AutoPhase: Juggling HLS Phase Orderings in Random Forests with Deep Reinforcement Learning

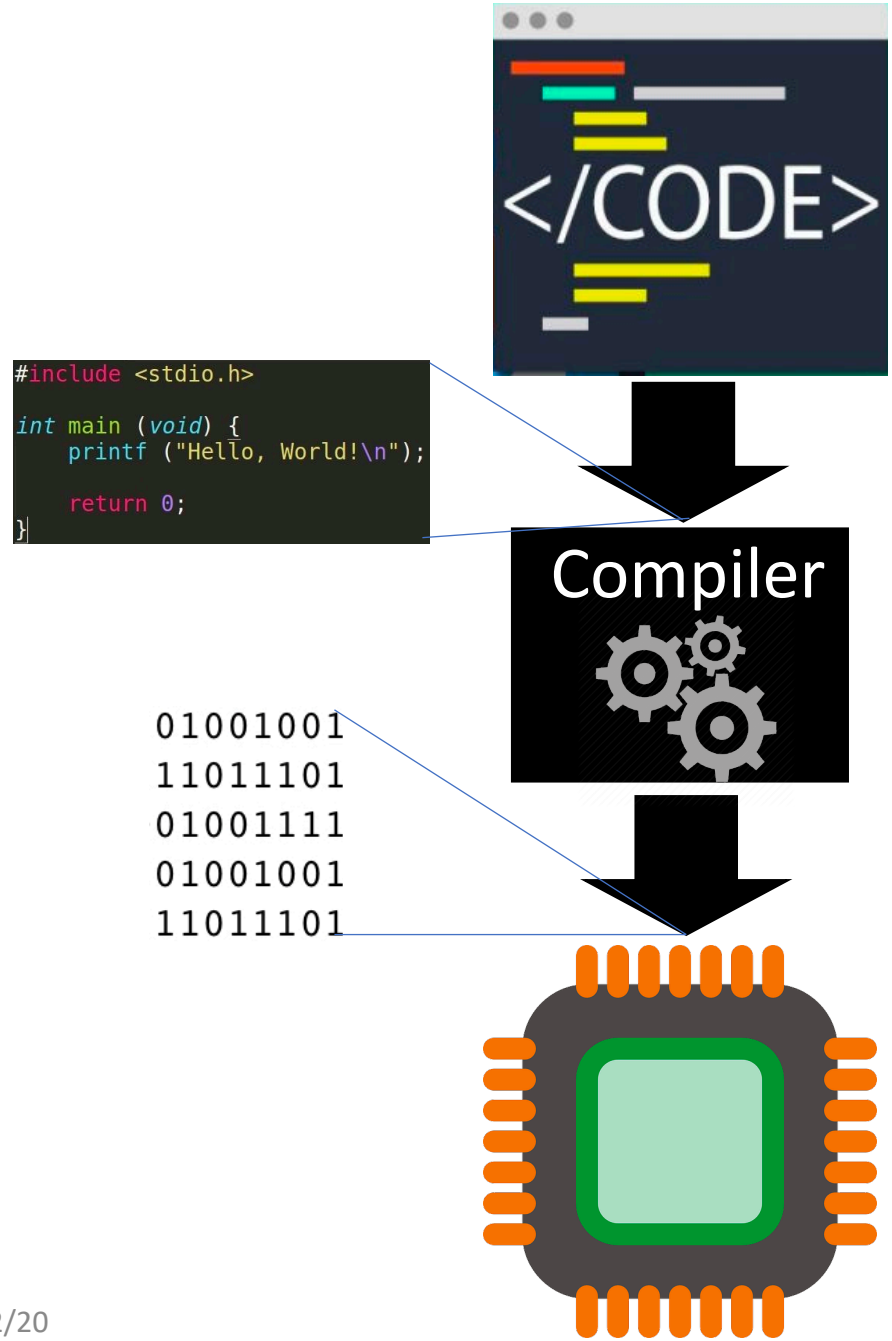
Ameer Haj-Ali*, Qijing Huang*,
W. Moses, J. Xiang, K. Asanovic, J. Wawrzynek, I. Stoica

***Equal Contribution**

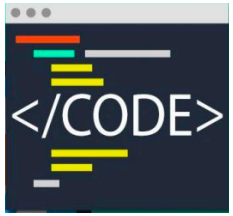
https://github.com/ucb-bar/autophase

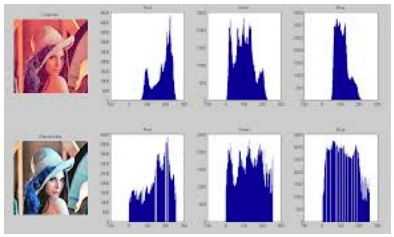
Toward automatic and intelligent optimization of off-the-shelf code

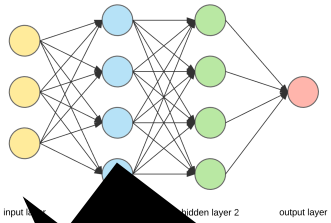
Berkeley Architecture Research



For years, software engineers used **heuristics** and **hand engineering** to optimize compilation!







Plethora of new applications

This is confusing …

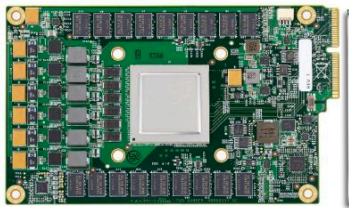
Compiler

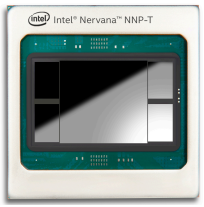
Need a way to compile applications to rapidly changing hardware and software without compromising performance!

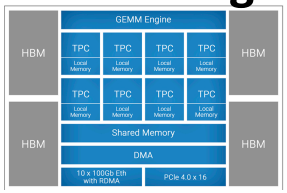
Application-Specific Integrated Circuits

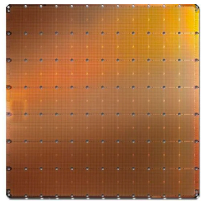
New Technologies

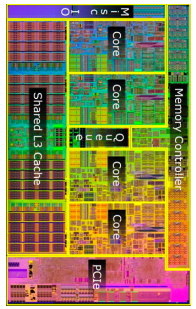
Distributed Computing

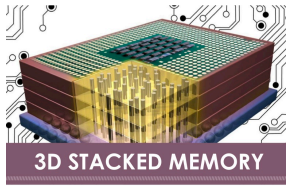


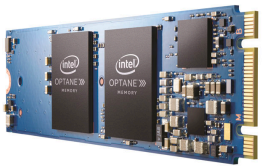


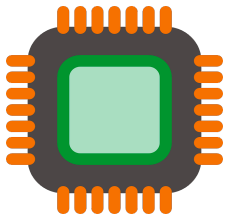


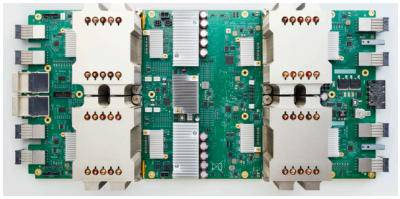




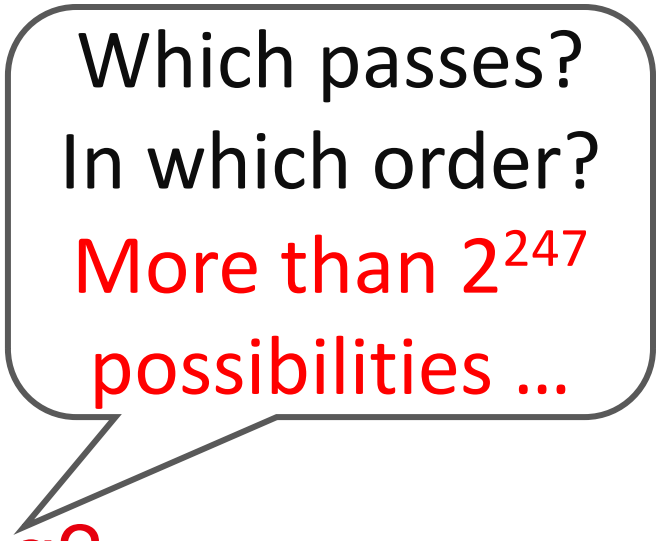








Compiler Challenge Example: Phase Ordering (NP-Hard)



clang program.c -flag1 -flag2 …

Touati et al., "On the decidability of phase ordering problem in optimizing compilation," *CD-ROM* 2006

Example – Normalizing a Vector

```
for(int i=0; i<n; i++) {
    out[i] = in[i] / mag(in, n);
}
```

Pass #1: -licm (loop-invariant code motion)

```
double precompute = mag(in, n);
for(int i=0; i<n; i++) {
    out[i] = in[i] / precompute;
}
```

Pass #2: -inline

```
double precompute, sum = 0;
for(int i=0; i<n; i++){
    sum += A[i] * A[i];
}
precompute = sqrt(sum);
for(int i=0; i<n; i++) {
    out[i] = in[i] / precompute;
}
```

$\Theta(n)$

Example – Normalizing a Vector

```
for(int i=0; i<n; i++) {
    out[i] = in[i] / mag(in, n);
}
```

Pass #1: -inline

```
for(int i=0; i<n; i++) {
    double sum = 0;
    for(int j=0; j<n; j++){
        sum += A[j] * A[j];
    }
    out[i] = in[i] / sqrt(sum);
```

Pass #2: -licm (loop-invariant code motion)

```
double sum;
for(int i=0; i<n; i++) {
    sum = 0;
    for(int j=0; j<n; j++){
        sum += A[j] * A[j];
    }
    out[i] = in[i] / sqrt(sum);}
```

$\cancel{\Theta(n)}$

$\Theta(n^2)$

Related Works to Remedy

- Heuristics, Greedy algorithms, Genetic algorithms, Multi-armed Bandit
  - Hand engineered features
  - Error prone
  - Requires retraining for every program
  - Nothing is being “learned”
  - Cannot generalize
- Supervised Learning
  - Requires hard/impossible-to-find labels

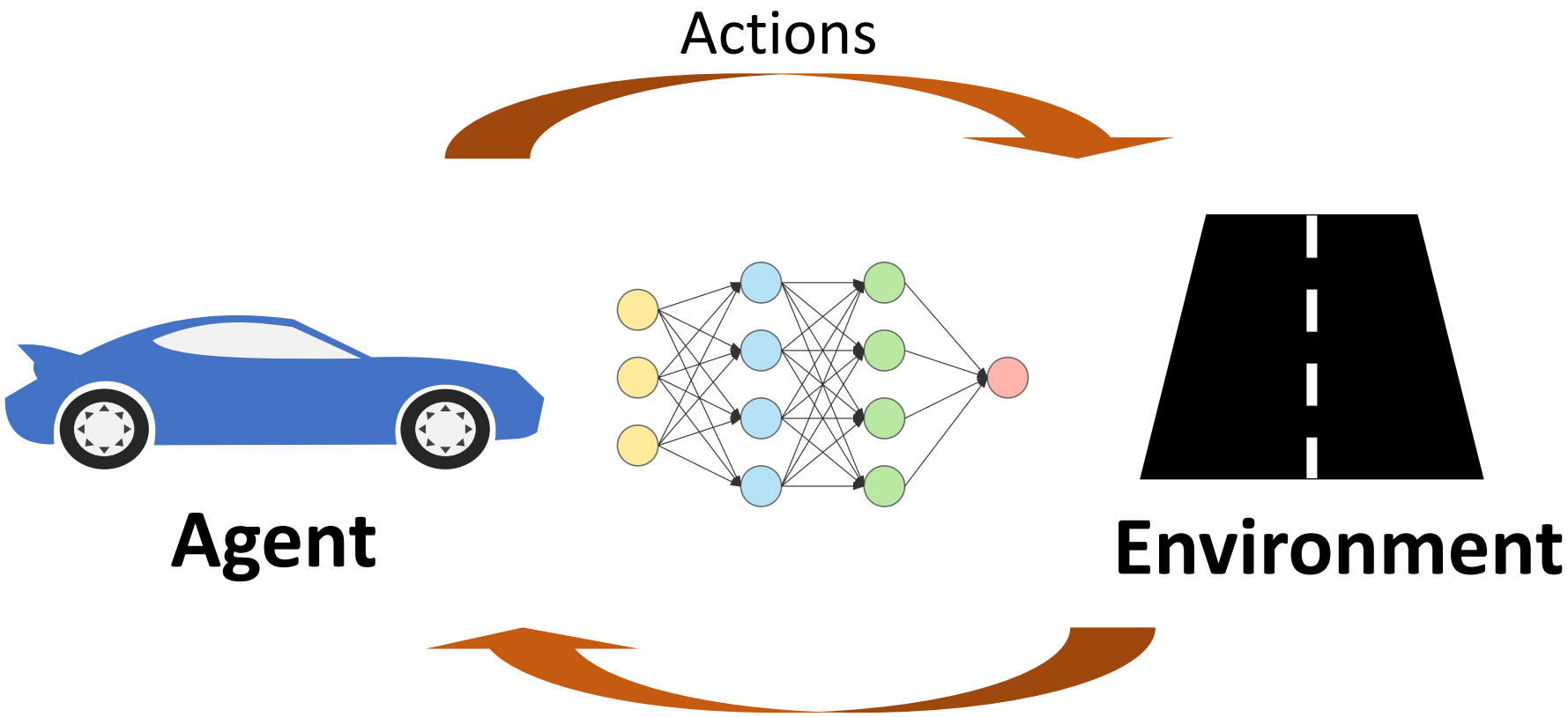
- C. Cummins *et al.,* “End-to-End Deep Learning of Optimization Heuristics,“ PACT 2017.
- J. Ansel *et al.,* "OpenTuner: An Extensible Framework for Program Autotuning," PACT 2014.
- Z. Wang *et al.*, "Machine Learning in Compiler Optimization." Proceedings of the IEEE 2018
- **Q. Huang** *et al.*, “The effect of compiler optimizations on high-level synthesis-generated hardware,” TRETS 2015.
- G. Fursin *et al.,* "MILEPOST GCC: machine learning based research compiler." 2008.
- M. Stephenson et al. "Meta optimization: Improving compiler heuristics with machine learning," ACM sigplan notices 2013
- F. Agakov et al. "Using machine learning to focus iterative optimization," CGO 2006.
- S. Kulkarni, "Mitigating the compiler optimization phase-ordering problem using machine learning," OOPSLA 2012.

The Ultimate Solution

- Handles complex search spaces and decisions
- Automatic
- Minim
- Do retraining (generalizes)
- Feedback/experience driven
- Does not require labels

Deep Reinforcement Learning

Deep Reinforcement Learning



Example

Decision(**Action**): Gas

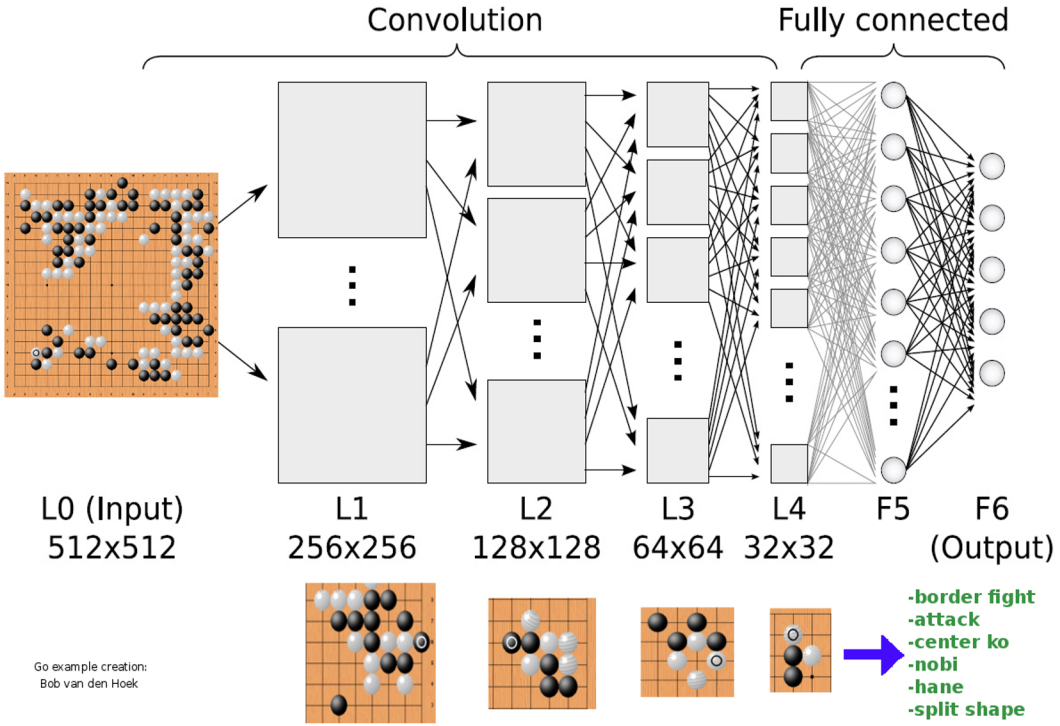
Decision(**Action**): Brake

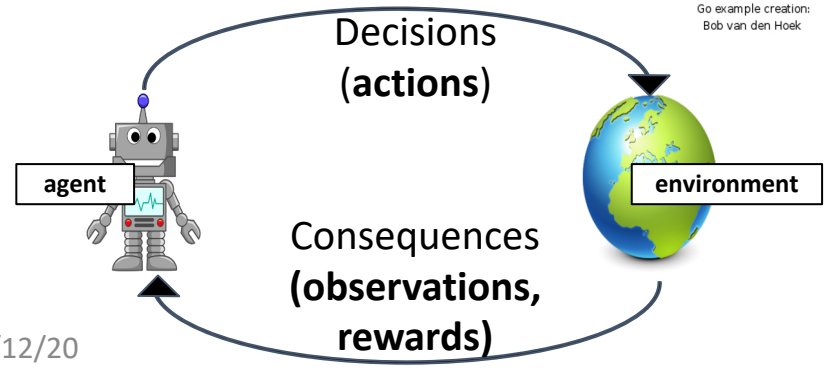
Reward

Go as a Deep Reinforcement Learning Problem

AlphaGo (Silver et al. 2016)

- **Observations**:
  - board state
- **Actions**:
  - where to place the stones
- **Rewards**:
  - 1 if win
  - 0 otherwise

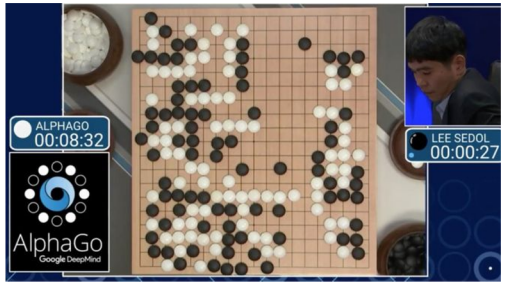


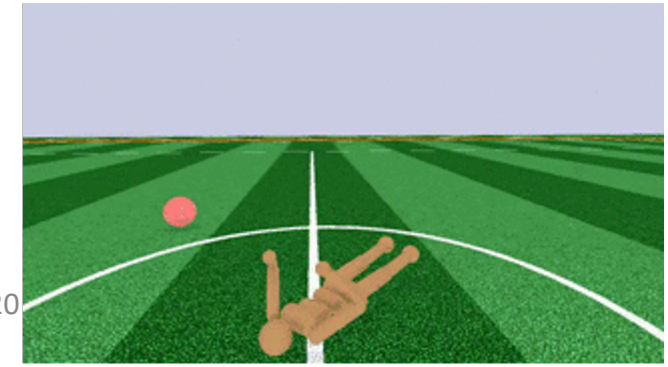


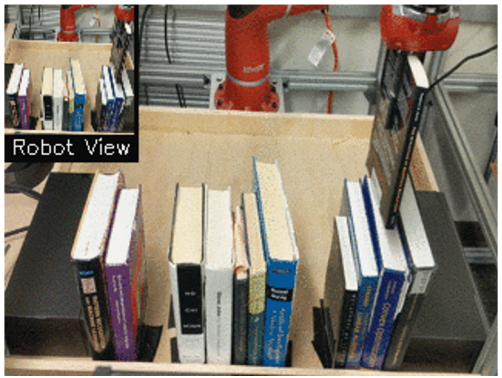
Go example creation:
Bob van den Hoek

Growing Number of RL Applications

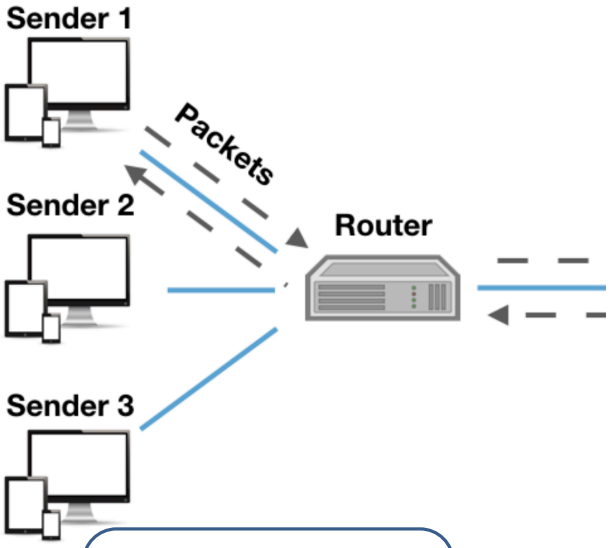
Robotics | **Industrial Control** | **Advertising** | **System Optimization** | **Finance**

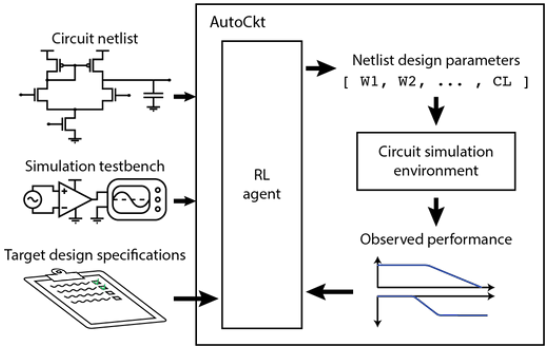






Deep RL is Used Everywhere in Systems





Which passes?
In which order?

gcc program.c -flag1 -flag2 …

- **Ameer Haj-Ali**, *et al*, "A View on Deep Reinforcement Learning in System Optimization", (submitted). *Available on arXiv*.
- **Qijing Huang*, Ameer Haj-Ali***, *et al.*, "AutoPhase: Compiler Phase-Ordering for High-Level Synthesis with Deep RL," *FCCM* 2019.
- Keertana Settaluri, **Ameer Haj-Ali**, **Qijing Huang**, *et al.*, "AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs," *DATE 2020*.
- Bin Li, …, **Ameer Haj-Ali**, *et al.,* "RLDRM: Closed Loop Dynamic Cache Allocation with Deep RL for Network Function Virtualization*", NetSoft 20'.*
- ***Ameer Haj-Ali**, et al.,* "NeuroVectorizer: End-to-End Vectorization with Deep Reinforcement Learning," *CGO 2020*, to appear.
- Jay, Nathan, et al. "A Deep Reinforcement Learning Perspective on Internet Congestion Control," *ICML* 2019.
- Mao, Hongzi, et al. "Resource management with deep reinforcement learning." *HotNets 2016*.
- Krishnan, Sanjay, et al. "Learning to optimize join queries with deep reinforcement learning." *arXiv preprint* 2018.
- Liang, Eric, et al., "Neural Packet Classification." *arXiv preprint* 2019.
- Xu, Cheng-Zhong, et al., "URL: A unified reinforcement learning approach for autonomic cloud management.", *JPDC* 2012.

AutoPhase: Juggling HLS Phase Orderings in Random Forests with Deep Reinforcement Learning

- A framework that uses Deep RL to address the phase-ordering challenge
- An importance analysis on the features
- Can generalize to different workloads
- Achieves 28% performance speedup over –O3

AutoPhase: Phase Ordering with Deep RL

Decision (**Action**):
- Gas
- Brake
- Steering

Destination

Fee (**Reward**): $100

GPS Location (**State**):
37.87631055, -122.2388

AutoPhase: Phase Ordering with Deep RL

Decision (**Action**):
- Gas
- Brake
- Steering

Destination

Fee (**Reward**): $100

GPS Location (**State**):
37.87631055, -122.2388

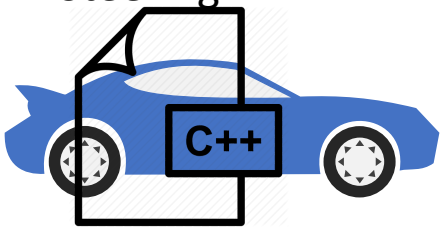
A Tree

Fee (**Reward**): -$500

GPS Location (**State**):
-30.43131384, 73.1693

AutoPhase: Phase Ordering with Deep RL

Decision (**Action**):
- Gas
- Brake
- Steering



Source Code

Action: Optimization Pass #
(e.g. -licm, -inline, -sroa, mem2reg)

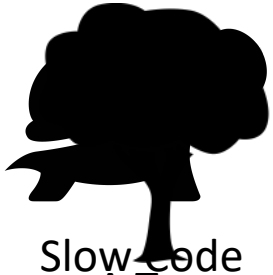
Destination
Fast Code

Fee (**Reward**): $100

GPS Location (**State**):
37.87631055, -122.2388

Reward:
Cycle Count Improvement

State: a) Program Features
b) Applied Passes



Slow Code
A Tree

Fee (**Reward**): -$500

GPS Location (**State**):
-30.43131384, 73.1693

Generating the States

1. **Program Features:**
   - 56 static features (# of instructions, # of loops, ...)
   - An LLVM analysis pass is built

2. **Histogram of Applied Passes:**
   - 45 optimization passes
   - Once pass ***i*** is applied, ***Histogram[i]++***
     (***Histogram:*** a vector of length 45, ***i:*** index of a pass)

Generating the Rewards

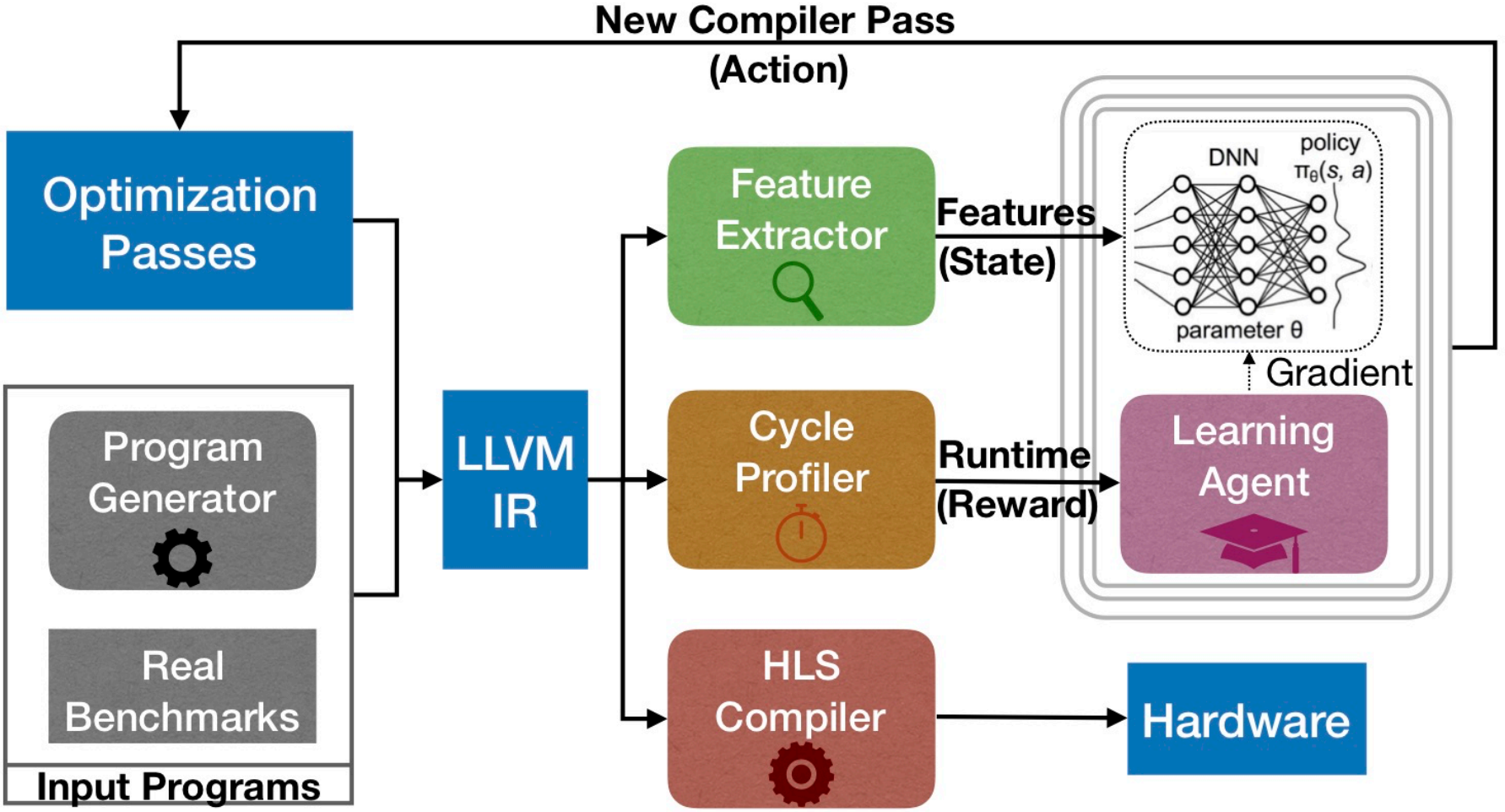
- Reward definition:

  cycles before a pass is applied

  – cycles after a pass is applied

- LegUp estimates the circuit cycles from C
  - Leveraging SW run information
  - 20× faster than RTL simulation
  - 0.48% error rate on our benchmarks

AutoPhase Framework



AutoPhase Example

Action: -licm

Before: 100 Cycles

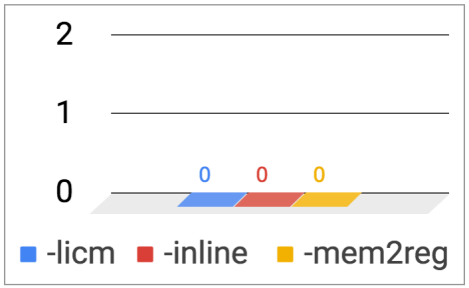
After: 100 Cycles

Rewards: 100 – 100 = 0

State:

a) # loops = 3, # insn = 20

b)



Trajectory Length = 3

Pass Sequence:

-licm

AutoPhase Example

Action: -mem2reg

Before: 100 Cycles

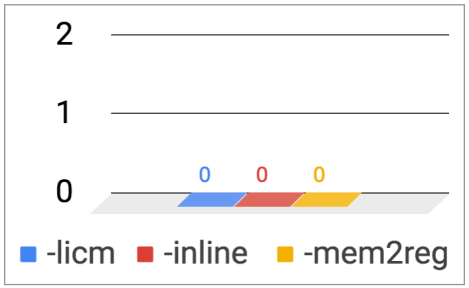
After: 90 Cycles

Rewards: 100 – 90 = 10

State:

a) # loops = 3, # insn = 20

b)



Trajectory Length = 3

Pass Sequence:

-licm

-mem2reg

AutoPhase Example

Action: -inline

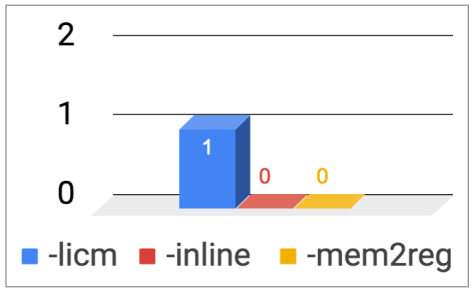
Before: 99 Cycles

After: 66 Cycles

Rewards: 99 – 66 = 33

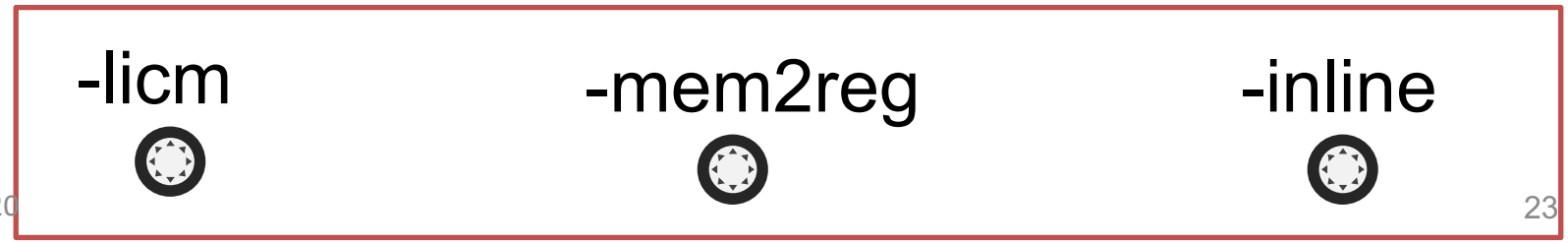
State:

a) # loops = 3, # insn = 20

b) 

Trajectory Length = 3

Pass Sequence:



AutoPhase Example

Action:

Before: 00 Cycles

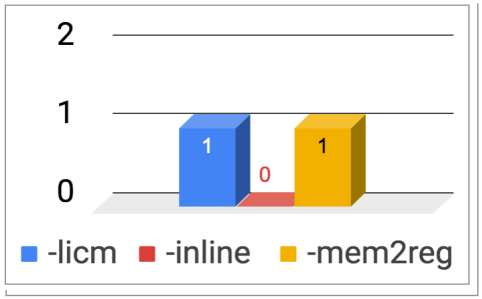
After: 56 Cycles

Rewards: 00 – 56 = 10

State:

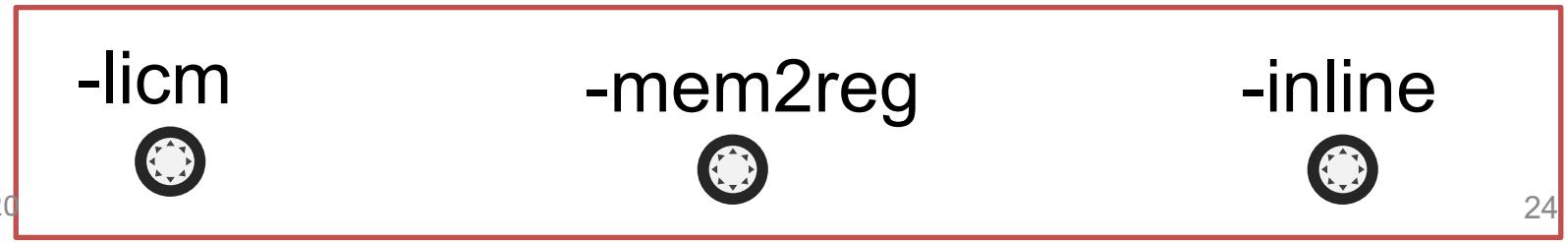
a) # loops = 3, # insn = 10

b)



Trajectory Length = 3

Pass Sequence:



Evaluated Algorithms

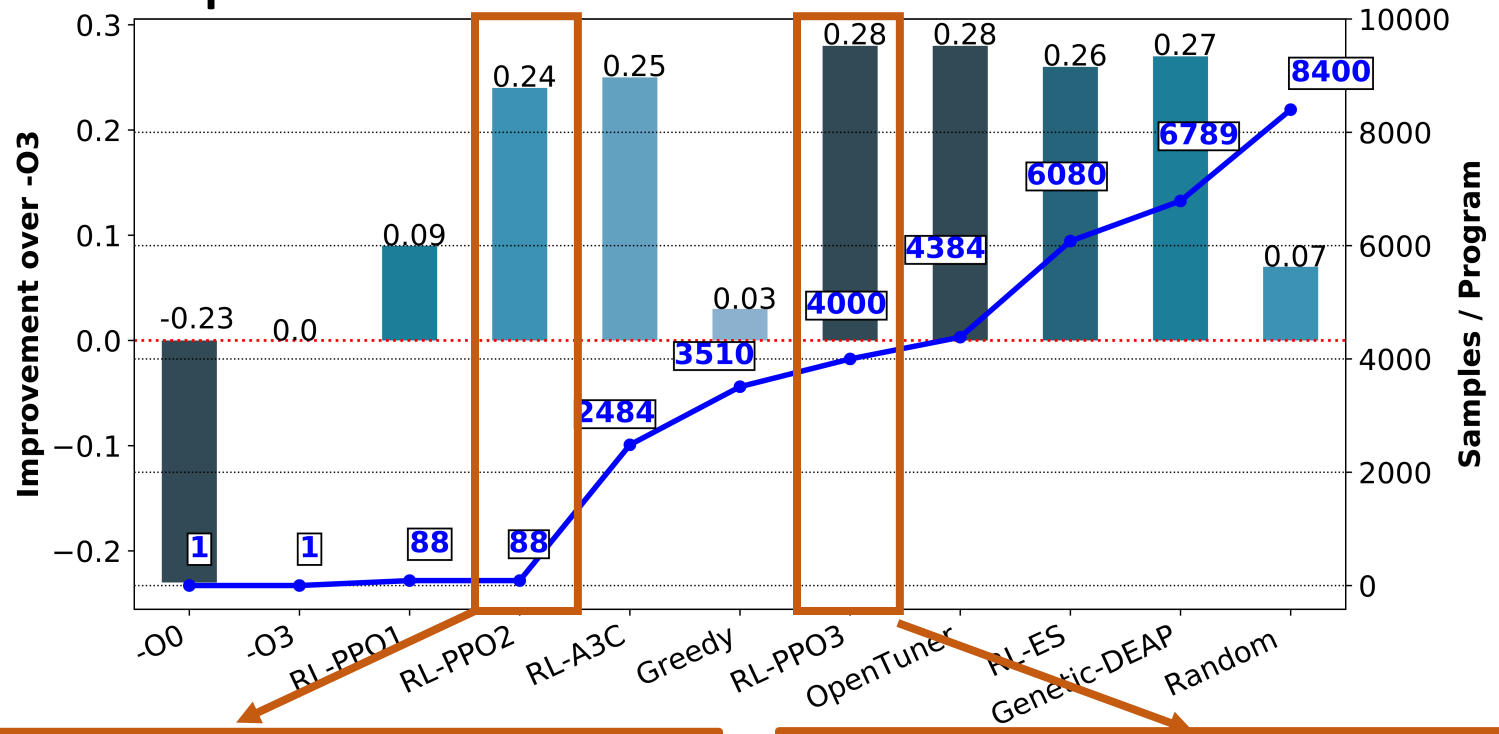
- Random Search
- Genetic Algorithms
- OpenTuner – Multiarmed Bandit/PSO
- Greedy Algorithms/Beam Search
- Deep Reinforcement Learning
  - Proximal Policy Optimization (PPO)
  - Asynchronous Actor-Critic Agents (A3C)
  - Evolutionary Strategies (ES)

Methodology

- 9 benchmarks from CHStone
- A3C, PPO, ES vs. OpenTuner, Genetic, Greedy, Random, -O3
- The network:
  - 256 × 256 FC
  - Trajectory length: 45
- RLlib/Ray, 4-core Intel i7-4765T CPU with a Tesla K20c GPU for training and inference

Performance (Execution Time)

- Fewer samples required
- 28% improvement over -O3



- Uses only **Histogram of Applied Passes**
- Compiles and runs **once** per trajectory

- Uses **Program Features** and **Histogram of Applied Passes**

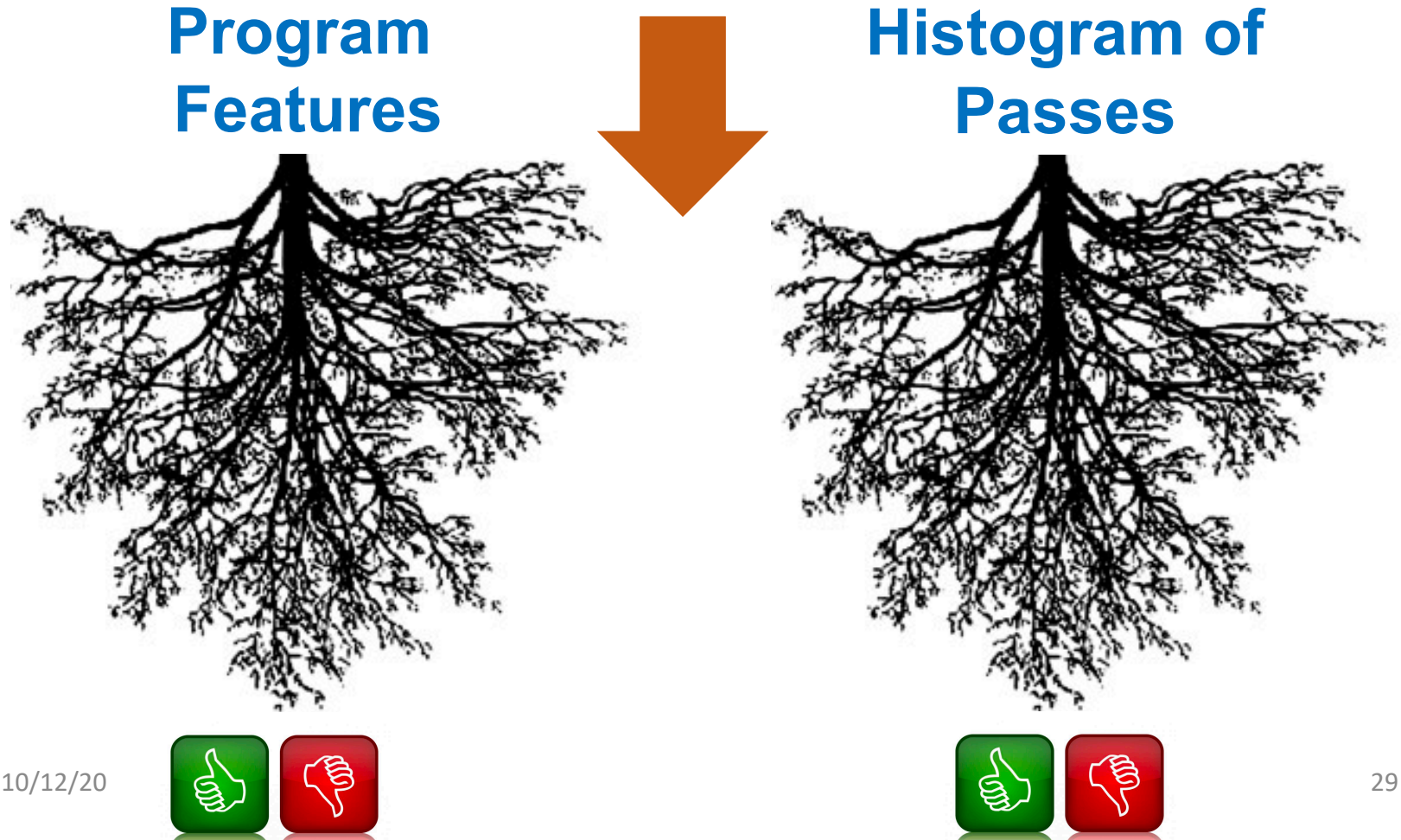
*** Results are normalized to –O3**

Can We Generalize?

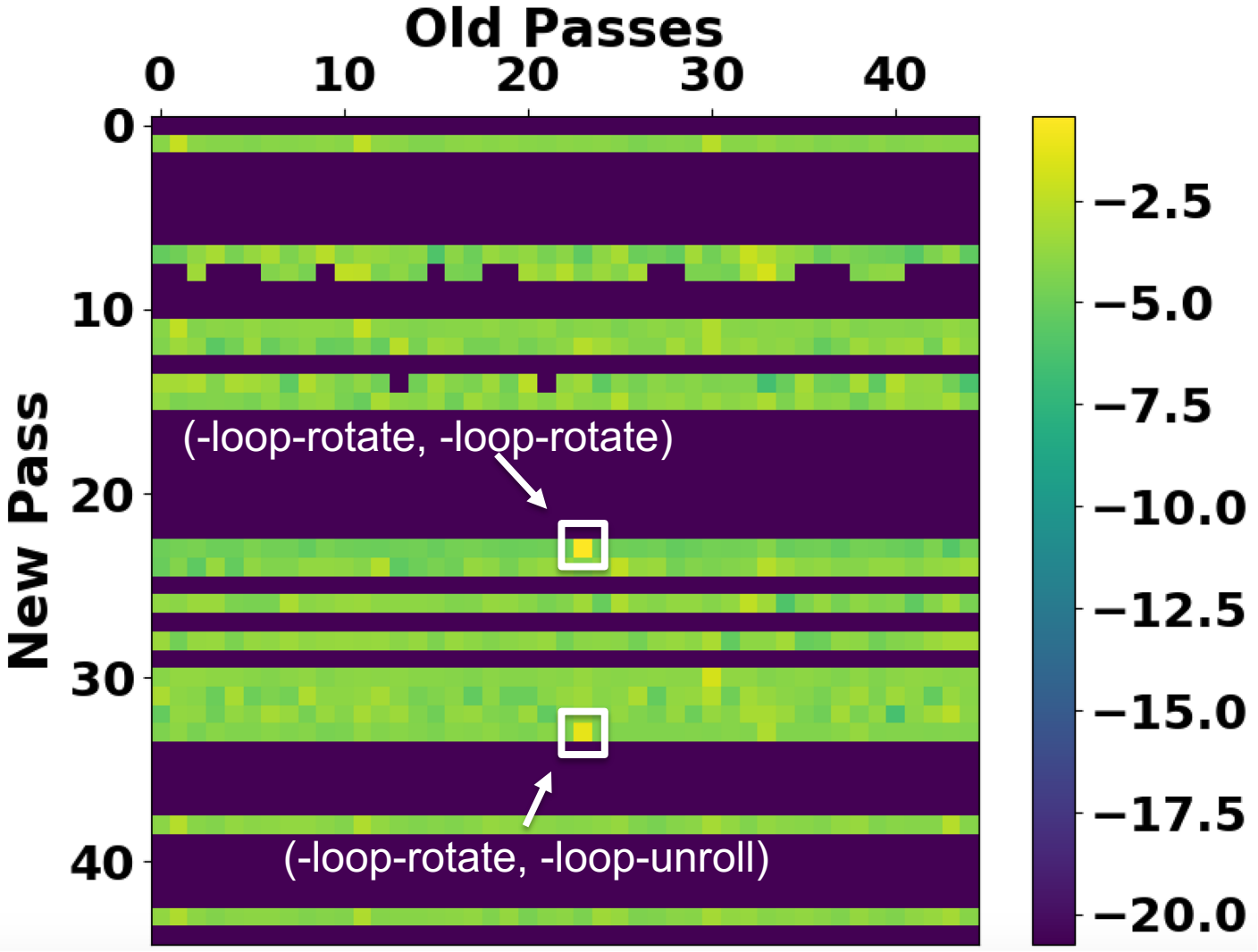
- Can we train a network and use it to predict the optimal sequence of any program in a single compilation? **Very Difficult!**
  - More than 45^{45} possible phase orderings
  - Limited observations
  - Limited data
  - Decisions are not explainable

Understanding the Correlations between Actions and States

- Two random forests for each optimization pass



Importance Analysis



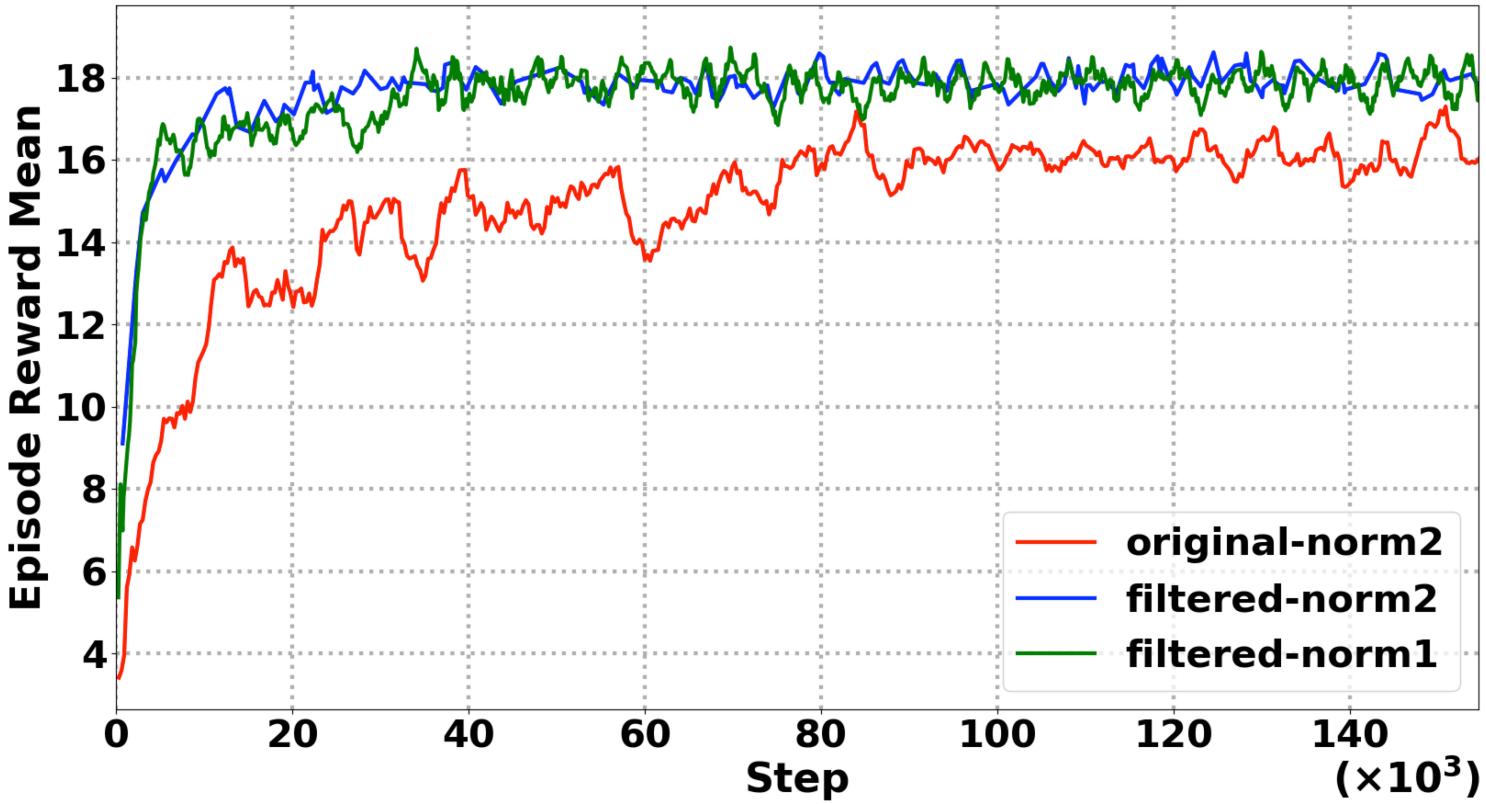
Normalization Schemes

1. Logarithm of program features and rewards:
   - Reduces the magnitude of features/rewards
   - The neural network learns to correlate the products of features instead of a linear combination of them
     - $w_1 \log(o_{f_1}) + w_2 \log(o_{f_2}) = \log(o_{f_1}^{w_1} \cdot o_{f_2}^{w_2})$

2. Dividing the program feature values by the total number of instructions

Training Results

- On 100 randomly-generated programs



Training Curve

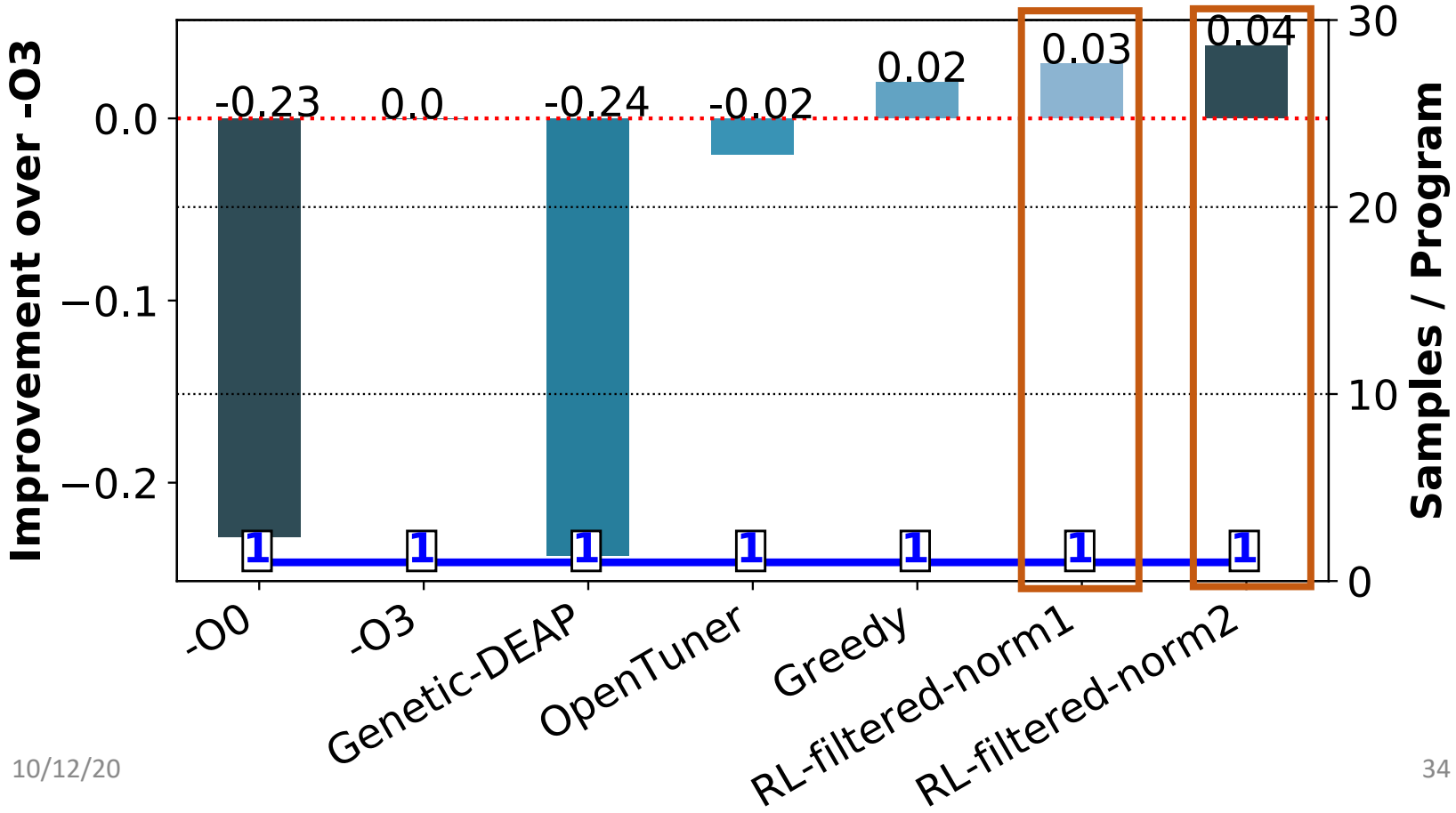
Generalization Results

Inference

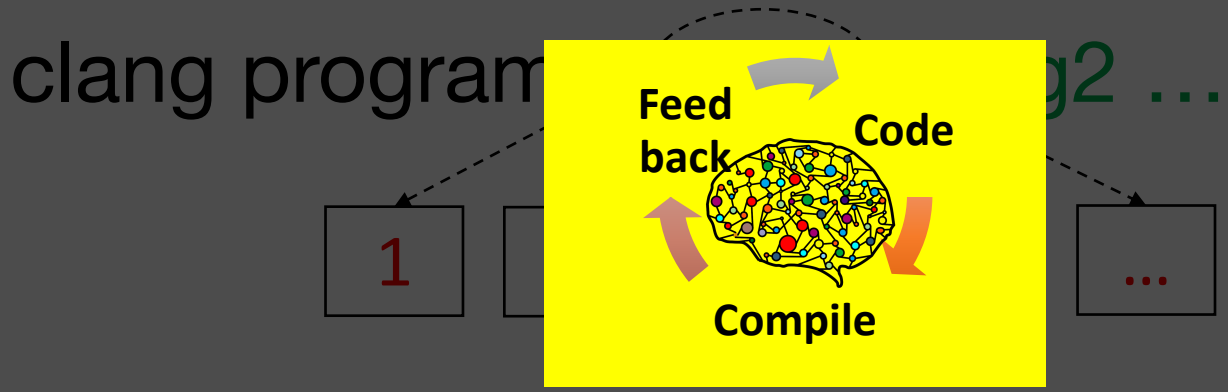
- 13,000 randomly-generated programs
  - 6% better than –O3
- CHStone benchmarks
  - different from the randomly-generated programs

Generalization Results

- ~4% improvement over -O3 on CHStone
- -O3 fails to compile the gsm benchmark



It is NeuroVectorizer: End-to-End Vectorization with Deep Reinforcement Learning



Ameer Haj-Ali, *et al.*, "NeuroVectorizer: End-to-End Vectorization with Deep Reinforcement Learning," CGO 2020.

https://github.com/intel/neuro-vectorizer

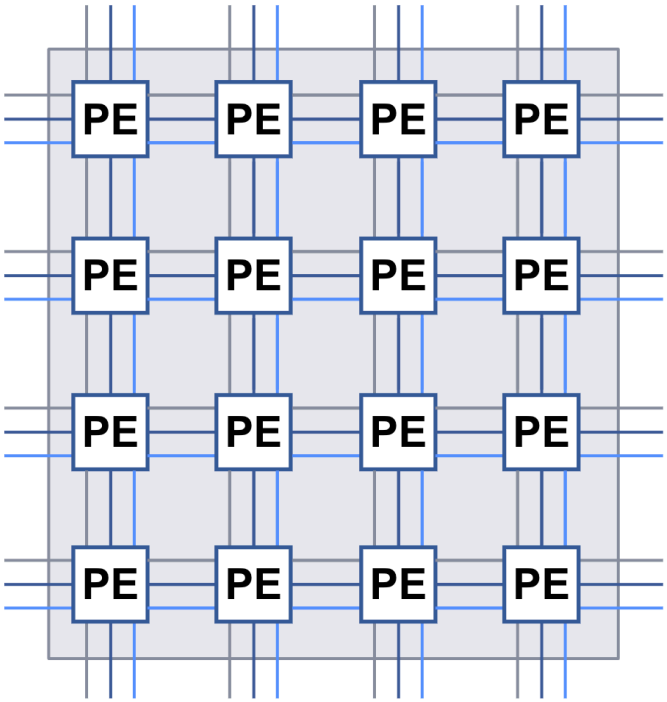
Ongoing Work

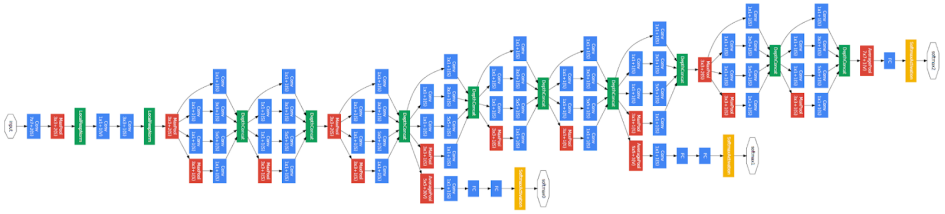
More optimizations on CPUs/GPUs:

- Scheduling
- Multithreading
- Loop transformations
  - Vectorization
  - Parallelization
  - Tiling
  - Interchange
  - Fusion

Ongoing Work

NoC Scheduling

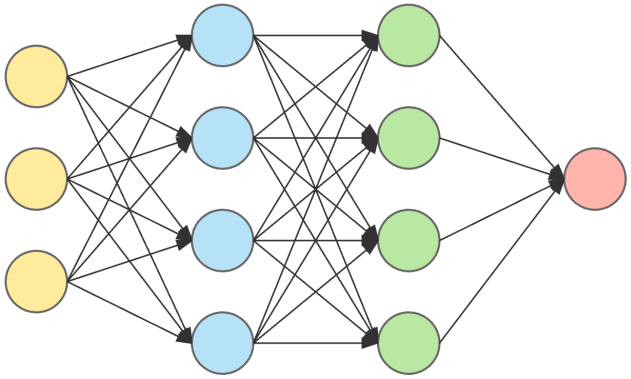


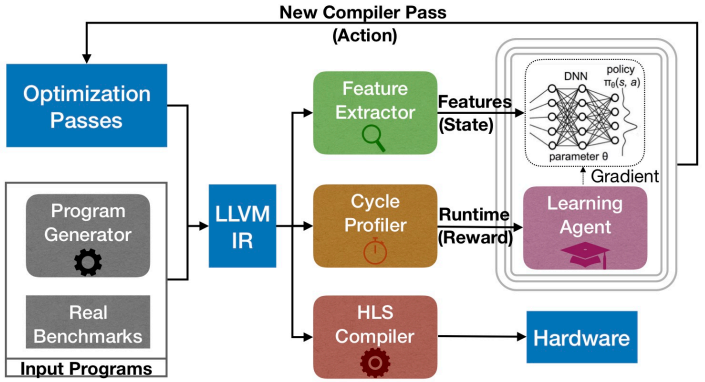


1. **Temporal Organization**
   - Tasks to execute next
2. **Spatial Organization**
   - Allocation & Placement

Conclusions

- RL shows potential to tackle the phase ordering problem with fewer samples
- AutoPhase achieves 28% circuit speedup over -O3
- AutoPhase generalizes to real benchmarks from training on random programs
- AutoPhase is compatible with SW optimization

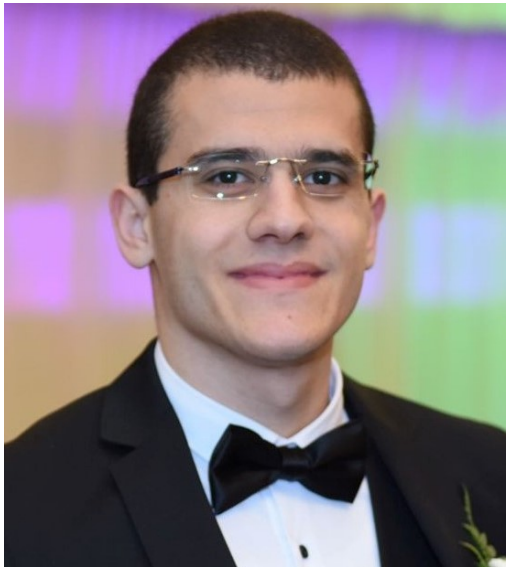




Access to code: https://github.com/ucb-bar/autophase

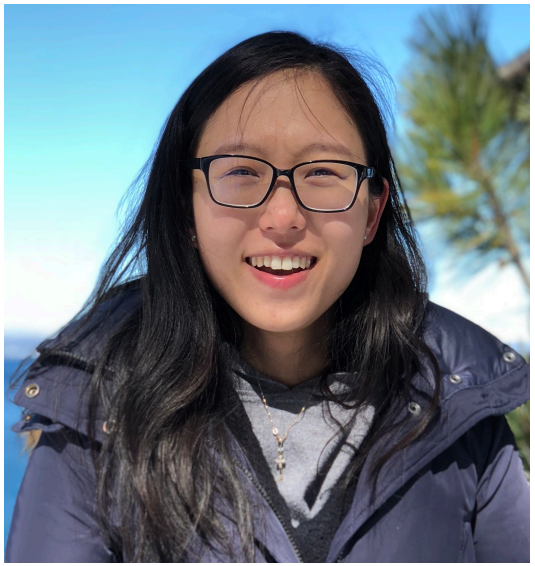
Thanks!

ameerh@berkeley.edu



ameerhajali.com

qijing.huang@berkeley.edu



ADEPT

Berkeley Architecture Research

riselab UC Berkeley