Google Research

A Flexible Approach to Autotuning Multi-Pass Machine Learning Compilers

Phitchaya Mangpo Phothilimthana, Amit Sabne, Nikhil Sarda, Karthik Srinivasa Murthy, Yanqi Zhou, Christof Angermueller, Mike Burrows, Sudip Roy, Ketan Mandke, Rezsa Farahani, Yu Emma Wang, Berkin Ilbeyi, Blake Hechtman, Bjarke Roune, Shen Wang, Yuanzhong Xu, and Samuel J. Kaufman*

Google, *University of Washington

Google Research

Search-Based ML Compilers

optimization scope

graph	TASO PET	DeepCuts
subgraph		TVM Halide TensorComp... FlexTensor Ansor AdaTune Chameleon

Google Research

Search-Based ML Compilers

optimization scope		
graph	TASO PET	DeepCuts
subgraph		TVM Halide TensorComp... FlexTensor Ansor AdaTune Chameleon

Google Research

Search at Subgraph Level is Suboptimal

A common strategy **partitions** a graph into subgraphs **according to the neural net layers**, ignoring cross-layer optimization opportunities.

Empirical result: a **regression** of **up to 2.6x** and **32% on average** across 150 ML models by limiting fusions in XLA to be within layers.

Google Research

Search-Based ML Compilers

optimization scope

graph	TASO PET	DeepCuts
subgraph		TVM Halide TensorComp... FlexTensor Ansor AdaTune Chameleon

Google Research

Search Approaches: Long Compile Time

optimization scope	seconds	minutes	hours
graph	XLA	TASO PET	DeepCuts
subgraph			TVM Halide TensorComp... FlexTensor Ansor AdaTune Chameleon

compile time (for ResNet like inference)

Google Research

Production Compilers: Multi-Pass

- Models evaluated by research compilers: up to 1,000 node
- Industrial-scale models: up to **500,000 nodes!**
- That's why **production ML compilers** still decompose the compilation into **multiple passes**.
- **None** of the existing approaches **support** autotuning different optimizations in a **multi-pass compiler**.
  - **Challenge**: search space of a pass is highly dependent on decisions made in prior passes.

Our Goal

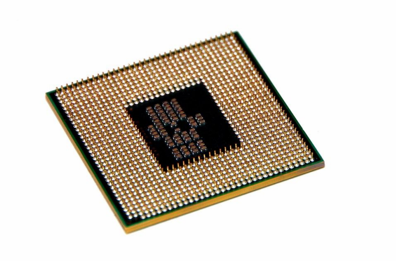
Bring the benefit of **search-base** exploration to **multi-pass compilers**:

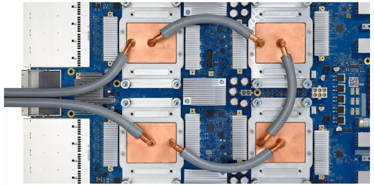
- for both graph and subgraph levels
- with flexibility via configurable search to tune subset of optimizations of interest

"A Flexible Approach to Autotuning Multi-Pass Machine Learning Compilers"

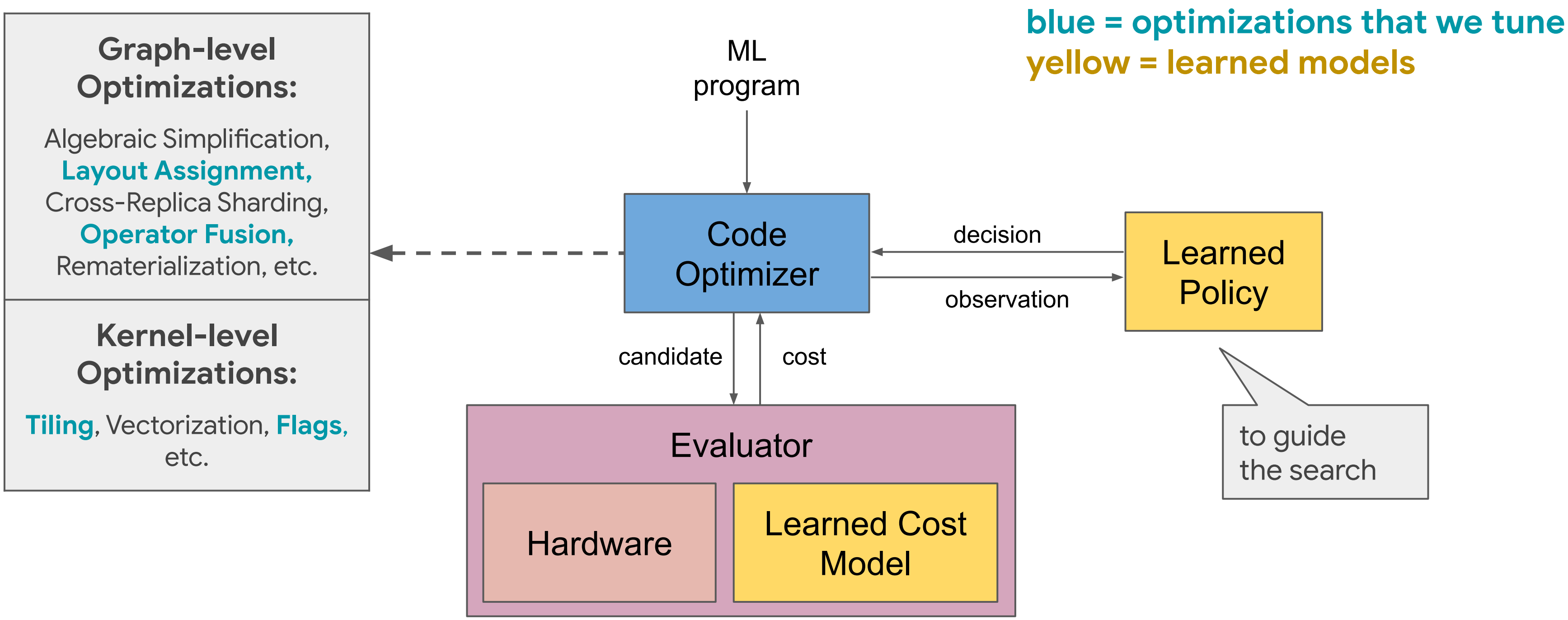
Production ML Compilation Stack at Google

PyTorch





XTAT: XLA TPU Autotuner



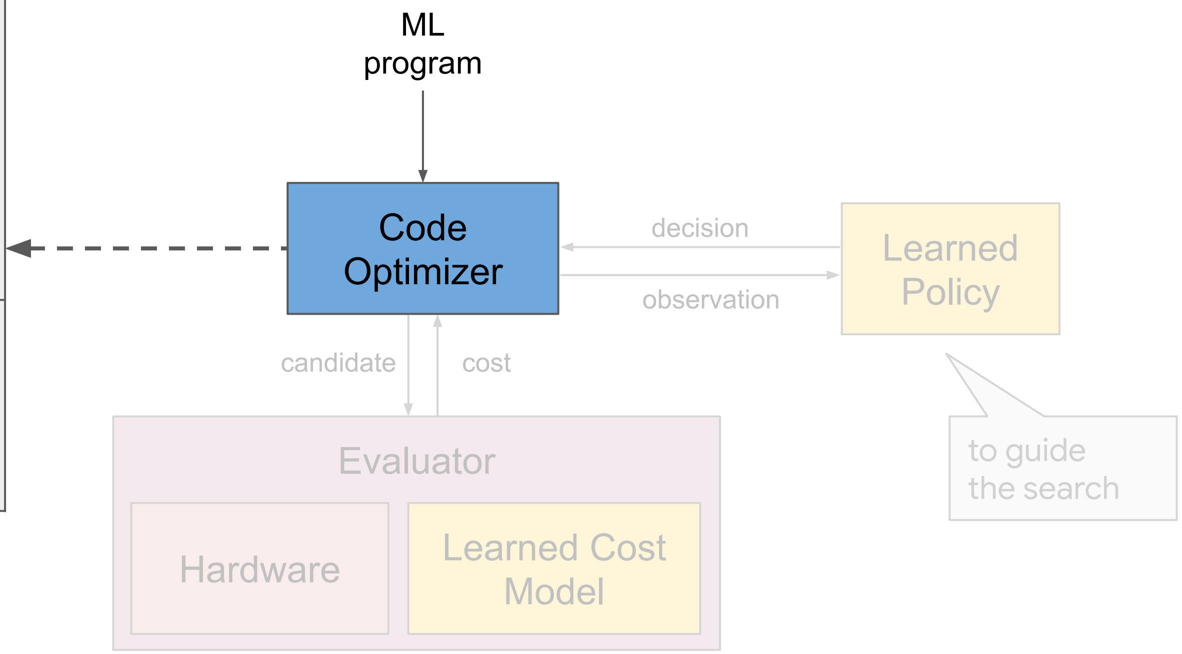
XTAT: XLA TPU Autotuner

Graph-level Optimizations:

Algebraic Simplification, **Layout Assignment,** Cross-Replica Sharding, **Operator Fusion,** Rematerialization, etc.

Kernel-level Optimizations:

Tiling, Vectorization, **Flags,** etc.

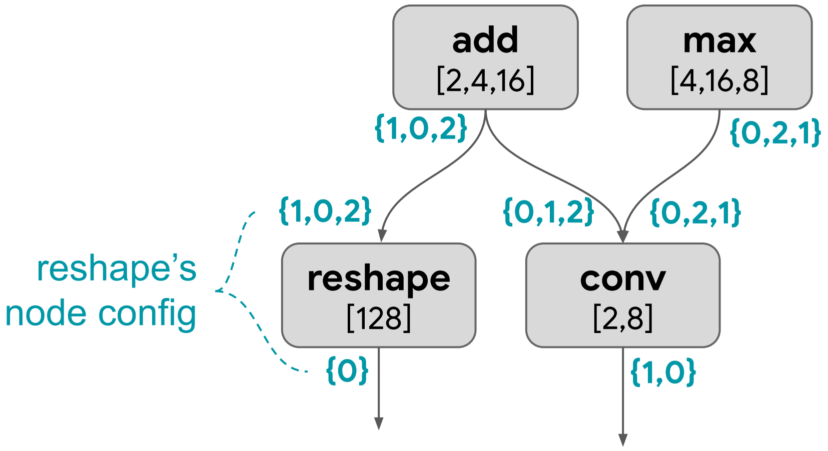


Pass Configuration

configuration on a tensor graph
for an **optimization pass**
is
a collection of per-node configurations that control
how the pass transforms each node in the graph

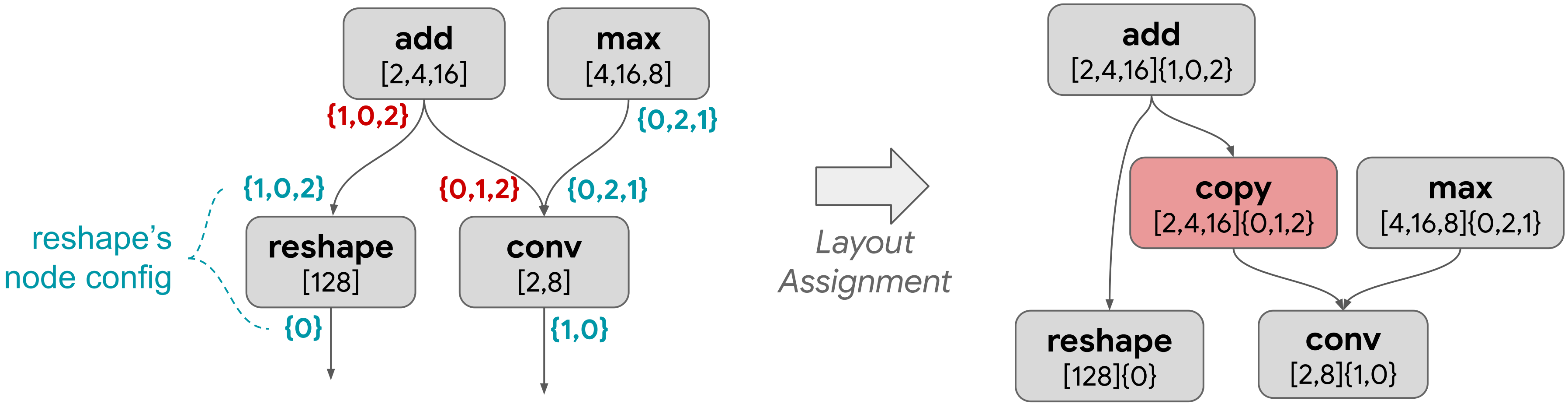
Layout Assignment

Example:



Layout Assignment

Example:



Layout Search Space

Option #1: Naive

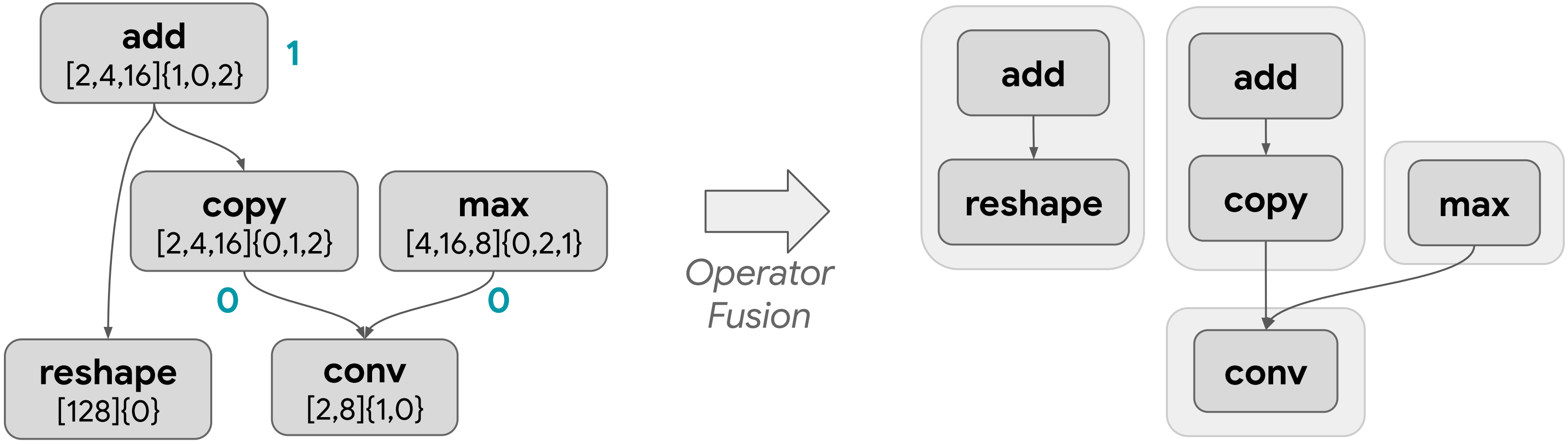
- Layout options for **each input/output** are **permutation** of its dimensions.
- Many **invalid configs** because there are constraints between tensors.

Option #2: Proposed

- Tune **layout options for important ops** (convolution and reshape).
- For each important op, get valid input-output layouts from compiler.
- Leverage XLA **layout propagation** algorithm.

Operator Fusion

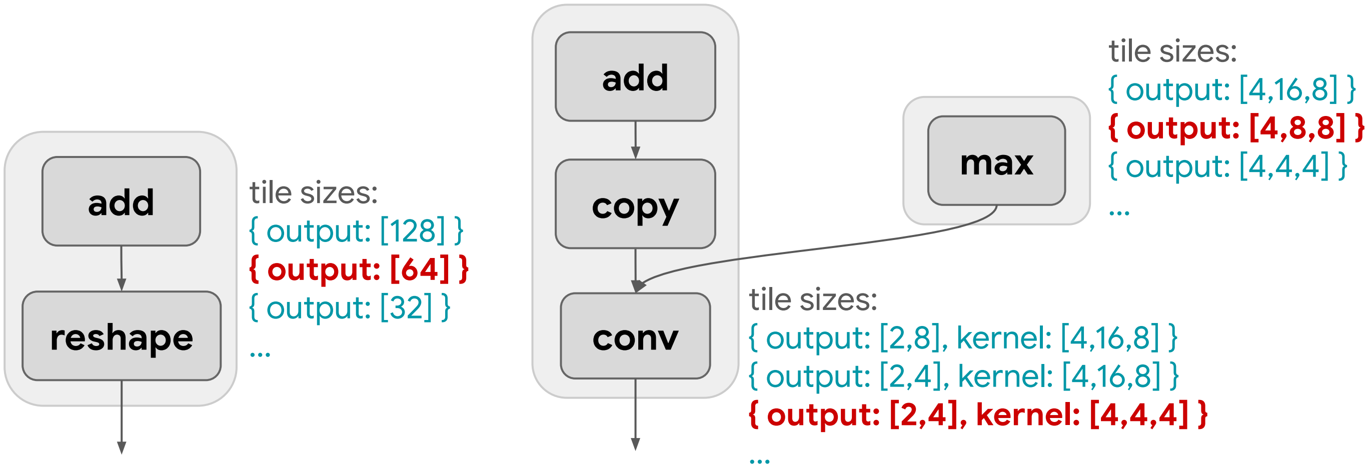
Example:



Google Research

Tile Size & Code Gen Flags Search Space

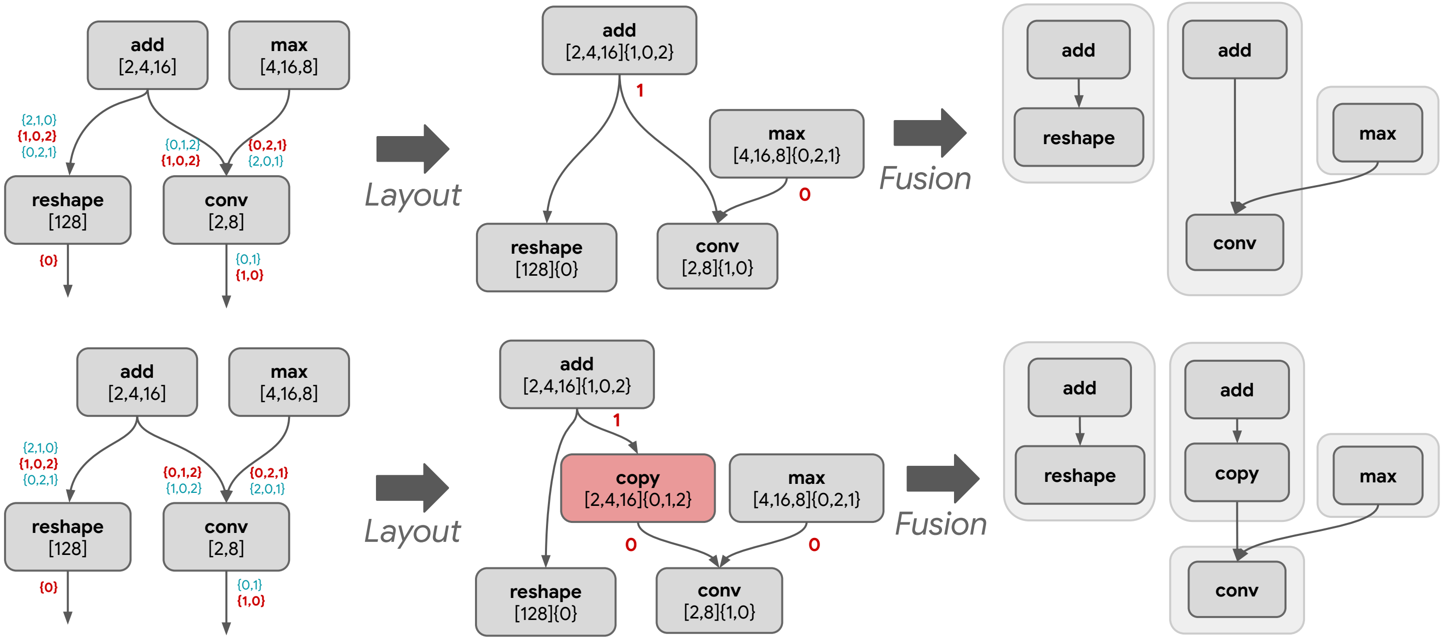
Tune config for each fused node (kernel) independently.



Joint Autotuning: Challenges

g_A --- $A(config_A)$ ---> g_B --- $B(config_B)$ ---> g_{out}

Layout *Fusion*



$config_A$ determines the input graph g_B to pass B and its search space

When we change $config_A$ to $config_A'$, g_B is changed, and $config_B$ is no longer valid.

How to not start the search for B from scratch when $config_A$ is changed?

Methodology for Joint Autotuning

function SEARCHSTEP(C)
    $opt_{id} \leftarrow$ SelectOpt$(Opts)$
    $C' \leftarrow$ GenerateCandidates(opt_{id}, C)
    for $c : C'$ **do**
        $UpdateAndApplyCandidate(opt_{id}, c)$
        $Evaluate(c)$
    end for
    return SelectCandidates(C, C')
end function

Returns:
A, A, ..., B, B, ..., C, C, ... **(sequential)**
A, B, C, A, B, C, ... **(joint tuning)**
or some combinations of them

Methodology for Joint Autotuning

function SEARCHSTEP(C)
  $opt_{id} \leftarrow$ **SelectOpt**($Opts$)
  $C' \leftarrow$ **GenerateCandidates**(opt_{id}, C)
  for $c : C'$ **do**
    $UpdateAndApplyCandidate(opt_{id}, c)$
    $Evaluate(c)$
  end for
  return **SelectCandidates**(C, C')
end function

Candidate c:
c.graphs = [g_A, g_B, g_{out}]
c.configs = [$config_A$, $config_B$]

Change $config_A$:
c.graphs = [g_A, g_B, g_{out}]
c.configs = [$config_A'$, $config_B$]

Fix c to be *well-formed*:
c.graphs = [g_A, g_B', g_{out}']
c.configs = [$config_A'$, $config_B'$]

Construct Well-Formed Candidate

Key ideas:

- Update subsequent graphs
- Update config_B' to have configurations for all nodes in g_B' from:
    - **config_B**
    - **global configuration store** (maintaining the best config per node)
    - **default value**

Change config_A:
c.graphs = [g_A, g_B, g_{out}]
c.configs = [config_A', config_B]

Fix c to be *well-formed*:
c.graphs = [g_A, g_B', g_{out}']
c.configs = [config_A', config_B']

Google Research

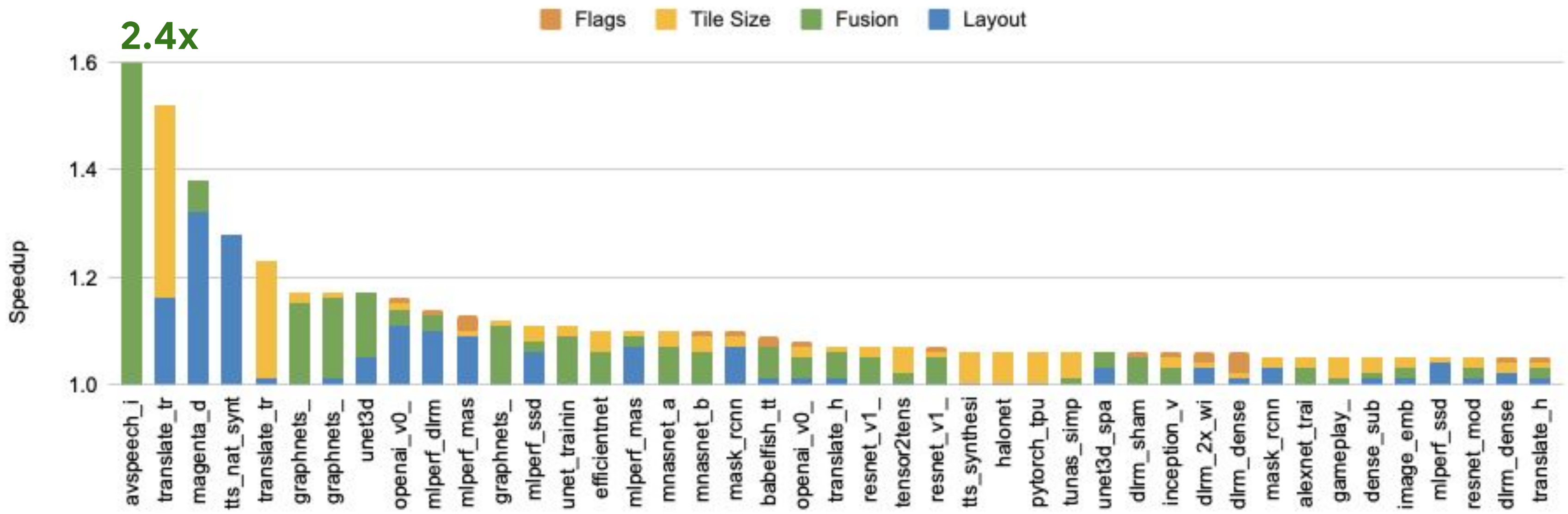
End-to-End Search Schedule

- Separate tuning graph-level and kernel-level optimizations for scalability
- Tuning **layout + fusion jointly is better** than sequentially
- Tuning **tile size + flag jointly is worse** than sequentially

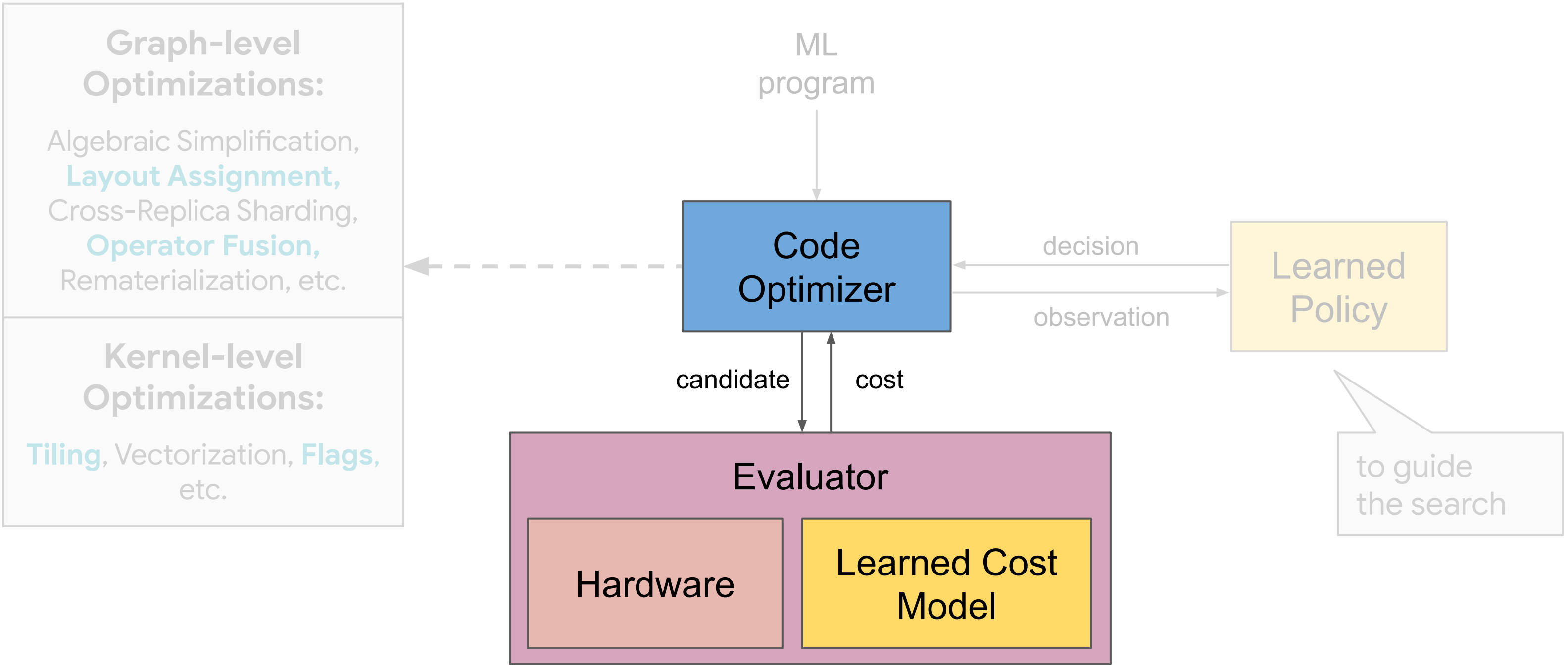
Tune **layout-fusion jointly** (simulated annealing)
⟶ then tune **tile size** (exhaustive)
⟶ then tune code gen **flags** (exhaustive)

End-to-End Runtime Speedup

We measured end-to-end model speedups from autotuning **150 ML models**.
The figure shows models that achieve 5% or more improvement.



Learned Cost Model



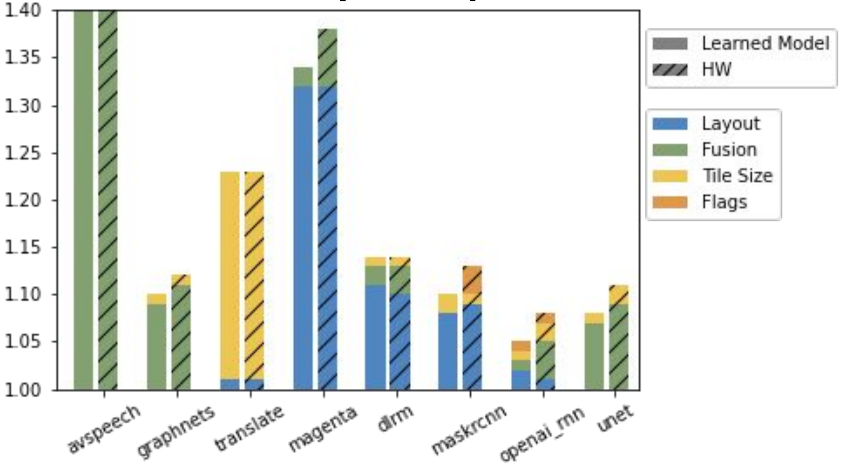
Ref: Kaufman and Phothilimthana et al., A Learned Performance Model for Tensor Processing Units, MLSys 2021.

Tuning with Learned Cost Model

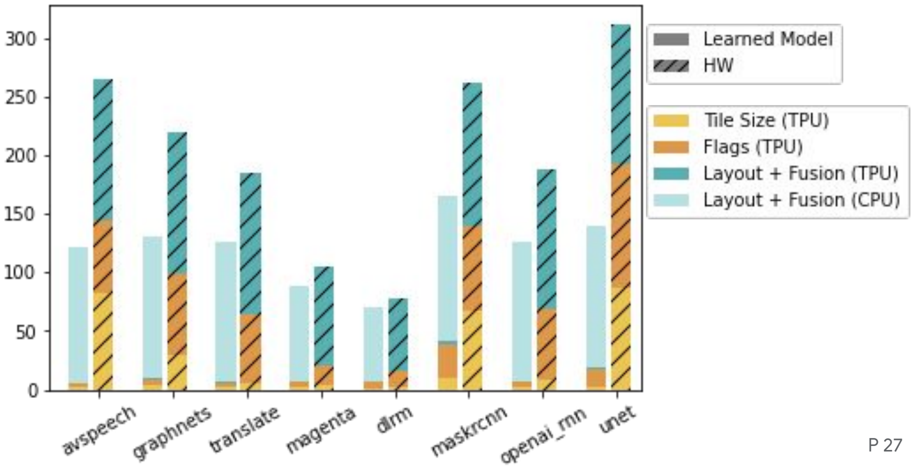
Execute the top k configurations from each worker according to the model on real hardware and pick the best.

- k = 10 for graph-level optimizations
- k = 5 for kernel-level optimizations

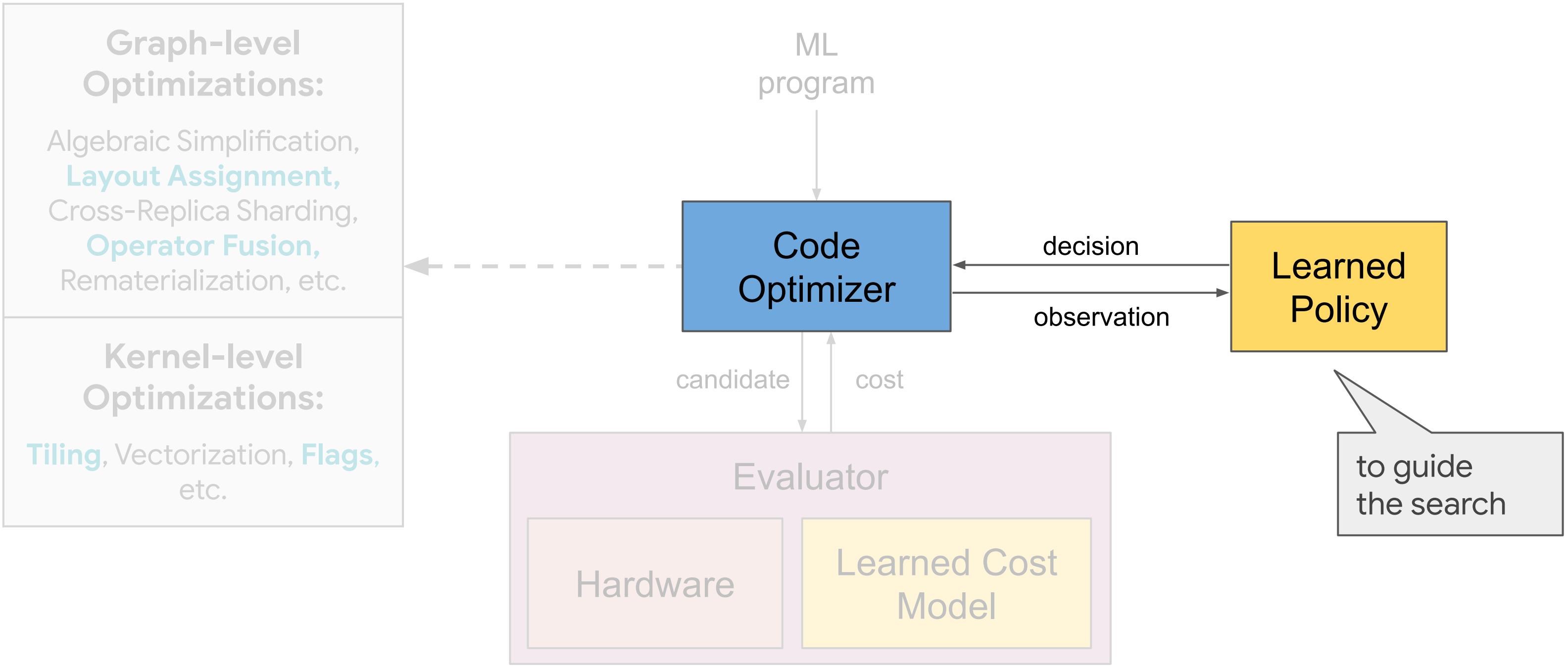
Runtime Speedup (x)



Tuning Time (min)



Search Strategies

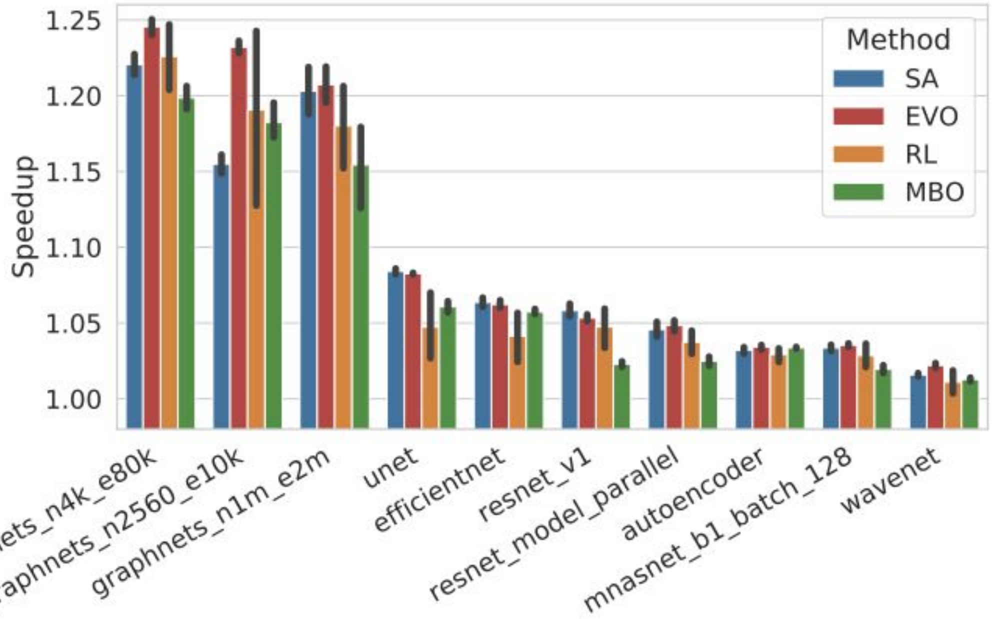


Search Strategies

- Exhaustive
- Simulated annealing (SA)
- Evolutionary (EVO)
- Model-based optimization (MBO)
- Deep reinforcement learning (RL)

Search Strategies: Fusion Autotuning

Average speedup across 10 runs. Each run evaluated 10,000 candidates.



XTAT: XLA TPU Autotuner