R 2 4 4

Tensor Program Optimization with Probabilistic Programs

MetaSchedule · Junru Shao et al., NeurIPS 2022

Presented by Yaxuan Li

PROBLEM

The search space is an engineering problem

A tensor computation has many equivalent implementations — with very different performance.

ORIGINAL LOOP

```
for i in range(1024):
    B[i] = ReLU(A[i])
```

→

AFTER SPLITTING THE LOOP

```
for io in range(256):
    for ii in range(4):
        i = io * 4 + ii
        B[i] = ReLU(A[i])
```

Defining useful candidates requires expert knowledge. MetaSchedule separates two tasks: *defining the space* and *searching it*.

PRIOR APPROACHES

How search spaces were built before

Vendor libraries

cuDNN · MKL

Experts hand-write optimised kernels.

Limitation Cannot keep up with new models and hardware.

Template auto-tuning

AutoTVM

Experts write a template per operator; the system tunes its parameters.

Limitation Space is fixed in advance; heavy manual effort.

Auto-scheduling

Ansor

Generic rules generate the space with little manual work.

Limitation Rules are hard-wired inside the framework.

The common problem: the search space is coupled with the search. *Changing it means changing the framework.*

KEY IDEA

A search space written as a probabilistic program

SAMPLING

```
a, b, c = SampleTile(i, parts=3)
```

TRANSFORMATIONS

```
i0, i1, i2 = Split(i, [a, b, c])
Parallelize(i0)
Vectorize(i2)
```

CONSTRAINT

$a \times b \times c = 1024$

POSSIBLE DRAWS

$32 \times 8 \times 4$

$16 \times 16 \times 4$

Each execution produces **one candidate schedule** — one short program describes the whole space.

KEY IDEA

Choices depend on the current program

EXAMPLE · DENSE + RELU

1 Sample tile sizes and split the loops of the Dense operator.

2 Choose where to fuse ReLU inside the new loop nest.

Later choices depend on earlier transformations — the program can inspect its own state before sampling.

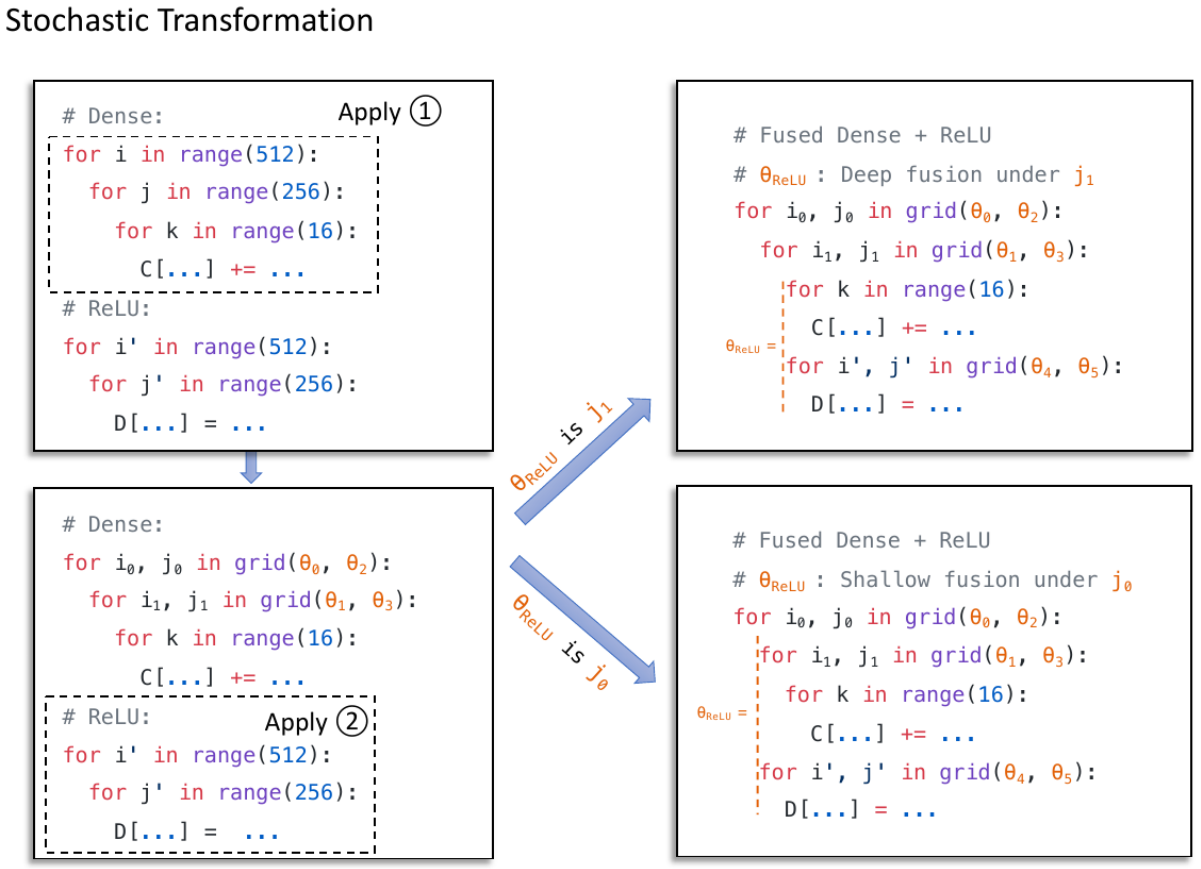


Figure 3, Shao et al. (2022)

Reusable modules encode expert knowledge

A module groups **analysis, sampling and transformations**.

Generic rules compose with **hardware-specific** optimisations such as Use-Tensor-Core.

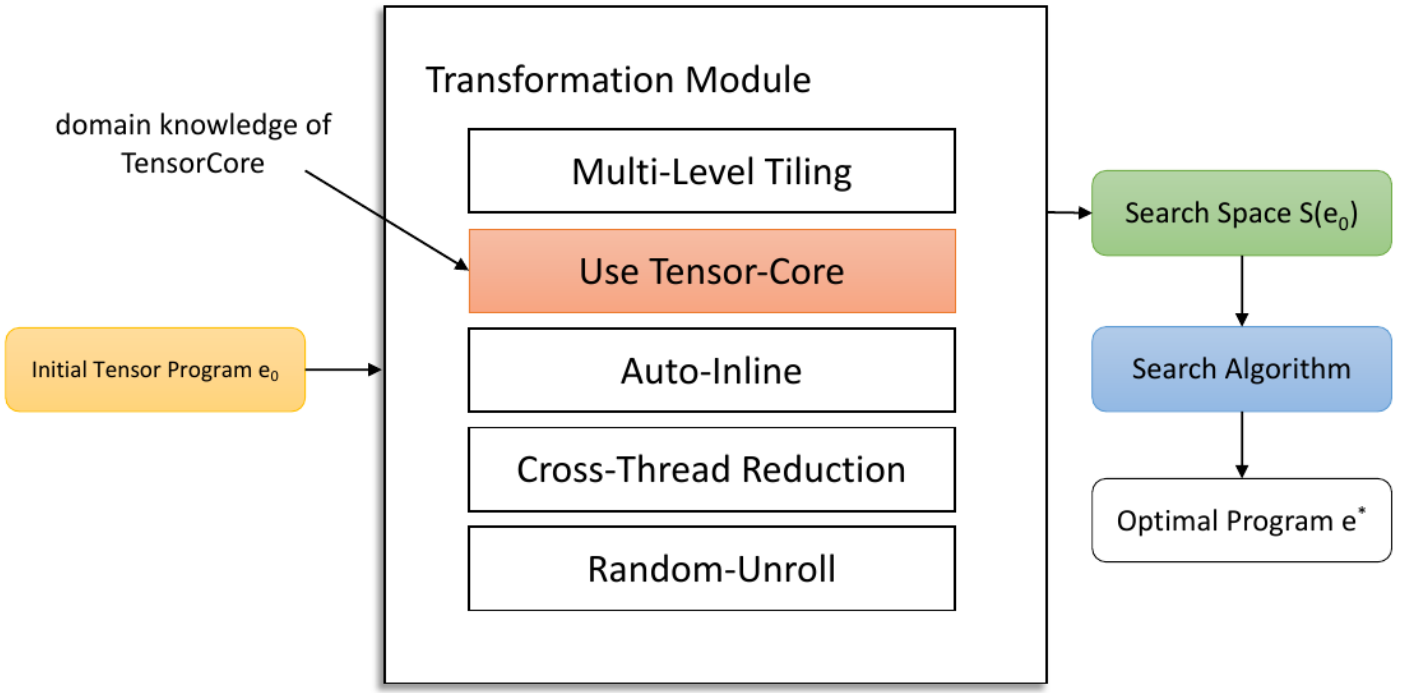


Figure 5, Shao et al. (2022)

The search algorithm consumes the space these modules define — construction and search are decoupled.

Search uses predictions and hardware feedback

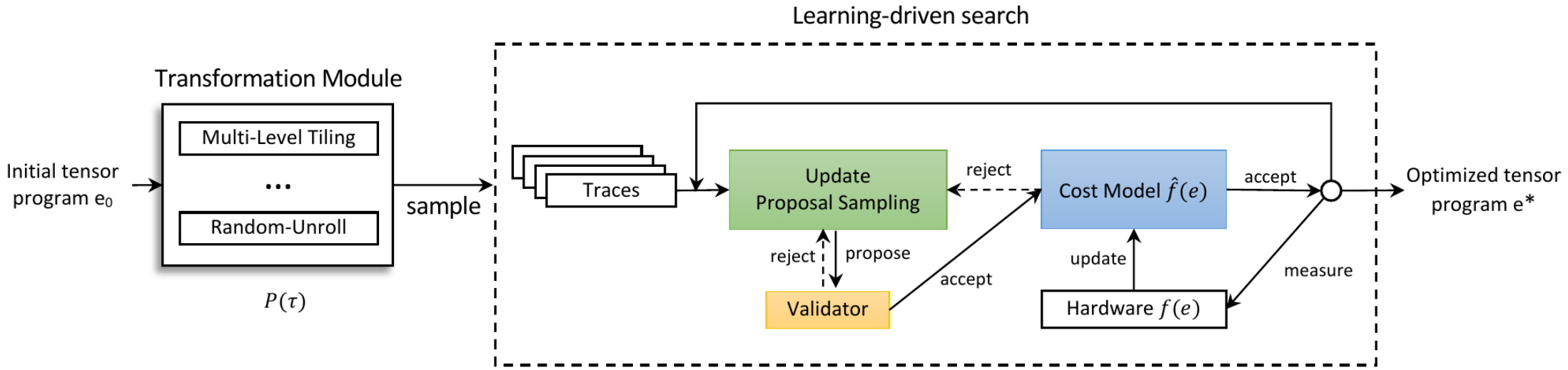


Figure 7, Shao et al. (2022)

CHEAP GUIDANCE

A learned cost model predicts performance — thousands of candidates, no hardware time.

EXPENSIVE FEEDBACK

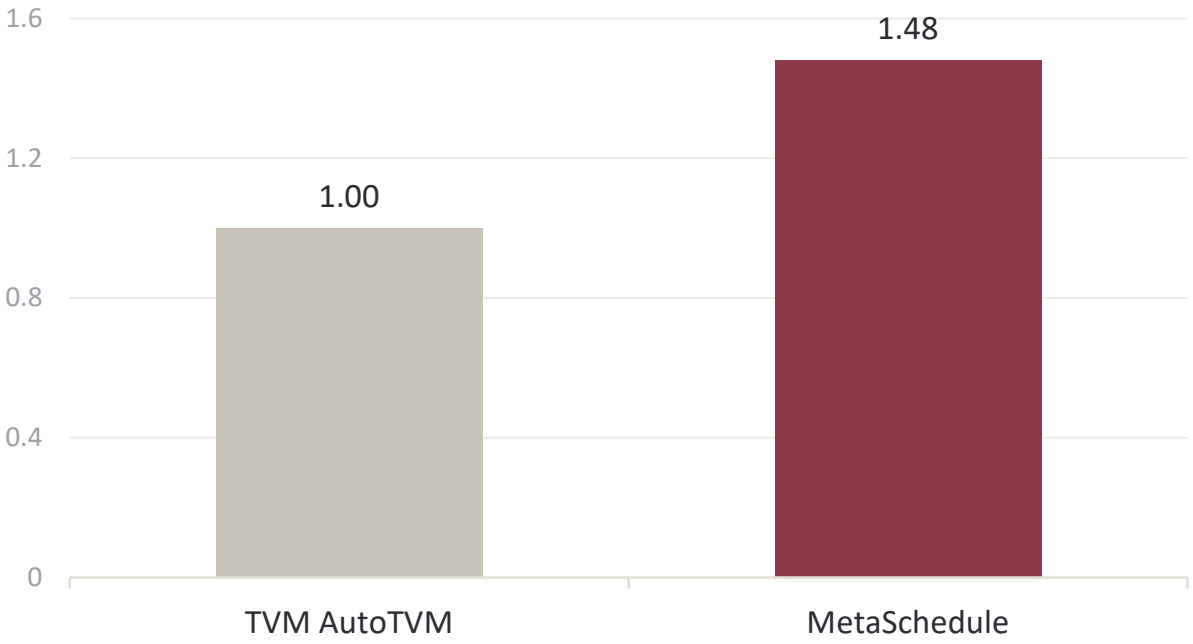
Selected candidates are measured on the target hardware; measurements update the model.

RESULTS

Evidence for an extensible search space

End-to-end performance is on par with TVM; extending the space pays off where it matters.

BERT-LARGE · ADDING THE USE-TENSOR-CORE MODULE



Relative performance · TVM AutoTVM = 1.00 · RTX 3070

+48%

speedup over TVM AutoTVM — specific to this experiment.

82 lines · 2 days

One graduate student wrote the Use-Tensor-Core module in Python (author-reported, single case).

COMPARISON

BOAT and MetaSchedule: where PP enters

	BOAT	MetaSchedule
Probabilistic program	Models system behaviour and performance	Constructs candidate program transformations
Expert knowledge	Structure of the performance model	Transformation modules and decision dependencies
Evaluation guidance	Bayesian inference and acquisition	Learned cost predictions and hardware feedback

Both use domain structure to spend an evaluation budget more effectively — *they differ in what the probabilistic program describes.*

CRITIQUE

Could the cost model overlook good candidates?

A POSSIBLE FEEDBACK LOOP

1 A fast candidate receives a high predicted latency.

↓

2 It is less likely to reach hardware measurement.

↓

3 The model receives no evidence that could correct its error.

WHAT I WOULD TEST

Reserve part of the same measurement budget for random or diverse candidates.

Repeat the search over multiple seeds.

Check whether random measurements find good programs the model had rated badly.

A risk, not a demonstrated failure — stochastic exploration may mitigate it.

Thanks for Listening