Tensor Program Optimization with Probabilistic Programs

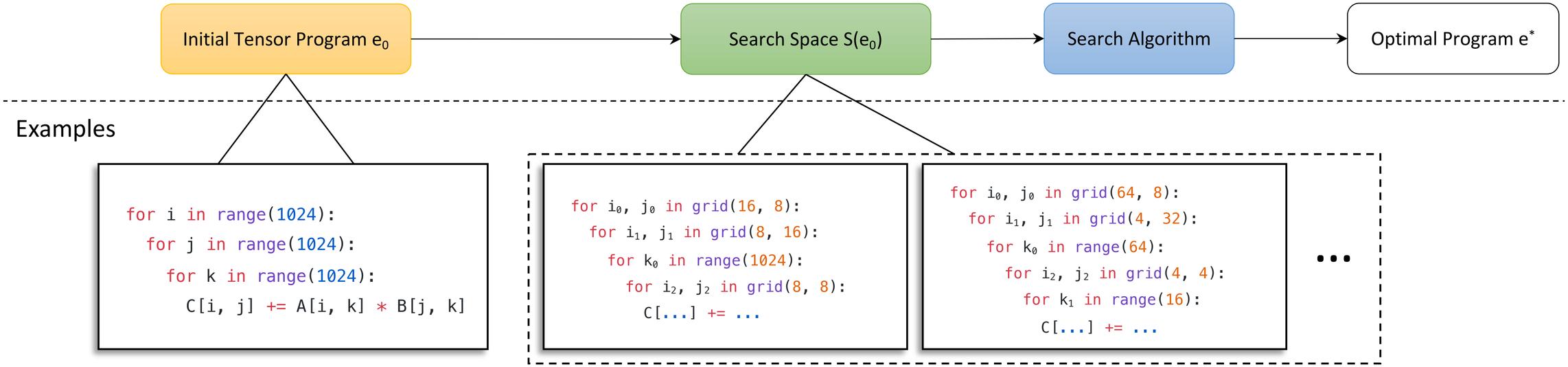
Junru Shao, Xiyou Zhou, Siyuan Feng, Bohan Hou, Ruihang Lai, Hongyi Jin, Wuwei Lin, Masahiro Masuda, Cody Hao Yu, Tianqi Chen

Shuntian Liu 9 Nov 2022

Code examples adapted from original paper

Optimising Tensor Program

Key elements in automatic tensor program optimization



Probabilistic Programming (PP)

- Probabilistic programming is **not** about writing software that behaves probabilistically.
- The key insight in PP is that statistical modelling can, when you do it enough, start to feel a lot like programming
- a probabilistic programming language is an ordinary programming language with `rand` and a great big pile of related tools that help you understand the program's statistical behaviour

Related work

- Halide (2013)
  - a language for fast, portable computation on images and tensors
- TVM (2018)
  - Template guided search space
- Ansor (2020)
  - Auto scheduler for subgraphs
- Difficult for expert to express domain knowledge in them

MetaSchedule, a domain-specific probabilistic programming language abstraction to construct a rich search space of tensor programs.

MetaSchedule, a domain-specific probabilistic programming language abstraction to construct a rich search space of tensor programs.

MetaSchedule Probabilistic language

```
# Dense:

for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = …
```

MetaSchedule Probabilistic language

```
# Dense:

for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```

MetaSchedule Probabilistic language

```
# Dense:

for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

Loop tiling

```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```

Loop fusion

MetaSchedule Probabilistic language

```
# Dense:

for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

Loop tiling

```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```


Loop fusion

MetaSchedule Probabilistic language

Tiling

```
# 1 Loop tiling for Dense
θ0, θ1 ~ Sample-Tile(i, parts=2)
θ2, θ3 ~ Sample-Tile(j, parts=2)
i0, i1 = Split(i, [θ0, θ1])
j0, j1 = Split(j, [θ2, θ3])
Reorder(i0, j0, i1, j1)
```

MetaSchedule Probabilistic language

```
# Dense:
for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

Loop tiling →

```
for i0, j0 in grid(θ0, θ2):
 for i1, j1 in grid(θ1, θ3):
  for k in range(16):
   C[…] += …
```

```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```

Loop fusion →

MetaSchedule Probabilistic language

```
# Dense:
for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

Loop tiling →

```
for i0, j0 in grid(θ0, θ2):
 for i1, j1 in grid(θ1, θ3):
  for k in range(16):
   C[…] += …
```

```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```

Loop fusion →

MetaSchedule Probabilistic language

Fusion

```
# 2 ReLU fusion
θReLU ~ Sample-Compute-Location(ReLU)
Compute-At(ReLU, θReLU)
```

MetaSchedule Probabilistic language

```
# Dense:
for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

Loop tiling →

```
# Fused Dense + ReLU
# θReLU : Deep fusion under j1
for i0, j0 in grid(θ0, θ2):
 for i1, j1 in grid(θ1, θ3):
  for k in range(16):
   C[…] += …
  for i', j' in grid(θ4, θ5):
   D[...] = ...
```

```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```

Loop fusion →

θReLU is j1

MetaSchedule Probabilistic language

```
# Dense:
for i in range(512):
 for j in range(256):
  for k in range(16):
   C[...] += ...
```

Loop tiling


```
# Fused Dense + ReLU
# θReLU : Shallow fusion under
j0
for i0, j0 in grid(θ0, θ2):
 for i1, j1 in grid(θ1, θ3):
  for k in range(16):
   C[…] += …
 for i', j' in grid(θ4, θ5):
  D[...] = ...
```

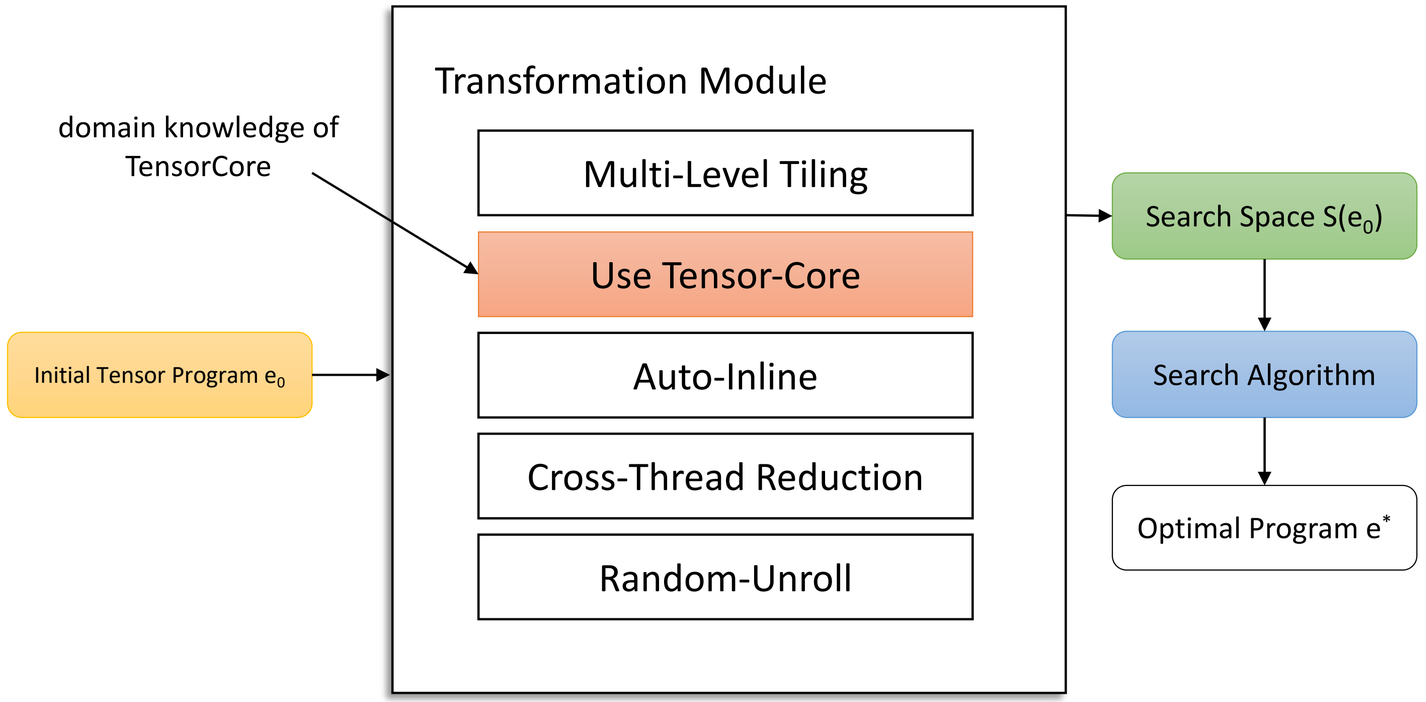
```
# ReLU:
for i' in range(512):
 for j' in range(256):
  D[...] = ...
```

Loop fusion

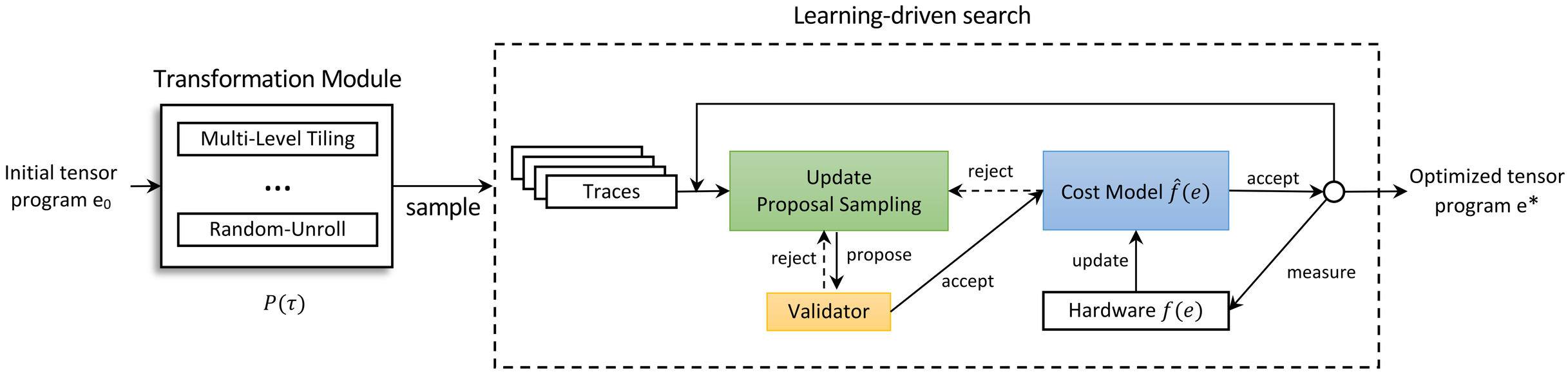
θReLU is j0

Modularity

- Previous program transformation modularised
- Modules are composable
- Pre-written modules
  - By domain specialist



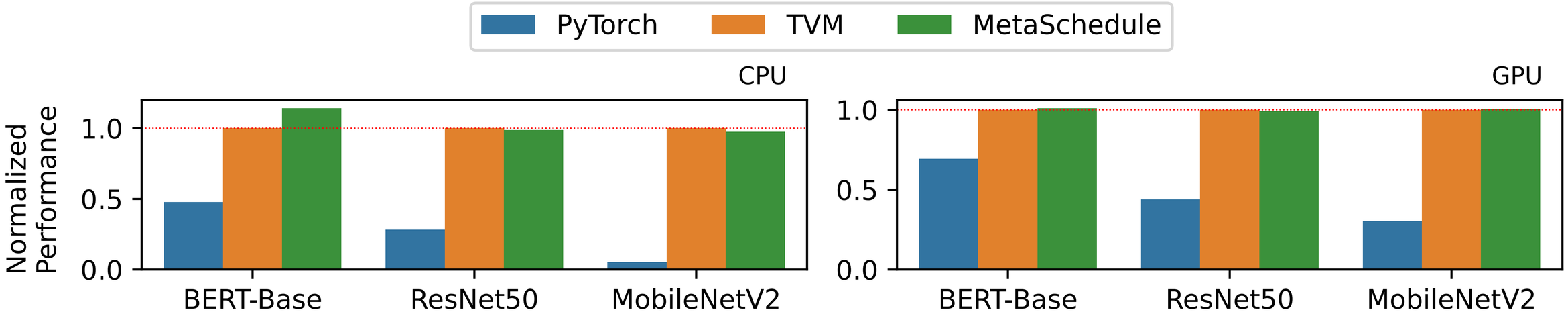
Learning driven approach



- End-to-end search
- Execution traces (validation)

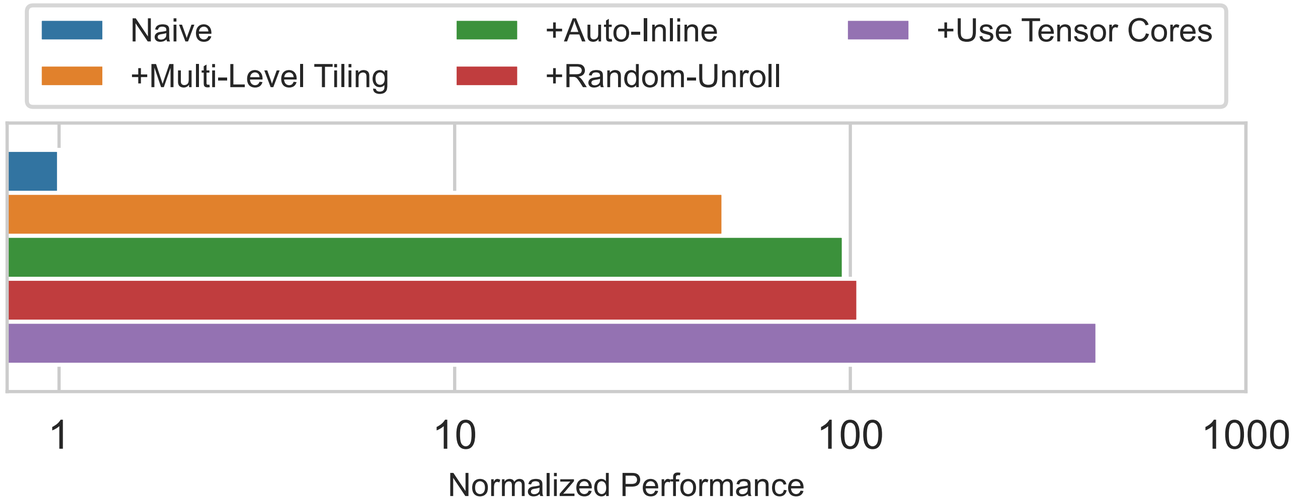
Performance

Optimising End-to-End Deep Learning Models

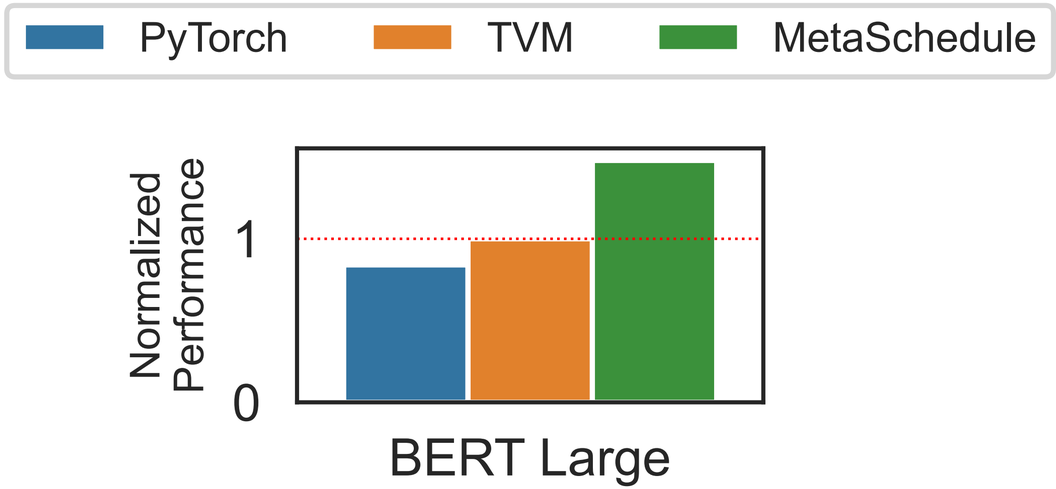


Performance

Composing modules



Performance with different search spaces.



BERT-Large Performance.

Thoughts & Critiques

Opinions are my own

- Why is probabilistic programming fit for search space construction
  - After all we can just do random sampling, which does not require PP
  - Guessing because it allows expressing of domain knowledge in a simple way
  - But this is more of PP’s contribution
- Evaluation comparing with hardware specific modules
  - Is this fair?
- Does not talk about searching time

Conclusions

- DSL (Probabilistic programming) for tensor program optimisation
- Rich search space construction
- Composable modules for program optimisation
- Learning driven approach based on search space specification

References

- Shao, Junru et al. “Tensor Program Optimization with Probabilistic Programs.” ArXiv abs/2205.13603 (2022): n. pag.
- Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: generating high-performance tensor programs for deep learning. In Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20). USENIX Association, USA, Article 49, 863–879.
- Ragan-Kelley, Jonathan & Barnes, Connelly & Adams, Andrew & Paris, Sylvain & Durand, Frédo & Amarasinghe, Saman. (2013). Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. ACM SIGPLAN Notices. 48. 519-530. 10.1145/2499370.2462176.
- Yan, Eddie Q. et al. “USING AUTOTVM TO AUTOMATICALLY GENERATE DEEP LEARNING LIBRARIES FOR MOBILE DEVICES.” (2019).
- Adrian Sampson, Probabilistic programming, url: http://adriansampson.net/doc/ppl.html, retrieved 26 Oct 2022