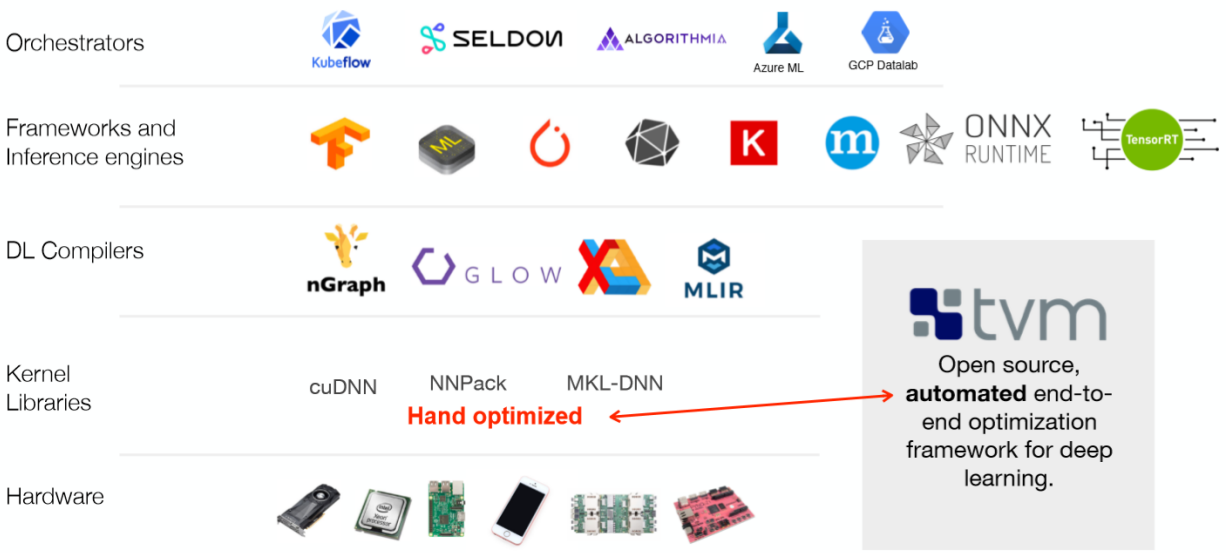
Silicon scaling limitations (Dennard and Moore):

Cambrian explosion of HW backends. Heterogeneous HW.

Deep learning “stack” (r?)evolution



Current Dominant Deep Learning Systems Landscape



Using ML for better ML systems..

Deal with design complexity and large parameter spaces…

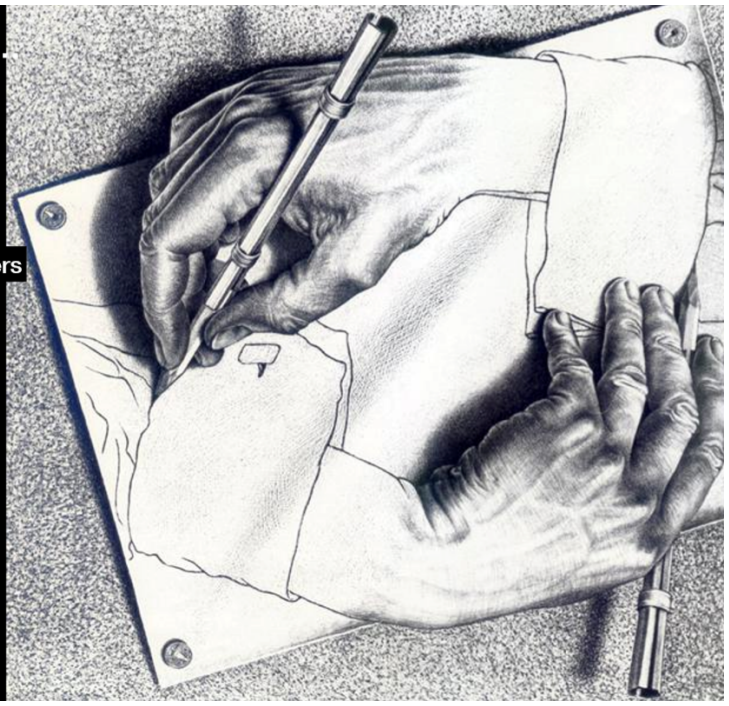
Model optimization strategies and parameters

Efficient operator implementations

Data communication patterns

Model-HW co-tuning

Searching for efficient HW designs



tvm Open Source Community Growth and Impact

70% growth from Dec 2018 to **295 contributors** from UW, Berkeley, Cornell, UCLA, Amazon, Huawei, NTT, Facebook, Microsoft, Qualcomm, Alibaba, Intel, …

Used in production at leading vendors:

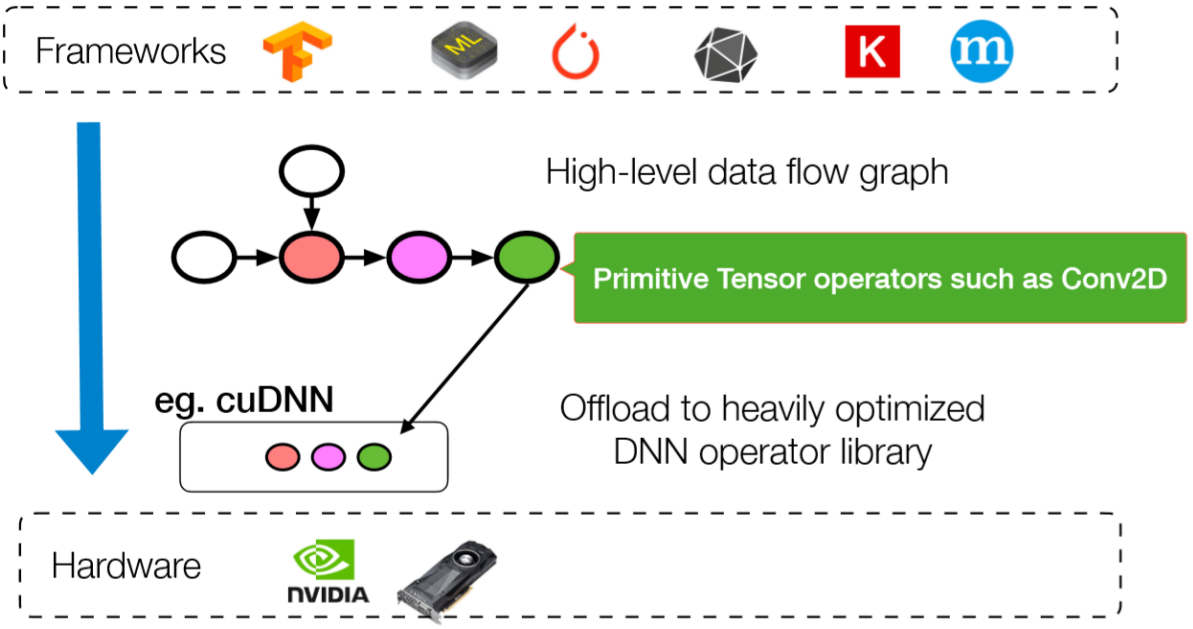
aws
Deep Learning Compiler Service

HUAWEI
Tensor Engine for mobile ASIC

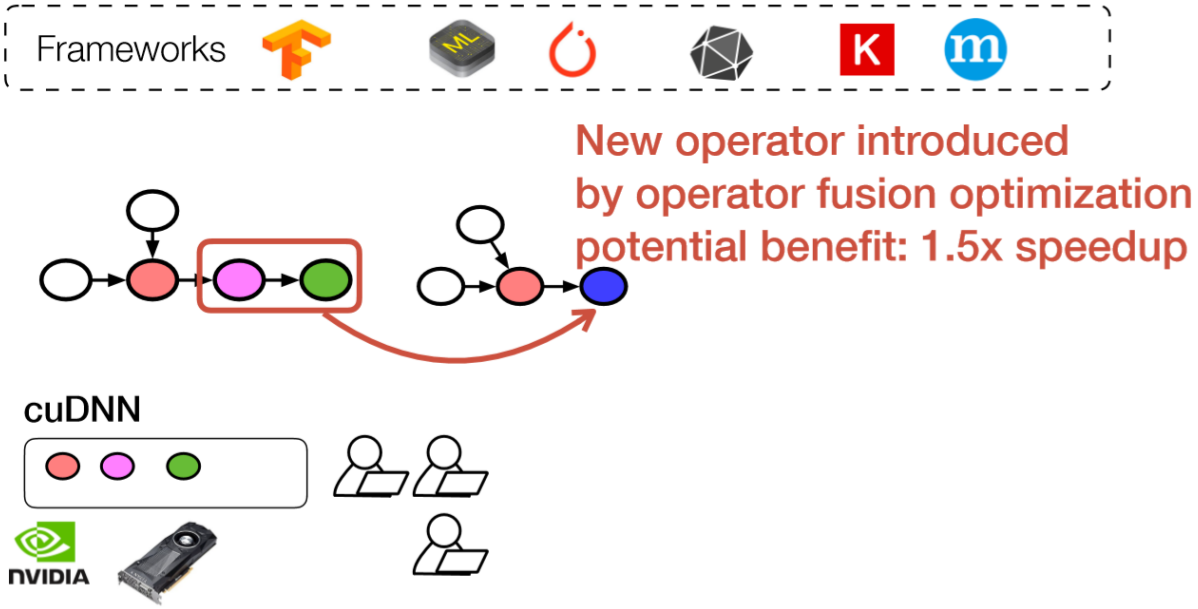
Mobile and Server Optimizations

Microsoft
Cloud-side model optimization

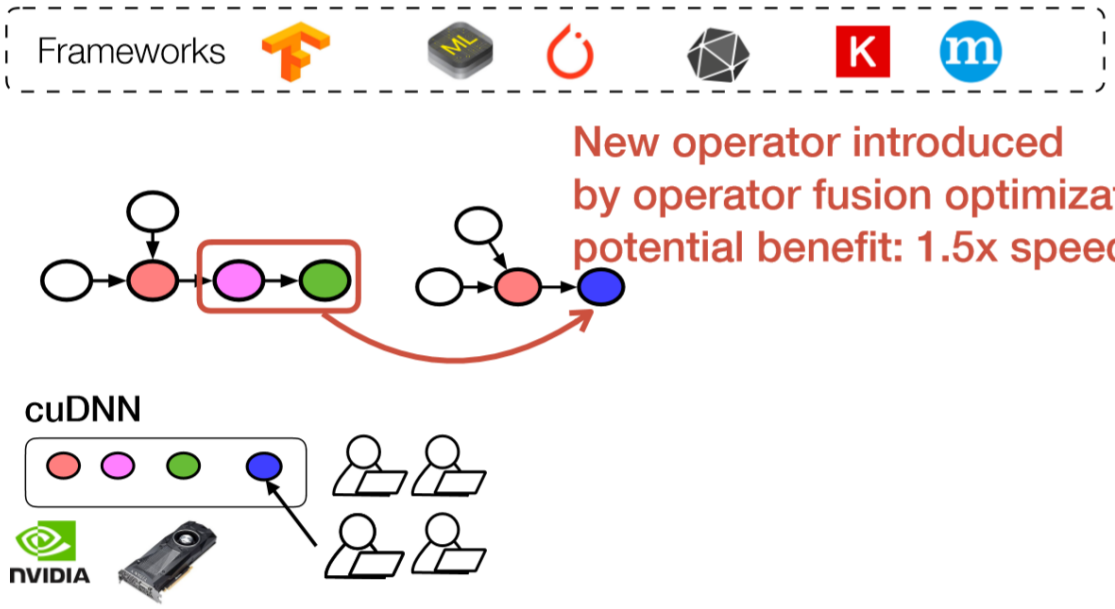
Existing Deep Learning Frameworks



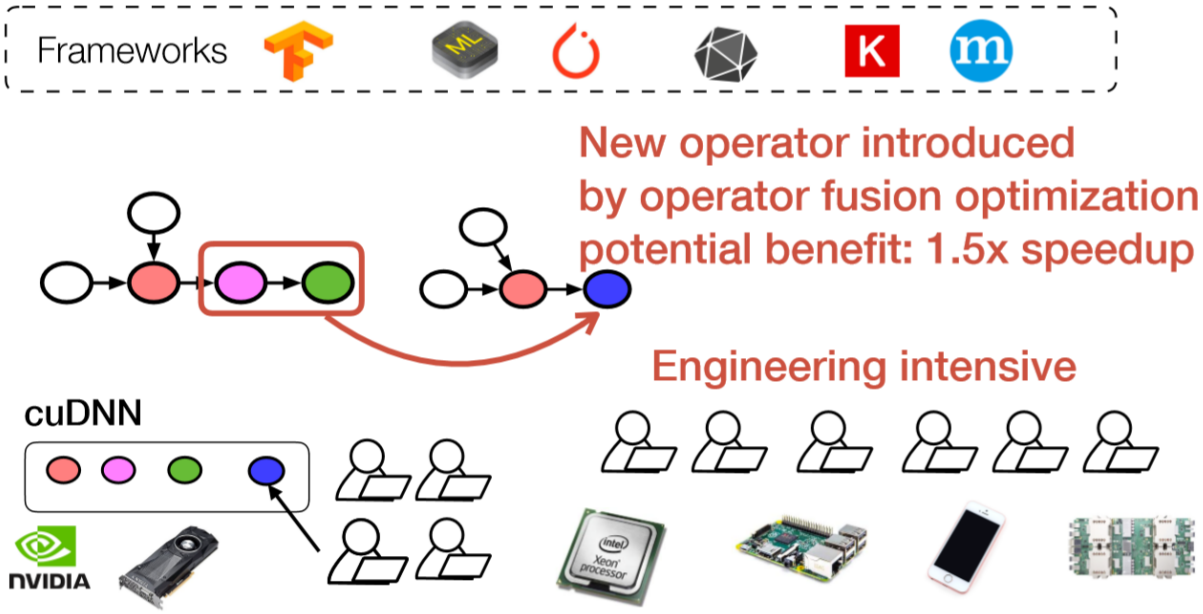
Limitations of Existing Approach



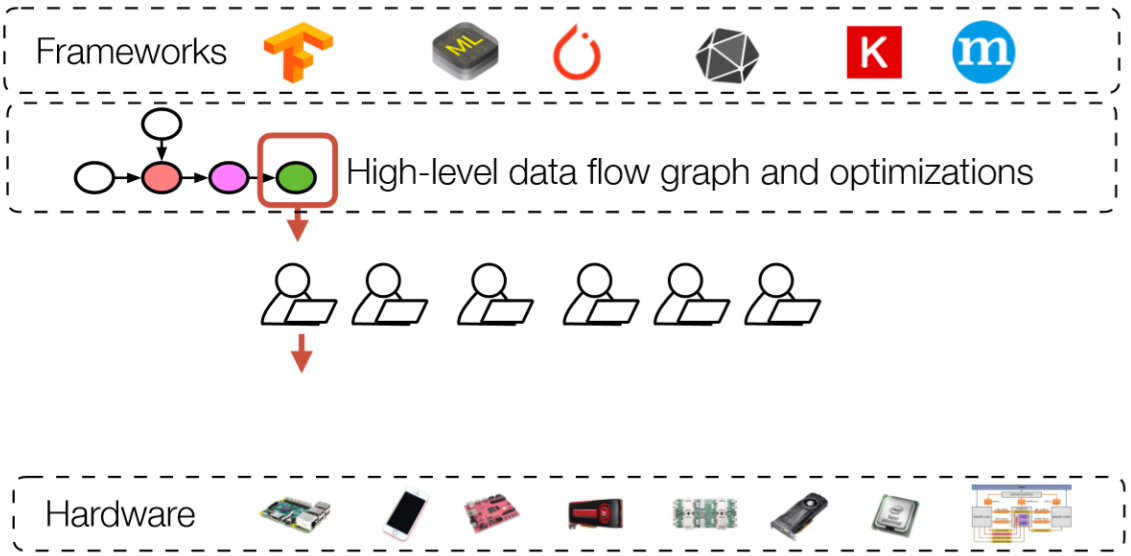
Limitations of Existing Approach



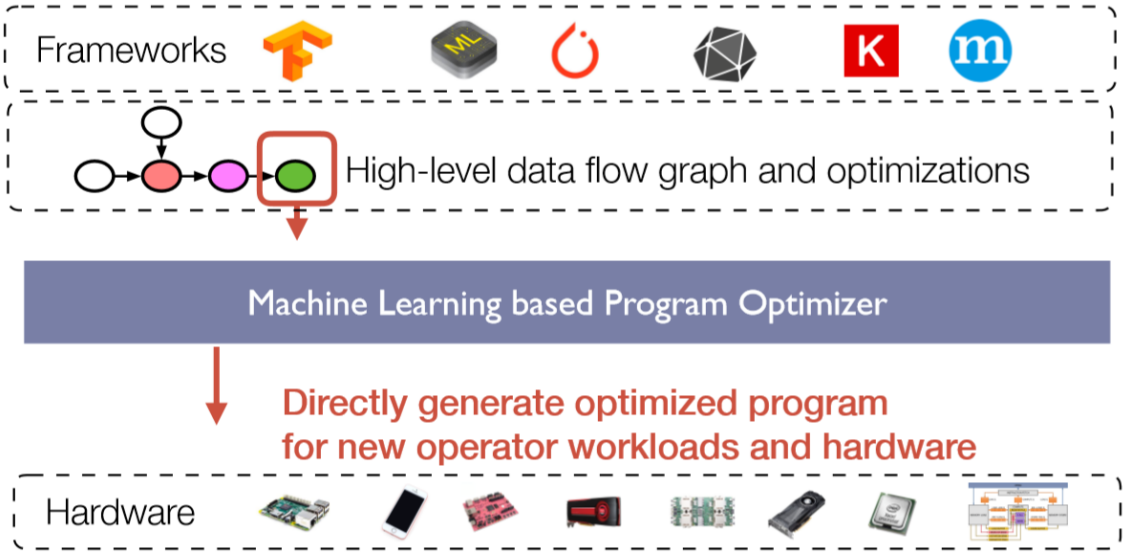
Limitations of Existing Approach



TVM: Learning-based Learning System

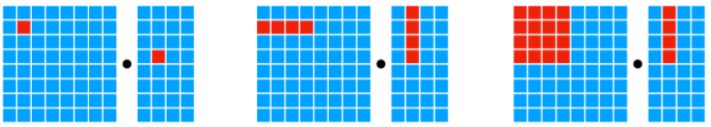


TVM: Learning-based Learning System



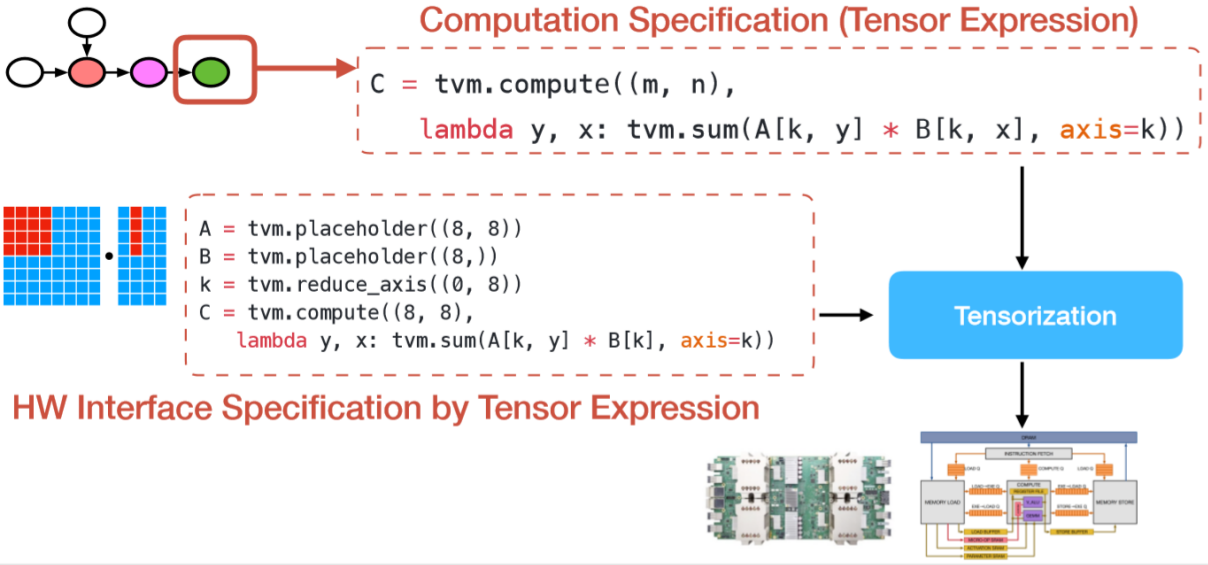
Tensorization Challenge

Compute primitives

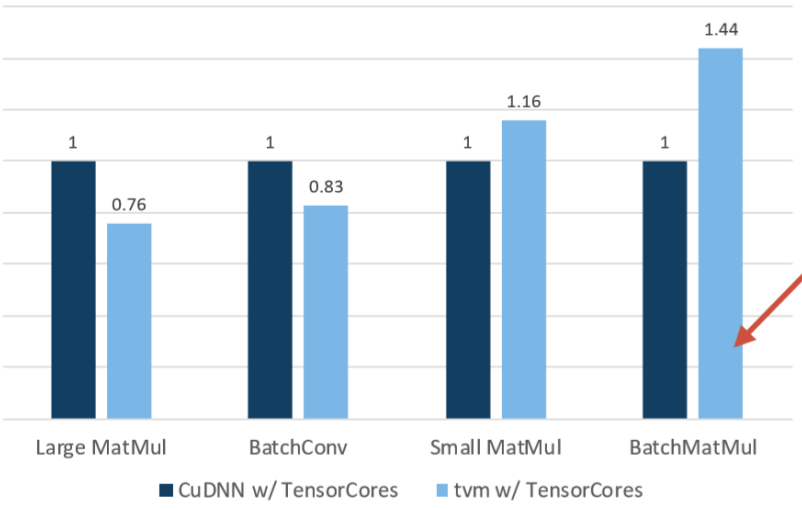


Challenge: Build systems to support emerging tensor instructions

Tensorization Challenge

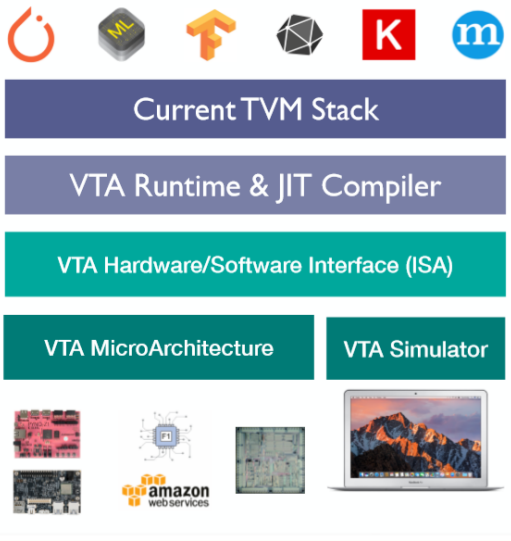


TVM for TensorCore



1.4x better on emerging workloads Transformer related workloads

VTA: Open & Flexible Deep Learning Accelerator



- Runtime JIT compile accelerator micro code
- Support heterogenous devices, 10x better than CPU on the same board.
- Move hardware complexity to software
- VTA 2.0 release - Chisel

compiler, driver, hardware design full stack open source

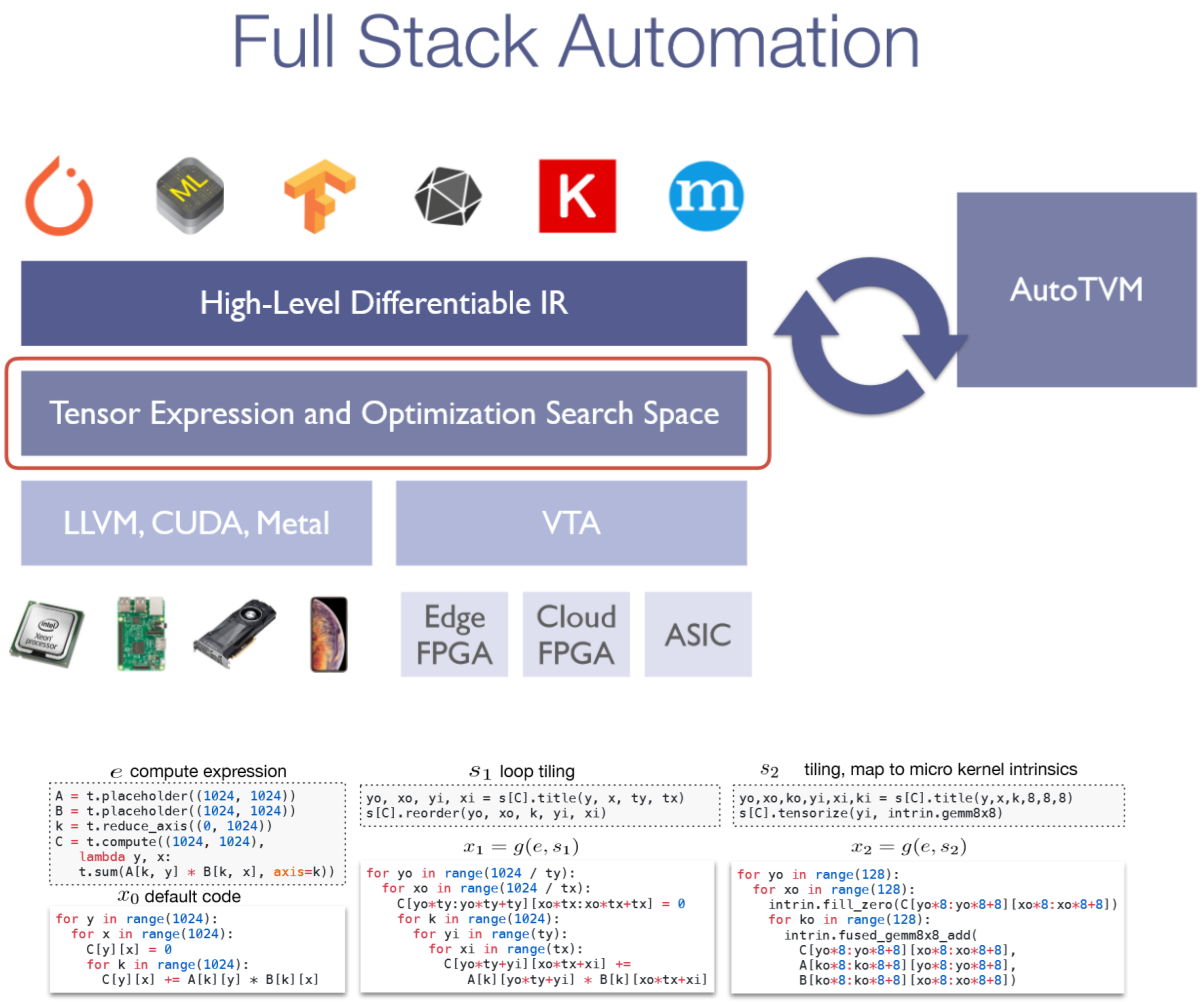


Figure 1: Sample problem. For a given tensor operator specification ($C_{ij} = \sum_k A_{ki} B_{kj}$), there are multiple possible low-level program implementations, each with different choices of loop order, tiling size, and other options. Each choice creates a logically equivalent program with different performance. Our problem is to explore the space of programs to find the fastest implementation.

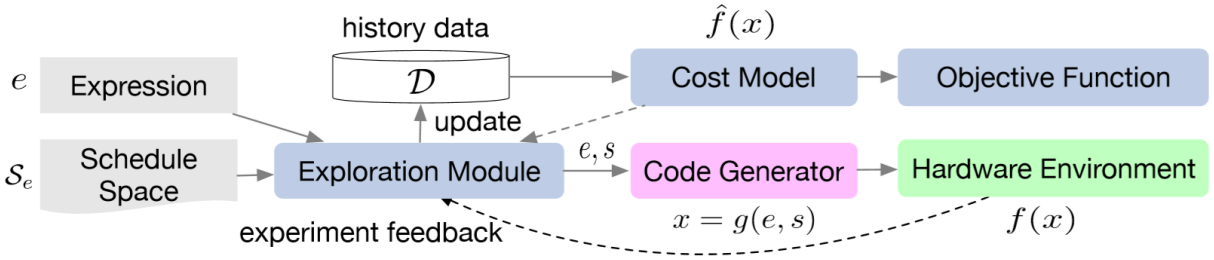


Figure 2: Framework for learning to optimize tensor programs.

We use $\mathcal{S}_e$ to denote the space of possible transformations (schedules) from e to low-level code. For an $s \in \mathcal{S}_e$, let $x = g(e, s)$ be the generated low-level code. Here, g represents a compiler framework that generates low-level code from e, s. We are interested in minimizing $f(x)$, which is the real run time cost on the hardware. Importantly, we do not know an analytical expression for $f(x)$ but can query it by running experiments on the hardware. For a given tuple of $(g, e, \mathcal{S}_e, f)$, our problem can be formalized as the following objective:

$$\arg\min_{s \in \mathcal{S}_e} f(g(e, s)) \tag{1}$$

We can choose from multiple objective functions to train a statistical cost model for a given collection of data $\mathcal{D} = \{(e_i, s_i, c_i)\}$. A common choice is the regression loss function $\sum_i (\hat{f}(x_i) - c_i)^2$, which encourages the model to predict cost accurately. On the other hand, as we care only about the relative order of program run times rather than their absolute values in the selection process, we can instead use the following rank loss function [6]:

$$\sum_{i,j} \log(1 + e^{-\operatorname{sign}(c_i - c_j)(\hat{f}(x_i) - \hat{f}(x_j))}). \tag{2}$$

We can use the prediction $\hat{f}(x)$ to select the top-performing implementations.

Algorithm 1: Learning to Optimize Tensor Programs

Input : Transformation space $\mathcal{S}_e$
Output : Selected schedule configuration s^*
$\mathcal{D} \leftarrow \emptyset$
while *n_trials* $<$ *max_n_trials* **do**
  `// Pick the next promising batch`
  $Q \leftarrow$ run parallel simulated annealing to collect candidates in $\mathcal{S}_e$ using energy function $\hat{f}$
  $S \leftarrow$ run greedy submodular optimization to pick $(1 - \epsilon)b$-subset from Q by maximizing Equation 3
  $S \leftarrow S \cup \{$ Randomly sample ϵb candidates. $\}$
  `// Run measurement on hardware environment`
  for s ***in*** S **do**
    $c \leftarrow f(g(e, s))$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{(e, s, c)\}$
  end
  `// Update cost model`
  update $\hat{f}$ using $\mathcal{D}$
  n_trials $\leftarrow$ n_trials $+ b$
end
$s^* \leftarrow$ history best schedule configuration