An ML-Driven Autoconfigurator for Sparse Tensor Kernels in MLIR

Gus Smith* w/ Aart Bik, Penporn Koanantakool, and Mangpo Phothilimthana (Google)

*on internship from UW

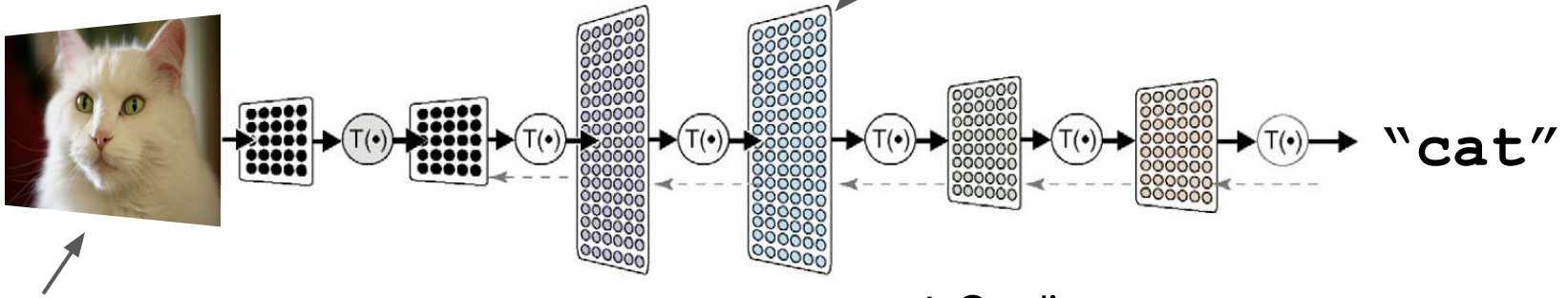
sparsity is everywhere!

2. Activation

ReLU, dropout, etc.

3. Weight

Pruning, regularization, non-cuboid filters, etc.



1. Input

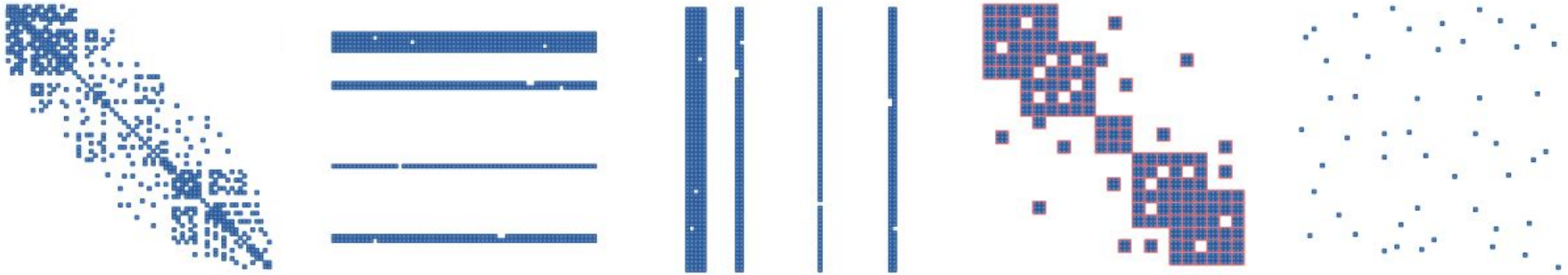
Naturally sparse, embedding lookups, sparse projection tensors, etc.

4. Gradient

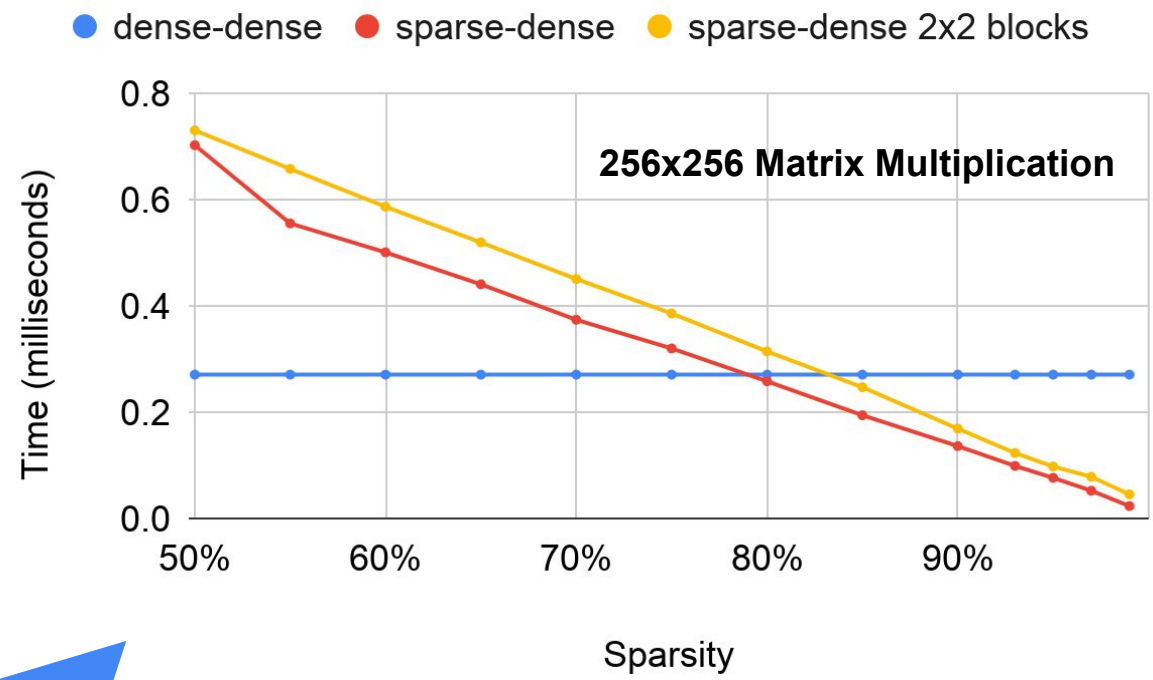
Pruning, gradually sending updates

(from Penporn Koanantakool)

sparsity takes many forms!



The tensor algebra compiler. Kjolstad et. al. 2017.

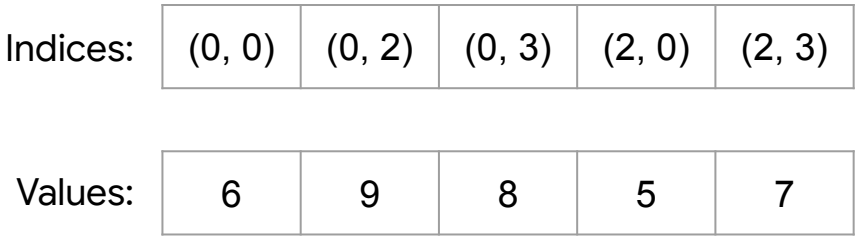


but, if we can exploit sparity, it's immensely beneficial!

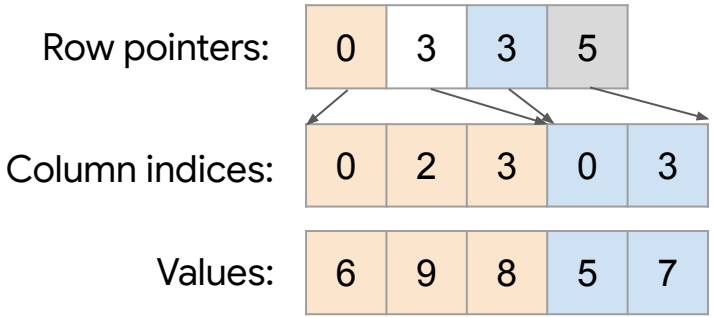
(from Penporn Koanantakool)

How do we exploit sparsity?

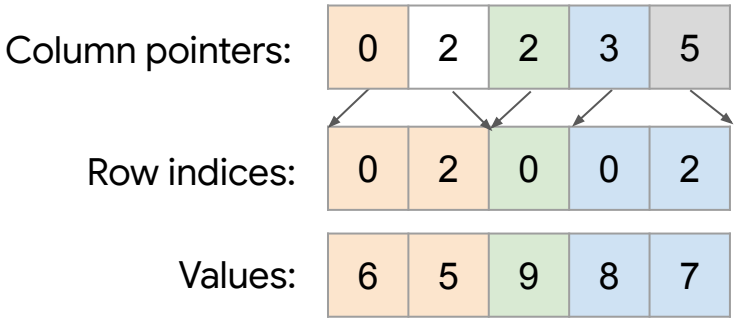
With sparsity formats!



Coordinate (COO)



Compressed Sparse Row (CSR)



Compressed Sparse Column (CSC)

plus DIA, ELL, and more…

…and then along came TACO…

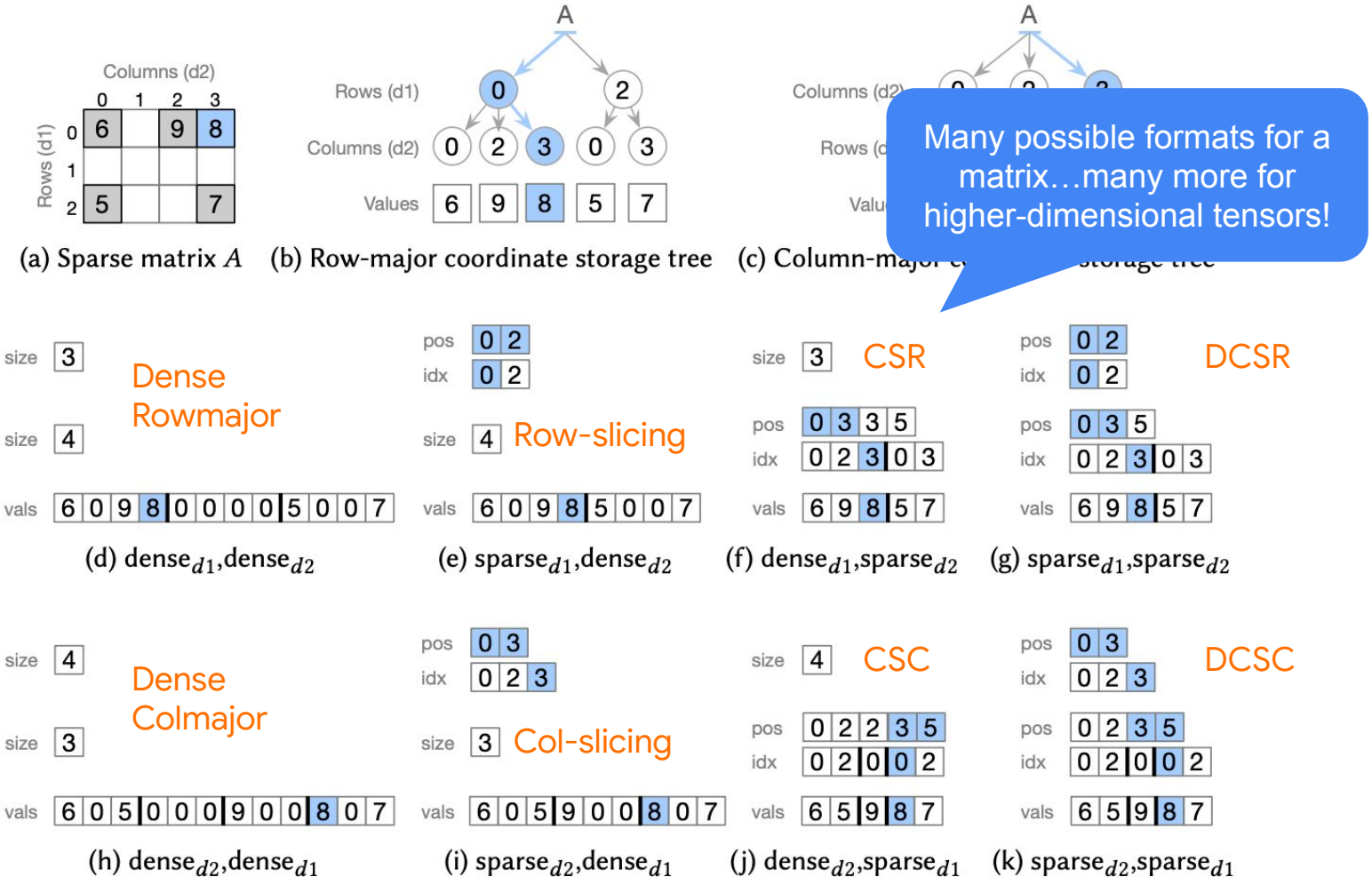
Dimension-wise Specification

- Each dimension can be sparse or dense
- Define traversal order: (1st dim, 2nd dim, ...)
- For a k^{th}-order tensor, this covers $k!2^k$ formats

Dense: All elements in that dimension are stored (e.g., dense row pointers in CSR).

Sparse: Zeros in that dimension aren't stored.

(from Penporn Koanantakool)



The tensor algebra compiler. Kjolstad et. al. 2017.

thanks to Aart and others, we can now easily configure sparse kernels in MLIR!

```
#CSR = #sparse_tensor.encoding<{
  dimLevelType = [ "dense", "compressed" ],
  dimOrdering = affine_map<(d0, d1) -> (d0, d1)>
}>

func @kernel(%a: tensor<?x?xf64, #CSR>,
             %b: tensor<?x?xf64>,
             %c: tensor<?x?xf64>) -> tensor<?x?xf64> {

  ...kernel implementation...

  return %d : tensor<?x?xf64>
}
```

Goal: given a kernel, can we configure its sparse format?

couldn’t you just brute force it?

Goal: given **any** kernel, can we **quickly** configure its sparse format?

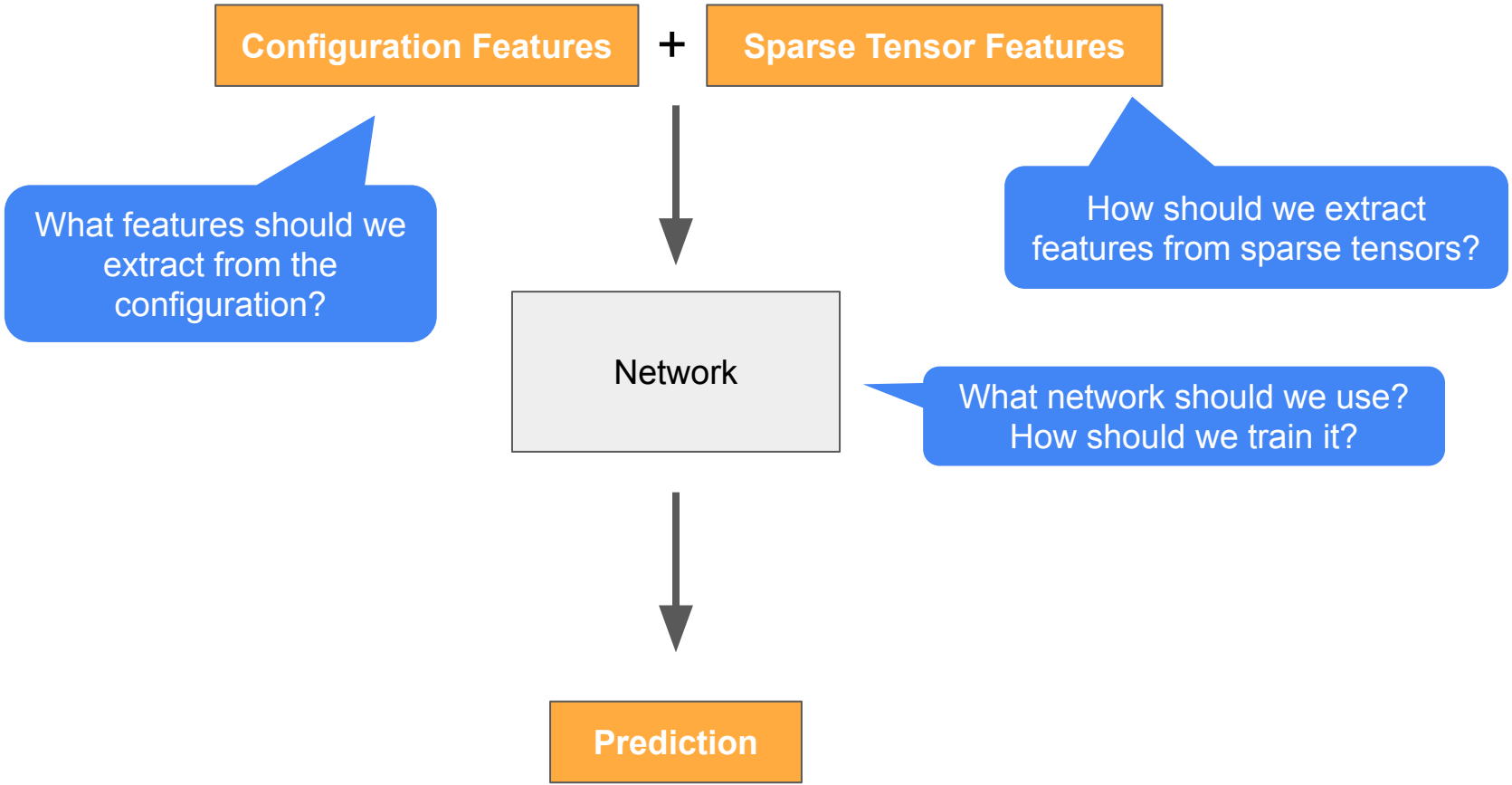
so we could use it in a JIT!

We use a “standard” ML-for-ML approach:

1. Train a cost model
2. Use the cost model in a search procedure

(A Learned Performance Model for Tensor Processing Units (Kaufman et. al.)
Learning to Optimize Halide with Tree Search and Random Programs (Adams et. al.)
Learning to Optimize Tensor Programs (Chen et. al.))

Training a Cost Model


Configuration Features
+
Sparse Tensor Features
What features should we extract from the configuration?
How should we extract features from sparse tensors?
Network
What network should we use? How should we train it?
Prediction

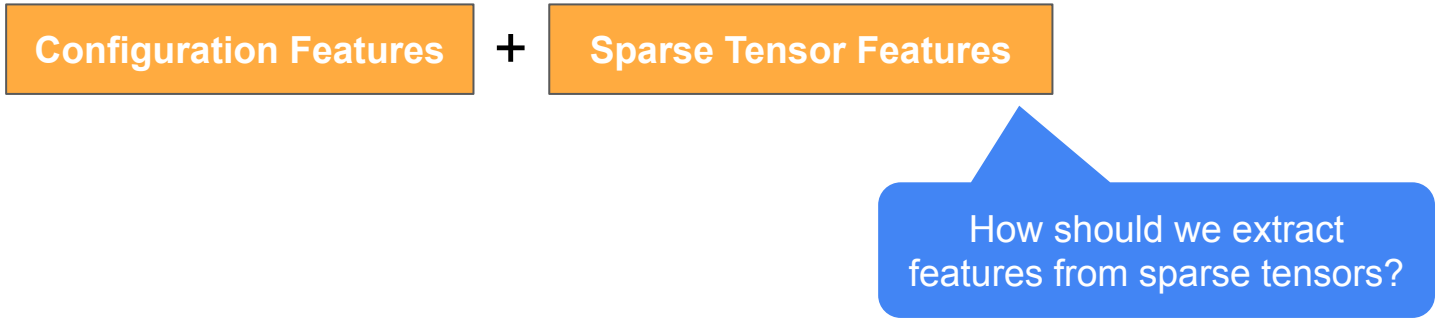
Configuration Features

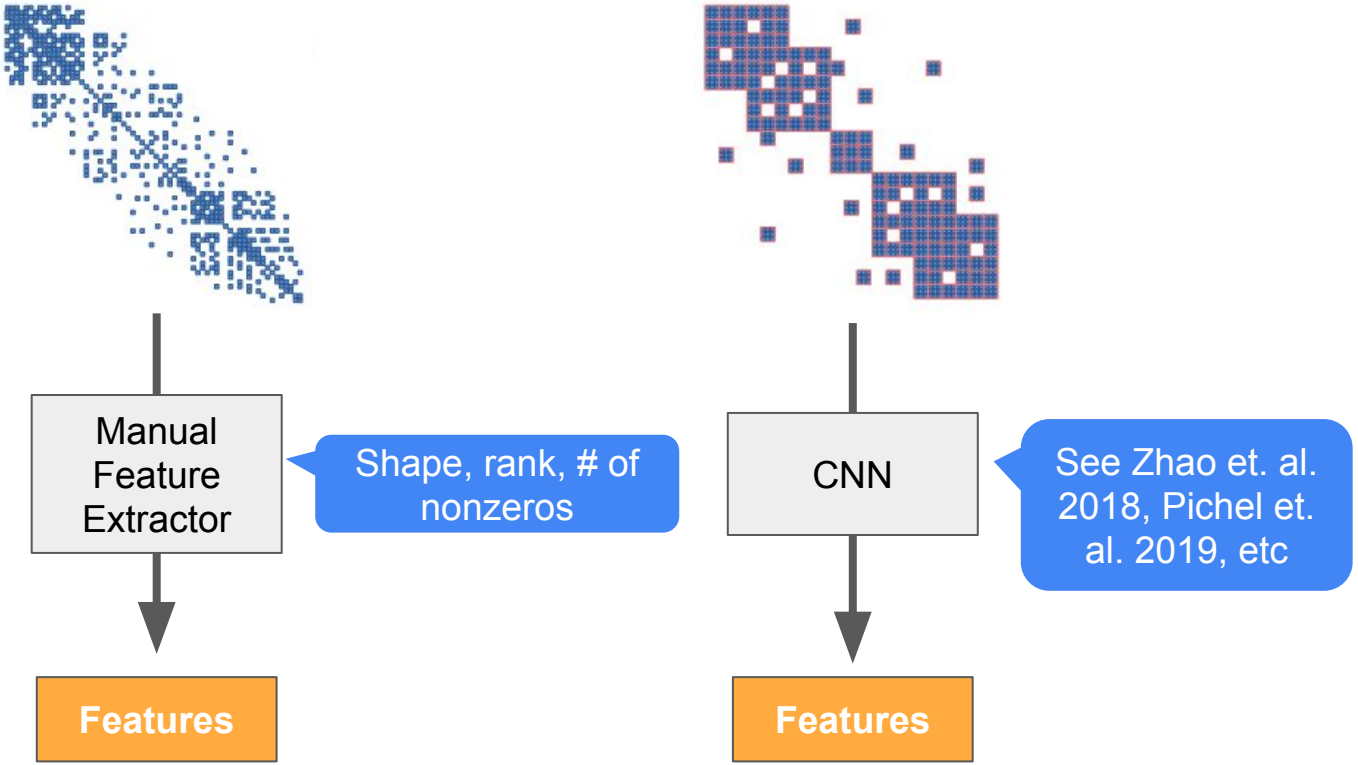
What features should we extract from the configuration?

Configuration Features

- Entire sparse kernel configuration is easily featurized by packing into fixed-length vector
- Per-dimension sparse formats, dimension ordering, plus other settings e.g. parallelization and vectorization levels

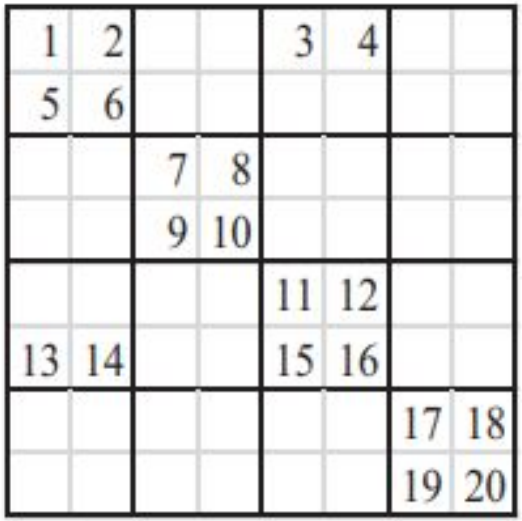
For sparse-dense matmul kernel: ~9k configurations


Configuration Features
+
Sparse Tensor Features
How should we extract features from sparse tensors?



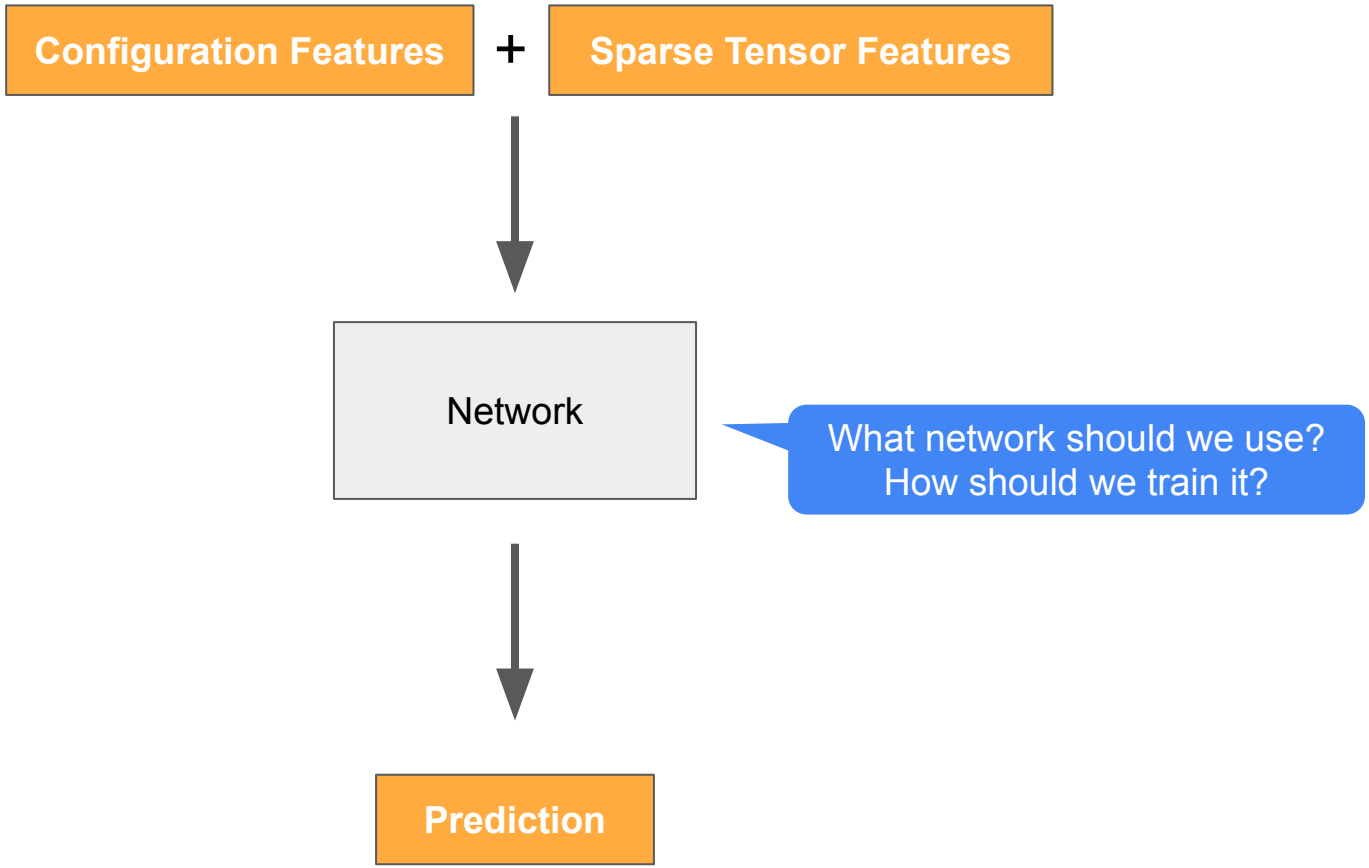
Bridging the Gap between Deep Learning and Sparse Matrix Format Selection, Zhao et. al 2018
Sparse Matrix Classification on Imbalanced Datasets Using Convolutional Neural Networks, Pichel et. al. 2019
IA-SpGEMM: an input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication, Xie et. al. 2019

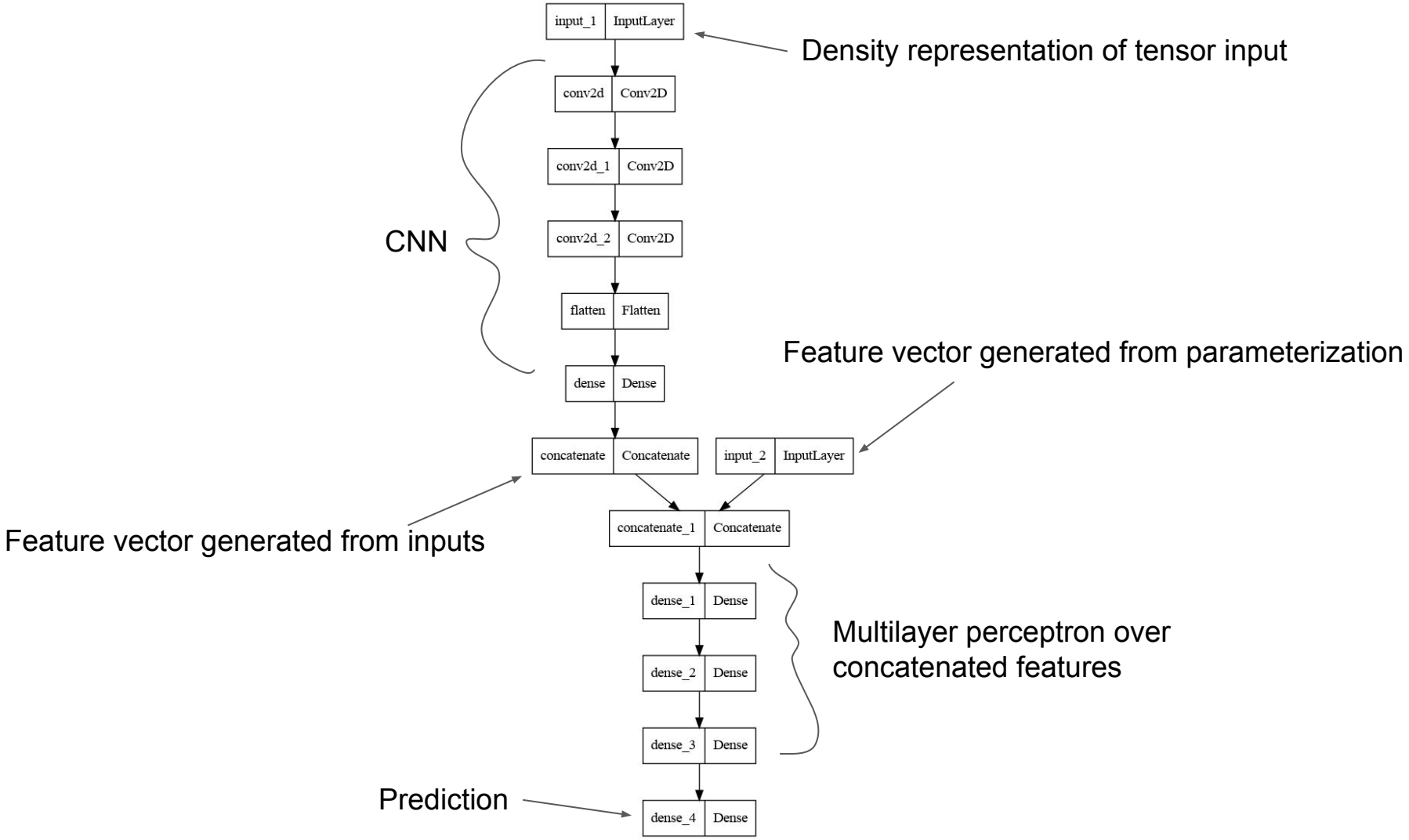
Variable-sized **sparse** input matrix

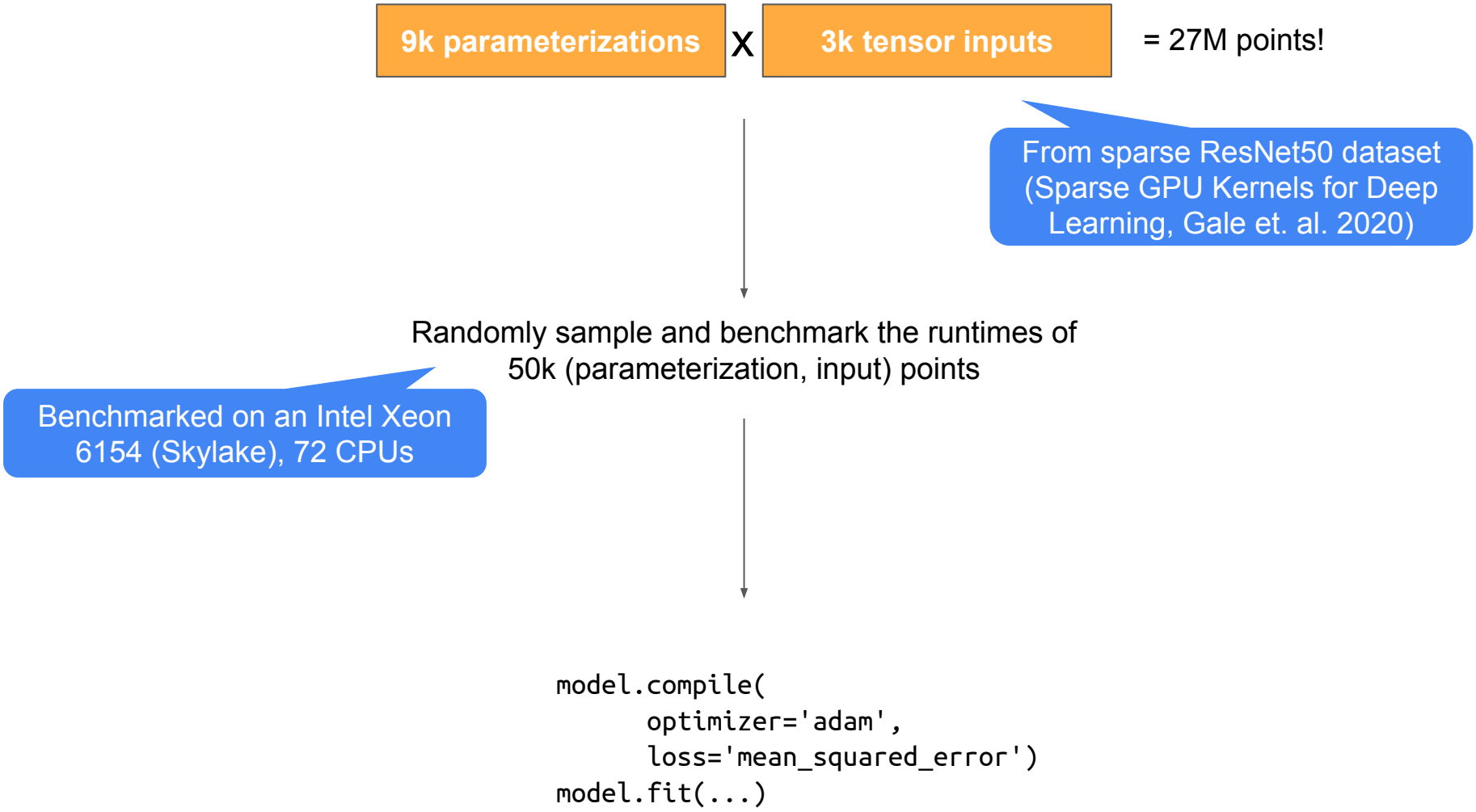


Now we can perform dense 2D convolution!

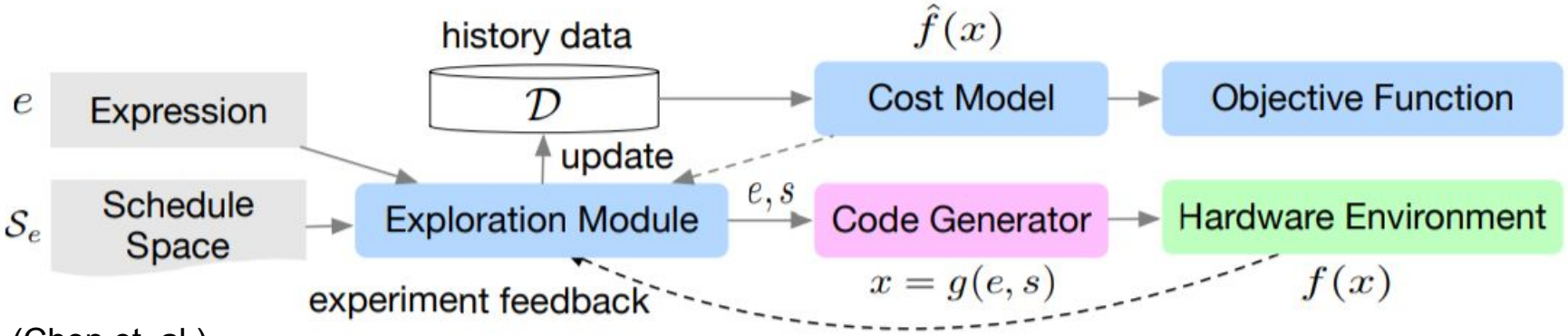
IA-SpGEMM: an input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication, Xie et. al. 2019


Configuration Features
+
Sparse Tensor Features
Network
What network should we use?
How should we train it?
Prediction

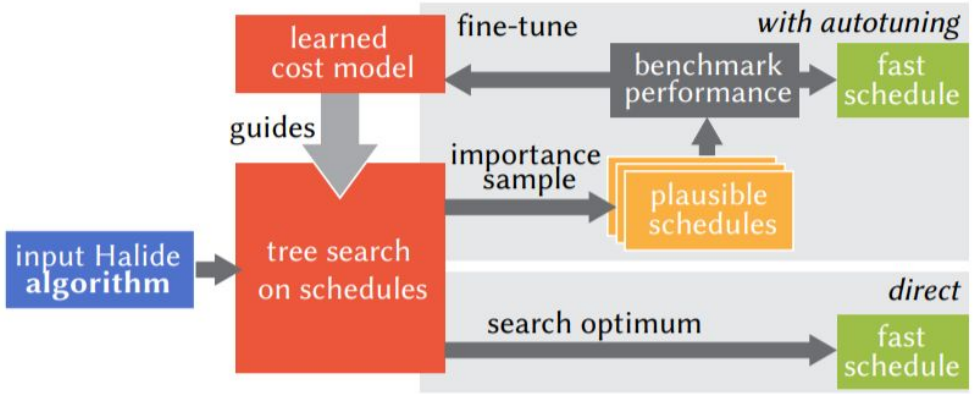

input_1 InputLayer
Density representation of tensor input
conv2d Conv2D
conv2d_1 Conv2D
conv2d_2 Conv2D
CNN
flatten Flatten
Feature vector generated from parameterization
dense Dense
concatenate Concatenate
input_2 InputLayer
Feature vector generated from inputs
concatenate_1 Concatenate
dense_1 Dense
dense_2 Dense
dense_3 Dense
Multilayer perceptron over concatenated features
Prediction
dense_4 Dense


9k parameterizations
X
3k tensor inputs
= 27M points!
From sparse ResNet50 dataset (Sparse GPU Kernels for Deep Learning, Gale et. al. 2020)
Randomly sample and benchmark the runtimes of 50k (parameterization, input) points
Benchmarked on an Intel Xeon 6154 (Skylake), 72 CPUs
model.compile(
optimizer='adam',
loss='mean_squared_error')
model.fit(...)

Search



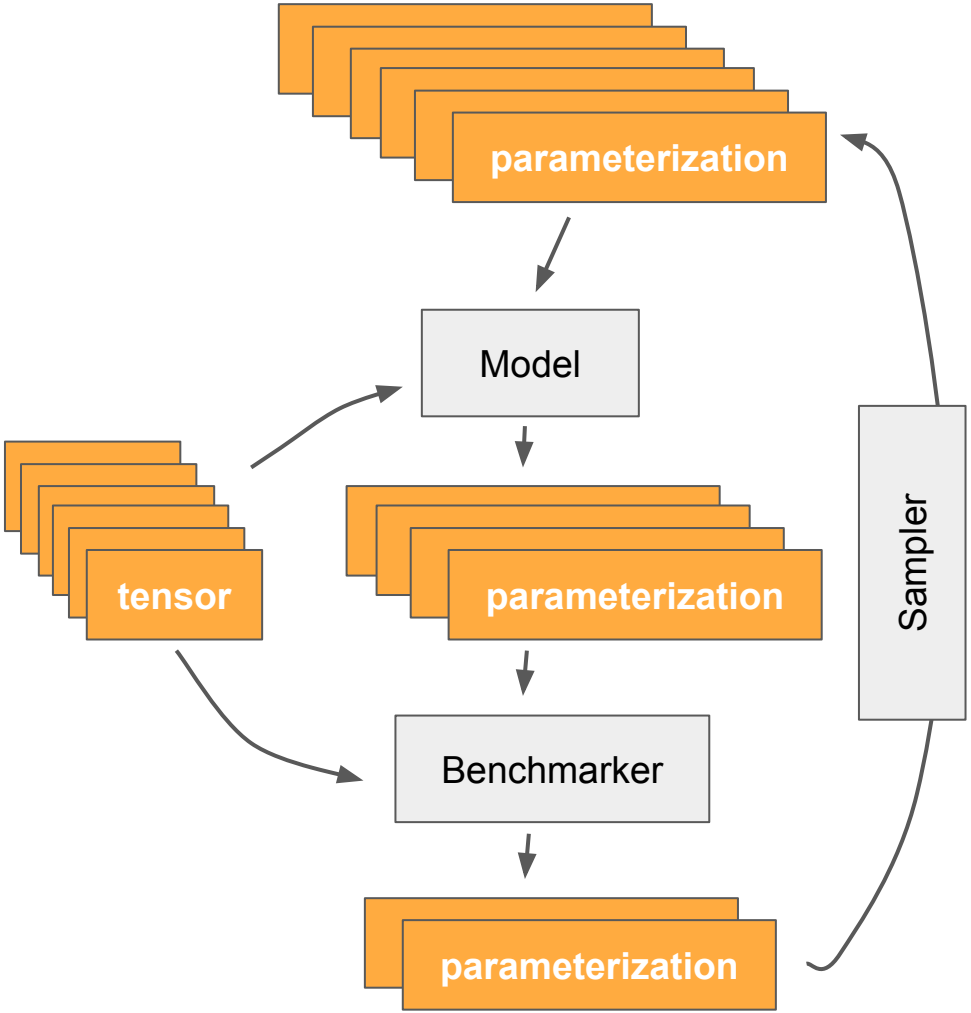
(Chen et. al.)

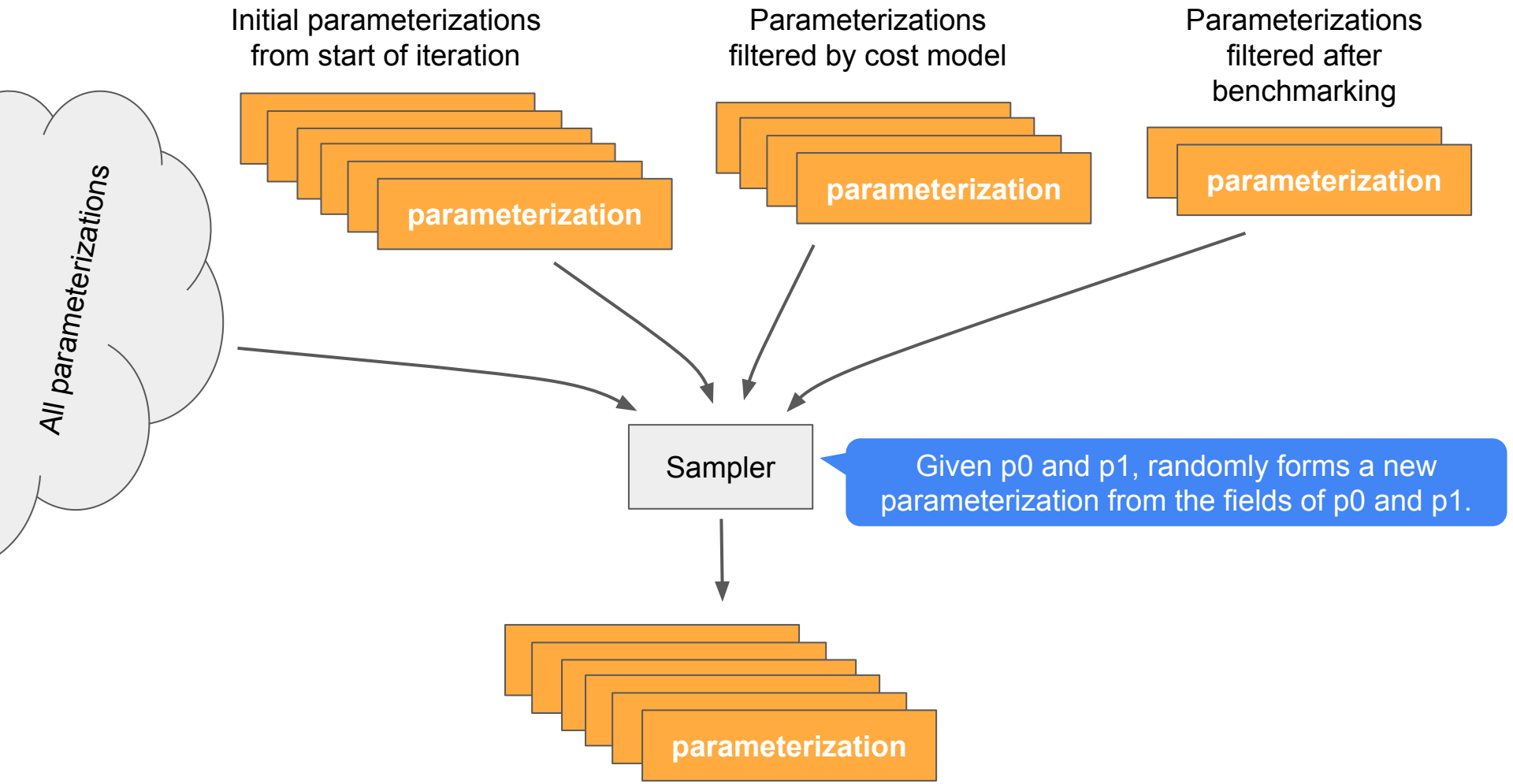


(Adams et. al.)

In related works, search follows similar structure

1. Randomly select test inputs
2. Randomly sample initial parameterizations
3. For each search round:
    a. Use the cost model to filter the set of candidates (fast)
    b. Benchmark the filtered set of candidates (slow)
    c. Record top benchmarked candidates
    d. Construct next generation of parameterizations




Initial parameterizations from start of iteration
Parameterizations filtered by cost model
Parameterizations filtered after benchmarking
All parameterizations
parameterization
parameterization
parameterization
Sampler
Given p0 and p1, randomly forms a new parameterization from the fields of p0 and p1.
parameterization

Future work

- Parameterize cost model over kernels
  - Generate features from kernels, e.g. via manual feature extraction or GNN
- Train on data from multiple kernels (SDDMM; random sparse kernels)

Thank you!