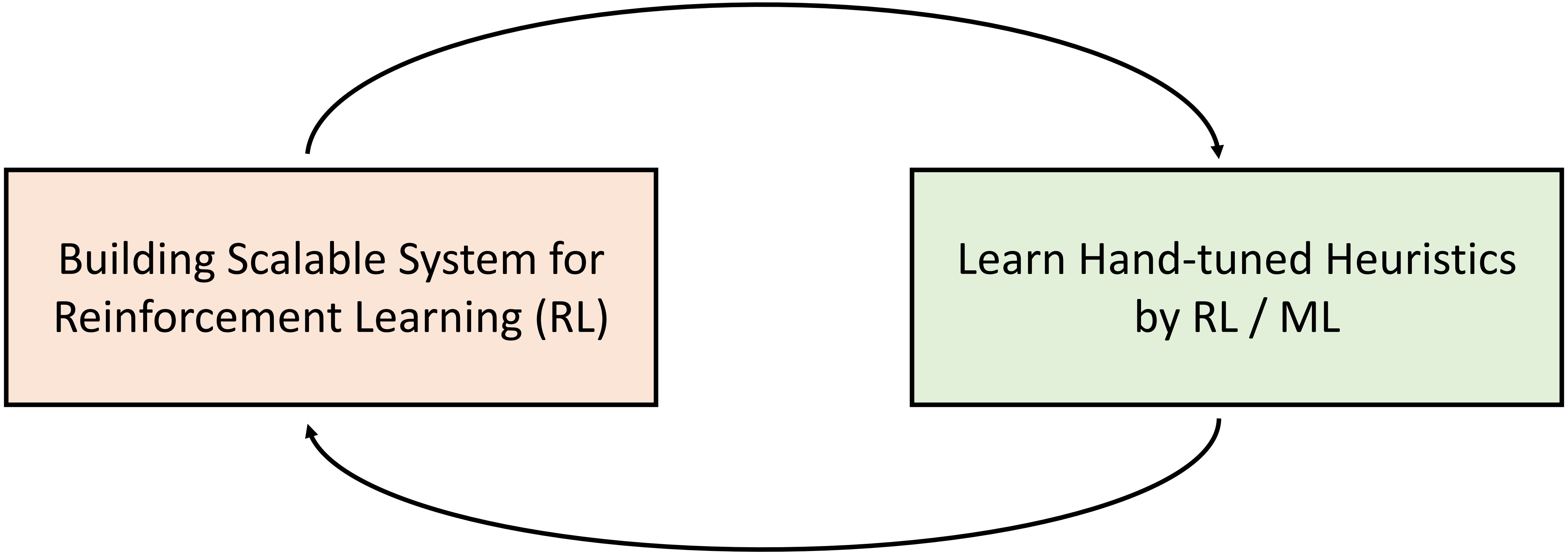
Building Scalable Systems for Reinforcement Learning and Using Reinforcement Learning for Better Systems

Presented by Yuandong Tian

Research Scientist and Manager
Facebook AI Research

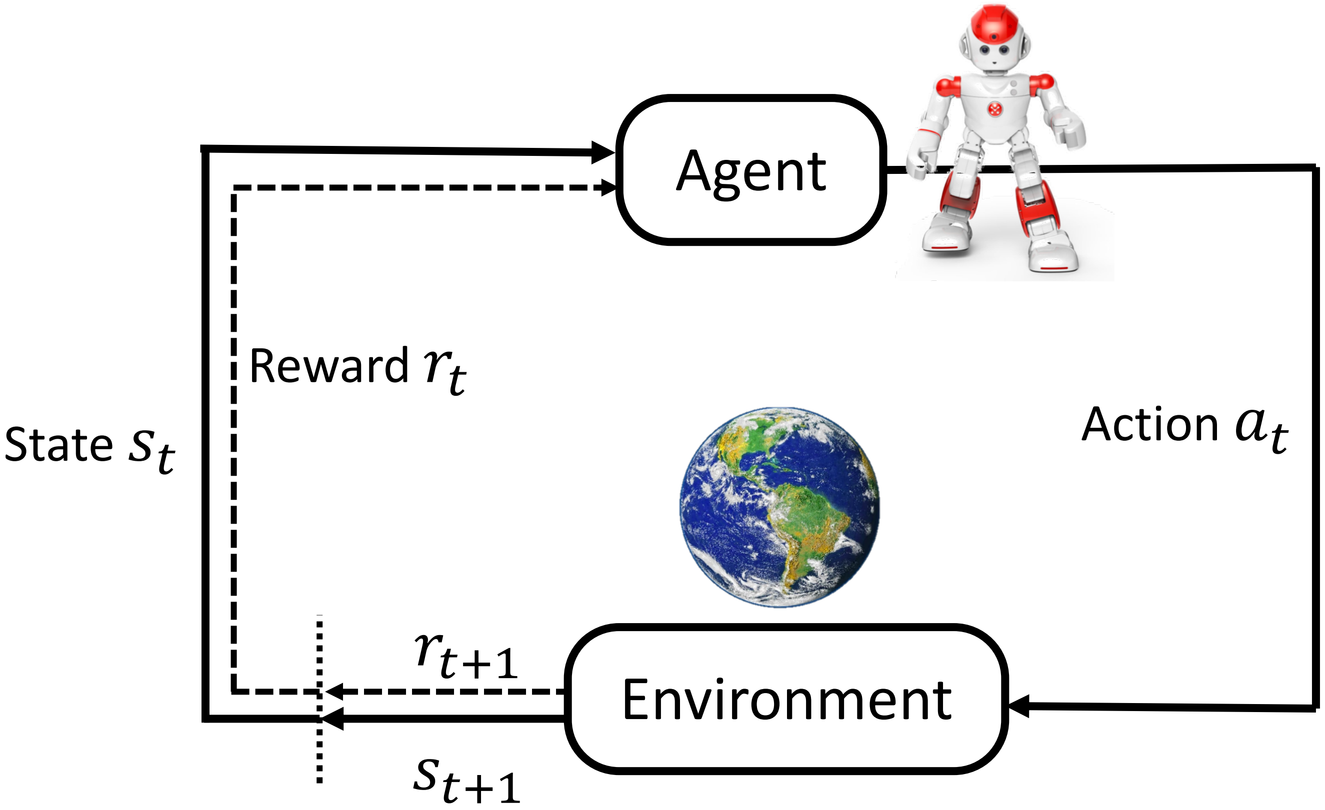
facebook Artificial Intelligence

Overview


Building Scalable System for Reinforcement Learning (RL)
Learn Hand-tuned Heuristics by RL / ML

Building Scalable System for RL

Crash Course of Reinforcement Learning



Reinforcement Learning works, but expensive



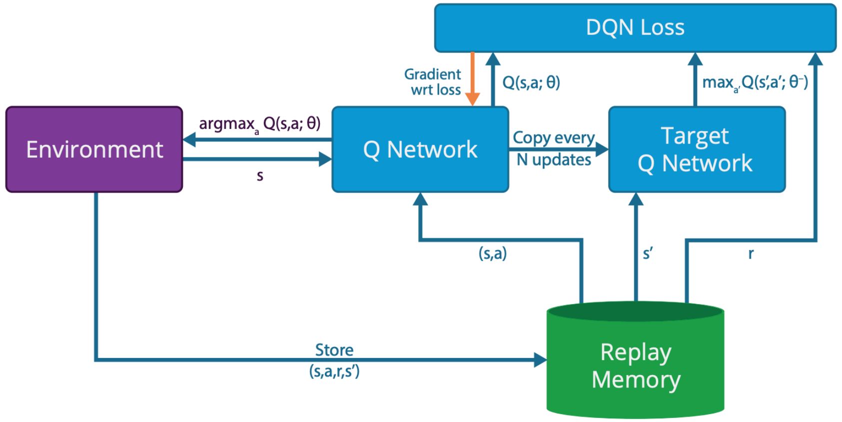


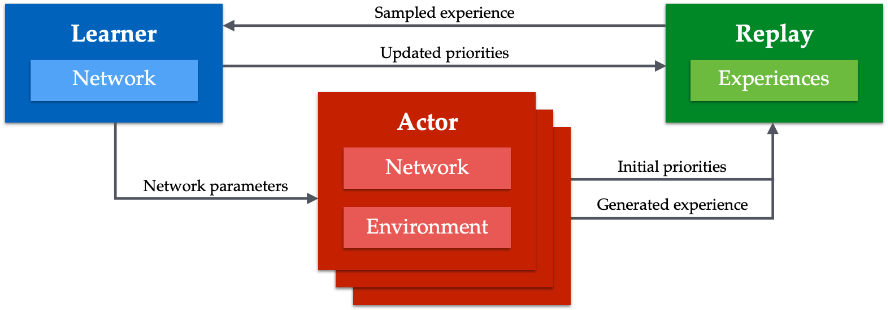
Year	Projects	Human Data	Training Resource	Training time
2016	DeepMind's AlphaGo	Yes	~50 GPUs + ? CPUs	~1 week
2017	DeepMind's AlphaGo Zero (20 blocks)	No	~2000 TPUs	3 days
2017	DeepMind's AlphaZero (20 blocks)	No	~5000 TPUs	8 hours
2018	OpenAI Five	No	128,000 CPUs + 256 GPUs	Several months
2019	DeepMind's AlphaStar	Yes	16,000 CPUs + 3072 TPUv3 cores	44 days

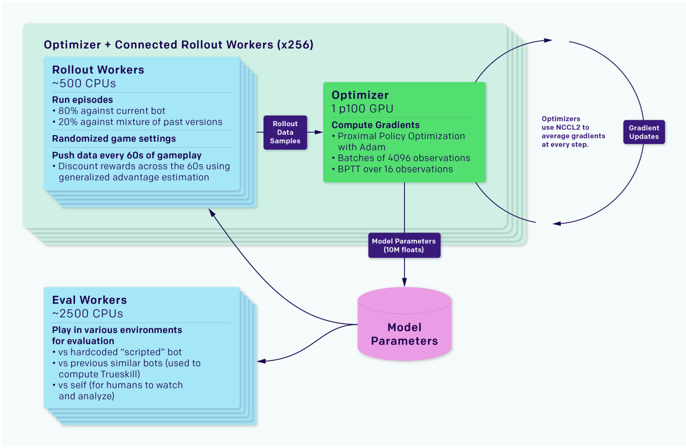
Challenges in large-scale RL Training System

- Trade-offs in a *heterogenous* system
  - **Different kind of objects**: Actor / Environment / Trainer / Replay buffer
  - CPUs / GPUs Allocations
  - Multi-threading versus Multiple Processes, Batching issues
  - Local versus Distributed
  - Synchronization / Asynchronization.
    - On-policy versus off-policy methods
    - Perfect synchronization might NOT give you the best performance
- Mingled Algorithm Design and System Design
  - New System design ←→ New RL algorithm

Distributed System for training RL agent







GORILLA

[Massively Parallel Methods for Deep Reinforcement Learning, AAAI 2015]

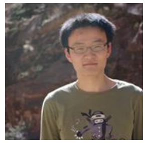
Ape-X / R2D2

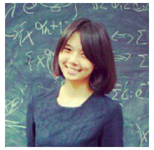
[Distributed Prioritized Experience Replay, Horgan et al, ICLR 2018]
[Recurrent Experience Replay in Distributed Reinforcement Learning Kapturowski et al, ICLR 2019]

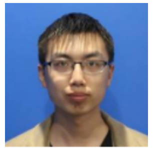
OpenAI Rapid

ELF: RL Framework for Game Research

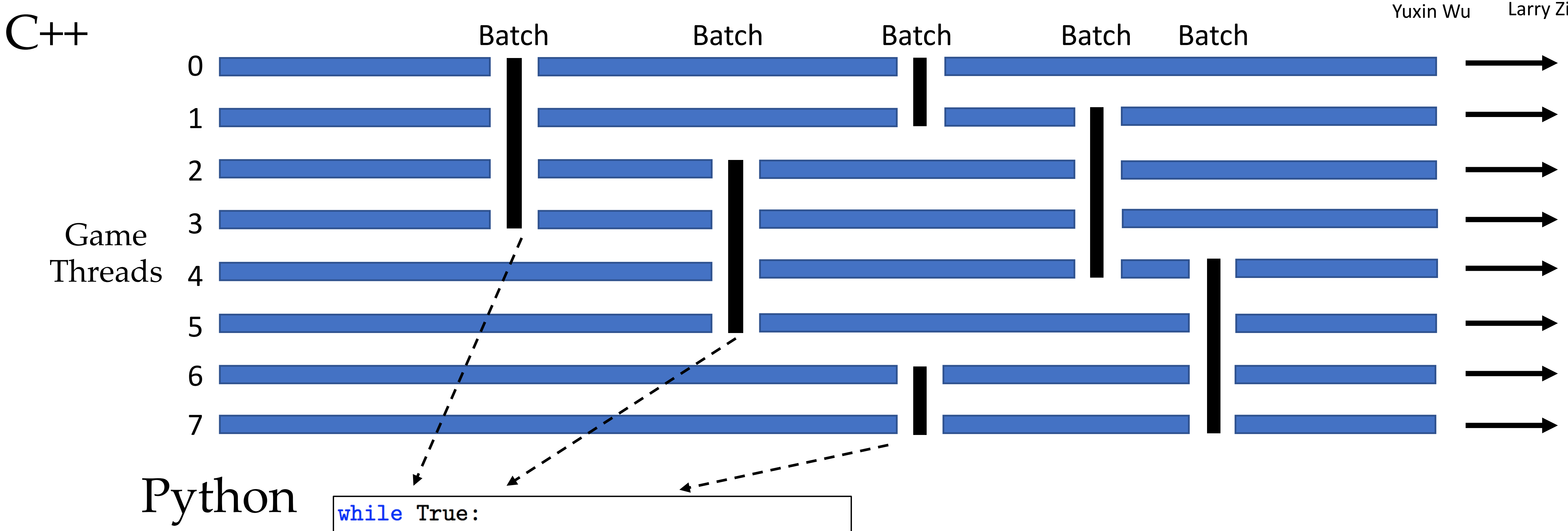

Yuandong Tian


Qucheng Gong


Wendy Shang


Yuxin Wu


Larry Zitnick

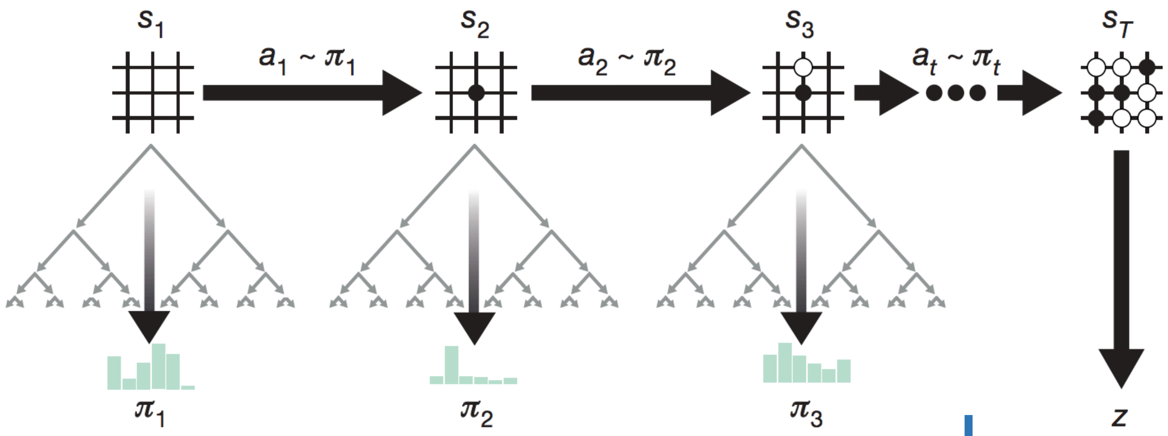



```python
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```

[ELF: Extensive, Lightweight and Flexible Framework for Game Research, Tian et al, NeurIPS 2019]

facebook Artificial Intelligence

AlphaGoZero / AlphaZero



Generate Training data

<u>Without human knowledge</u>

θ_i ← Update Models ← Self-Replays

[Silver et al, Mastering the game of Go without human knowledge, Nature 2017]

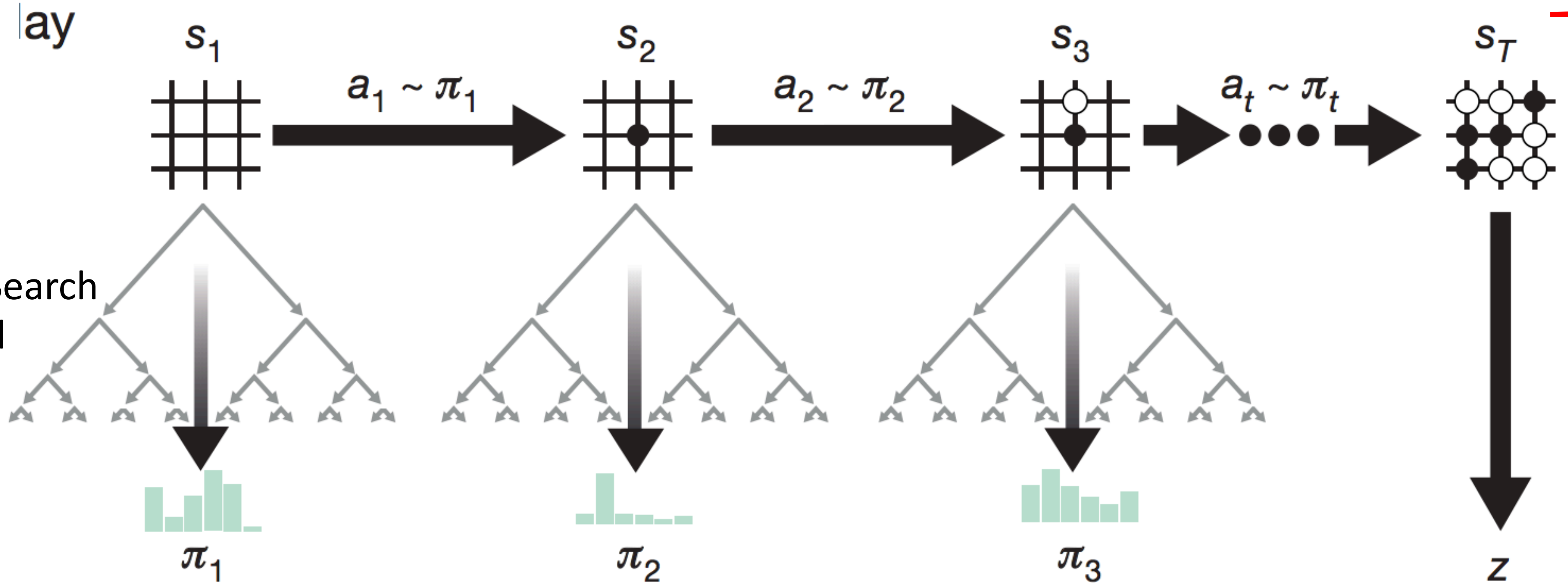
Generate Self-play Games

$\left(s_1, \pi_1, z\right)$

$\left(s_2, \pi_2, z\right)$

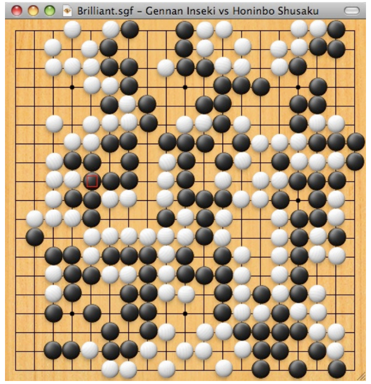
$\left(s_3, \pi_3, z\right)$

Training samples for θ_{i+1}

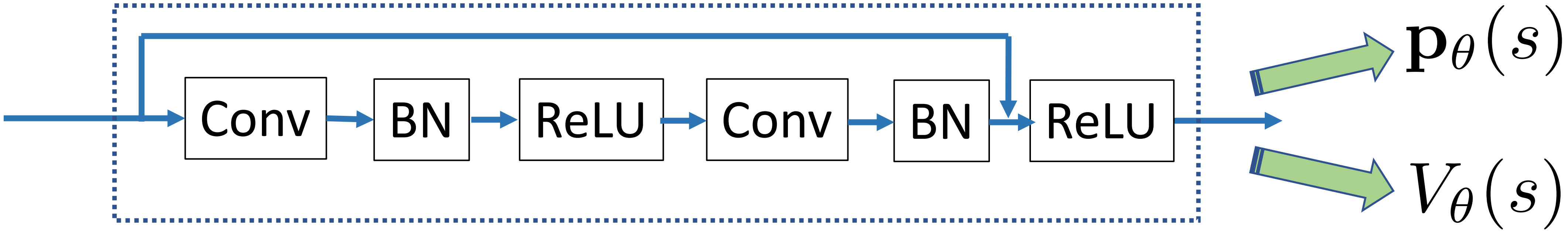


Monte Carlo Tree Search with current model θ_i

Update Models



$\boldsymbol{s}$



Player situation at time 0

Opponent situation at time 0

Player situation at t=-7

Color to play

Input features (19x19x17): $(X, Y, X_{-1}, Y_{-1}, \ldots, X_{-7}, Y_{-7}, C)$

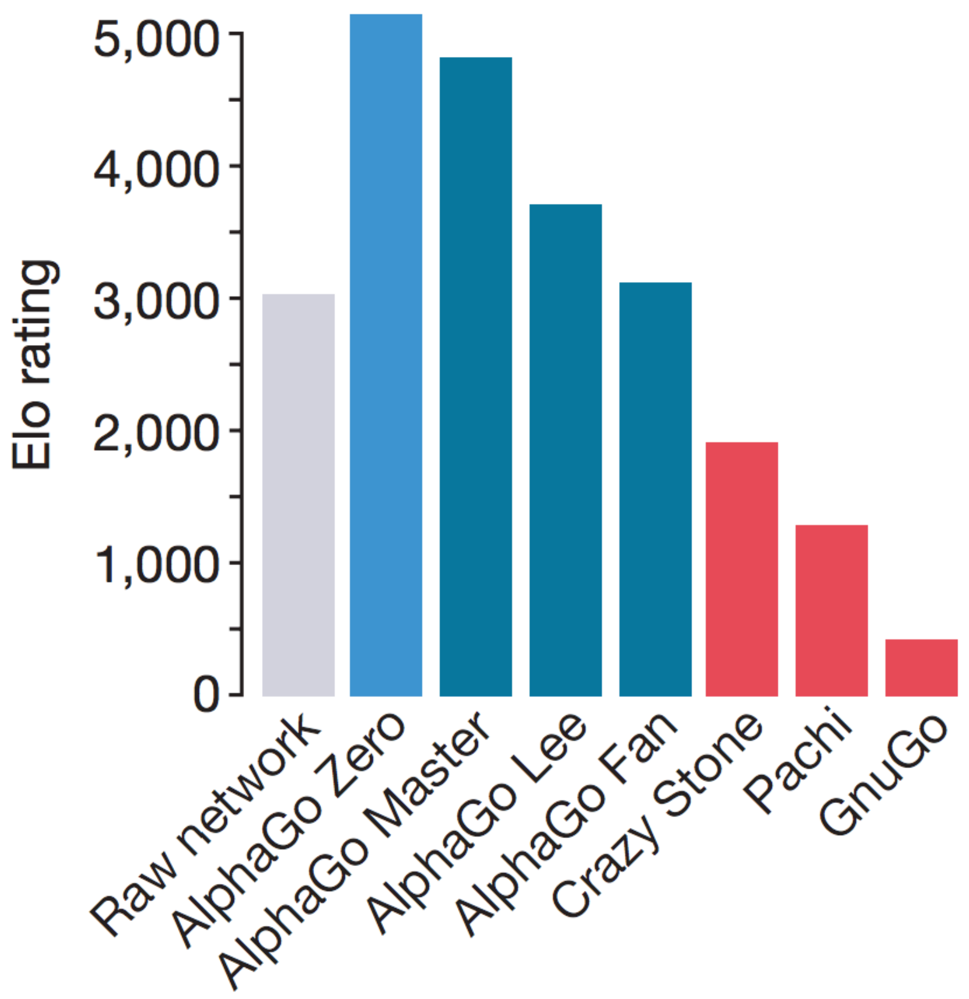
Objective: $(s, \boldsymbol{\pi}, z)$

$$J(\theta) = (z - V_\theta)^2 - \boldsymbol{\pi}^T \log \mathbf{p}_\theta + c\|\theta\|^2$$

AlphaGo Zero Strength

- 3 days version
  - 4.9M Games, 1600 rollouts/move
  - 20 block ResNet
  - Defeat AlphaGo Lee.

- 40 days version
  - 29M Games, 1600 rollouts/move
  - 40 blocks ResNet.
  - Defeat AlphaGo Master by 89:11



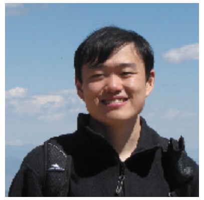
The Mystery of AlphaZero

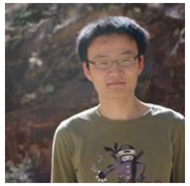
- Mystery
  - Is the proposed algorithm really universal?
  - Is the bot almighty? Is there any weakness in the trained bot?

- Lack of Ablation Studies
  - What factor is critical for the performance?
  - Is the algorithm robust to random initialization and changes of hyper parameters?
  - Any adversarial samples?

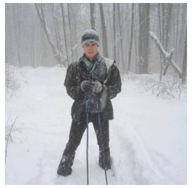
Impressive Results, No code, No model

ELF OpenGo


Yuandong Tian

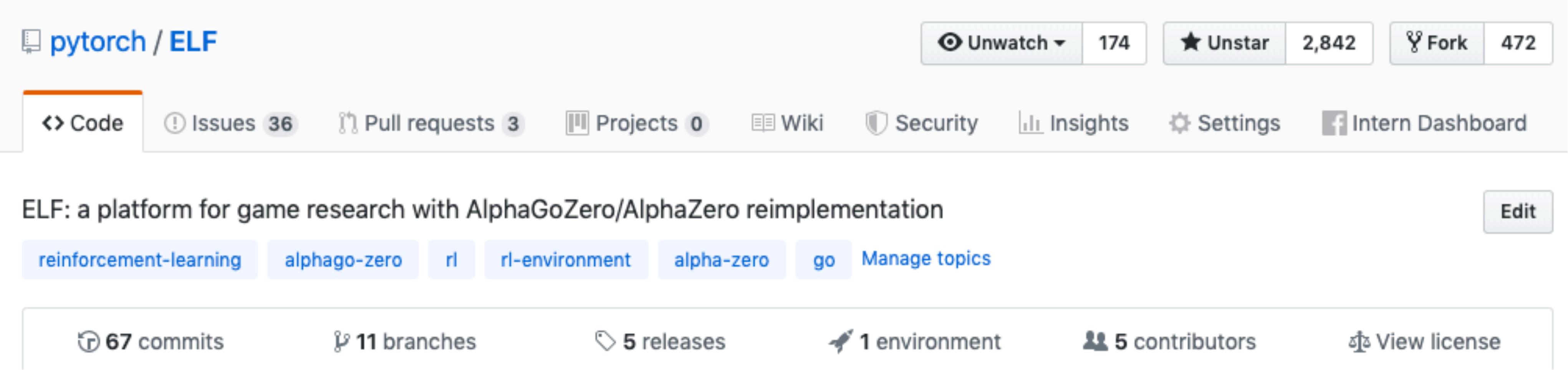
Jerry Ma*

Qucheng Gong*

Shubho Sengupta*


Zhuoyuan Chen

James Pinkerton

Larry Zitnick

- System can be trained with 2000 GPUs in 2 weeks (20 block version)
- Superhuman performance against professional players and strong bots.
- Abundant ablation analysis.



We open source the code and the pre-trained model for the Go and ML community

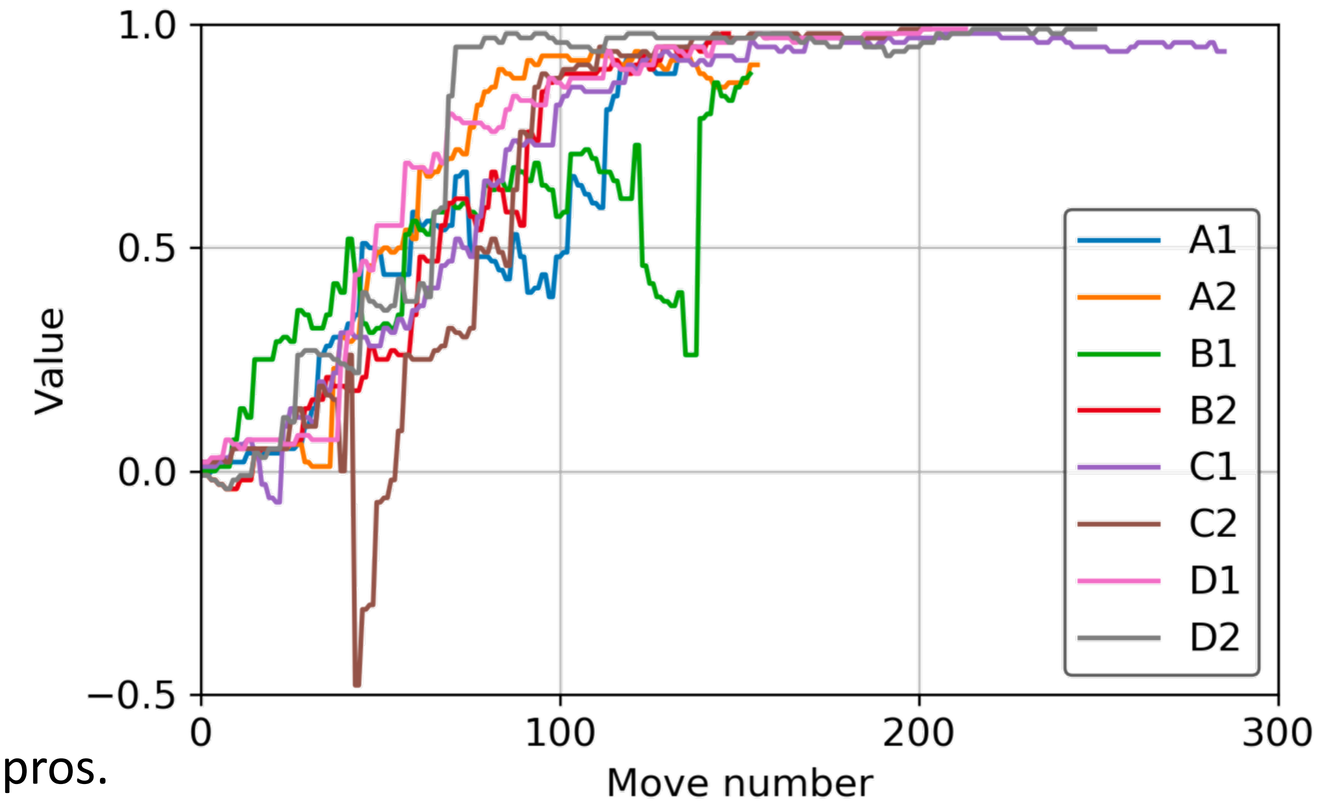
[ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero, Y. Tian et al, ICML 2019]

ELF OpenGo Performance

Vs top professional players

Name (rank)	ELO (world rank)	Result
Kim Ji-seok	3590 (#3)	5-0
Shin Jin-seo	3570 (#5)	5-0
Park Yeonghun	3481 (#23)	5-0
Choi Cheolhan	3466 (#30)	5-0

Single GPU, 80k rollouts, 50 seconds
Offer unlimited thinking time for the players



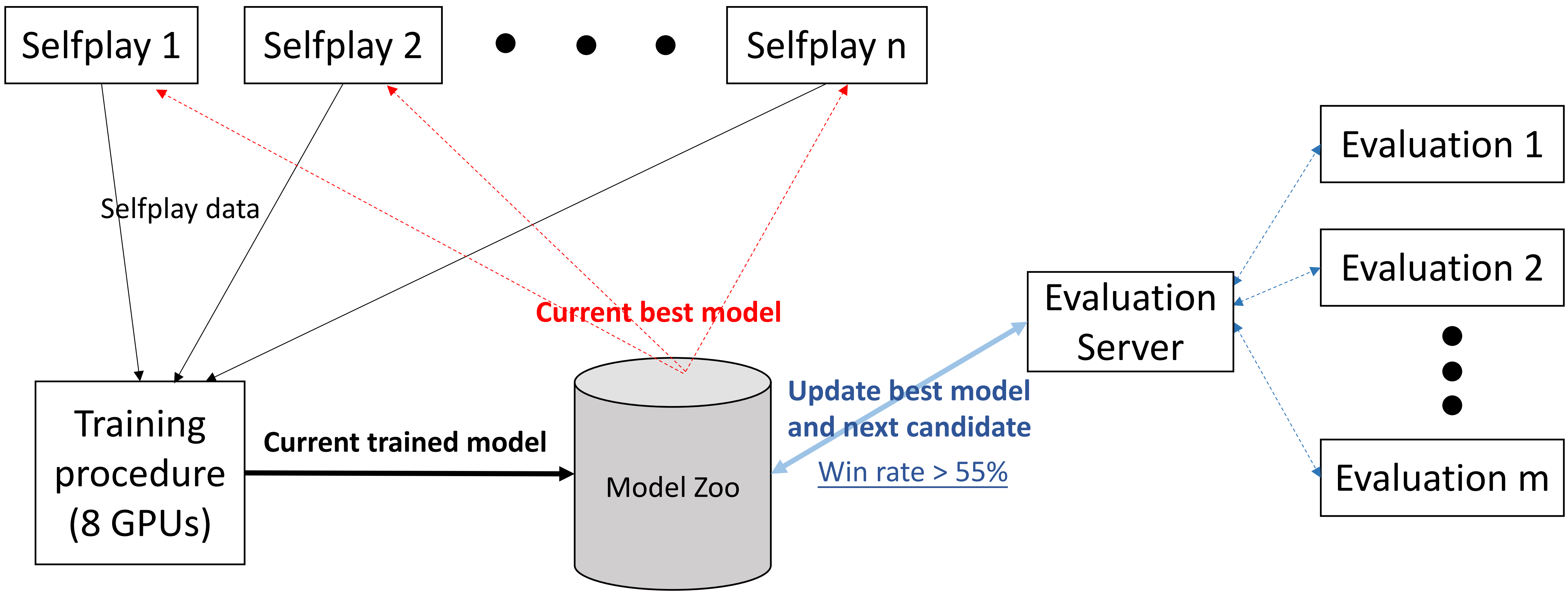
Vs professional players

Single GPU, 2k rollouts, 27-0 against Taiwanese pros.

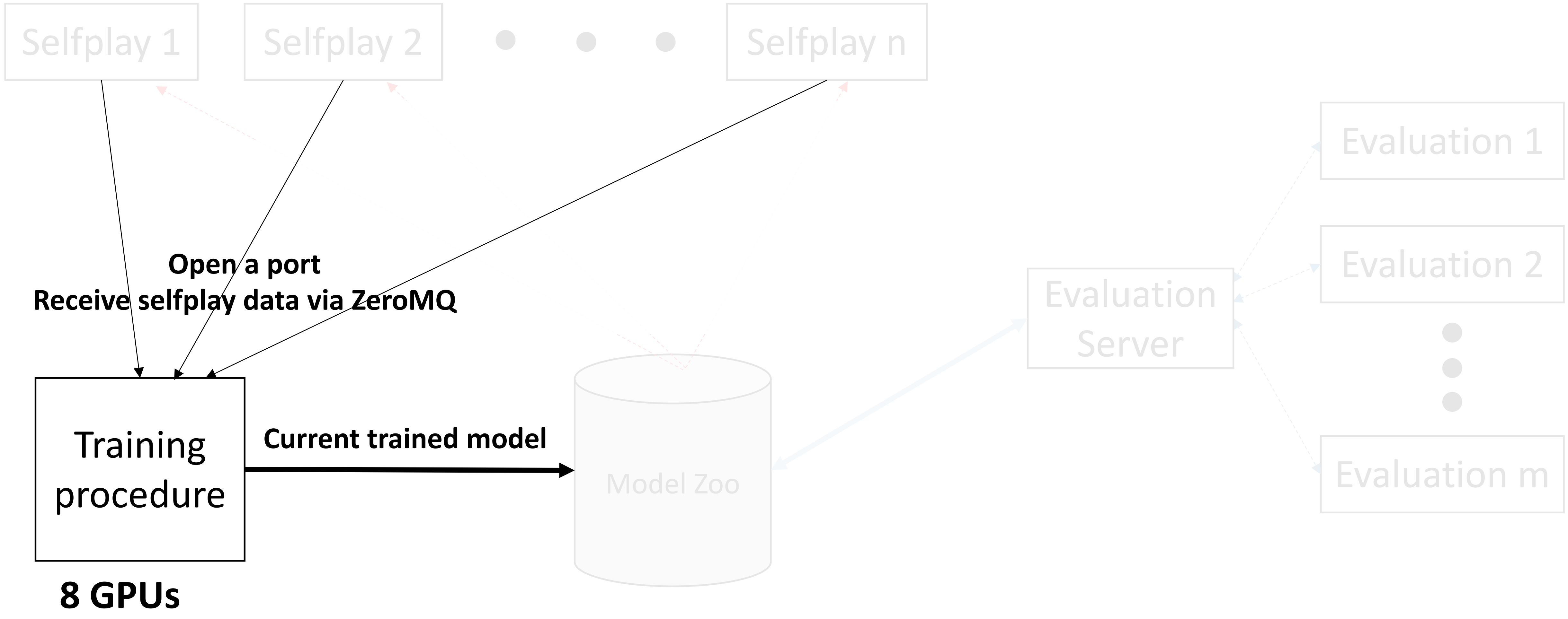
Vs strong bot (LeelaZero)

[158603eb, 192x15, Apr. 25, 2018]: 980 wins, 18 losses (98.2%)

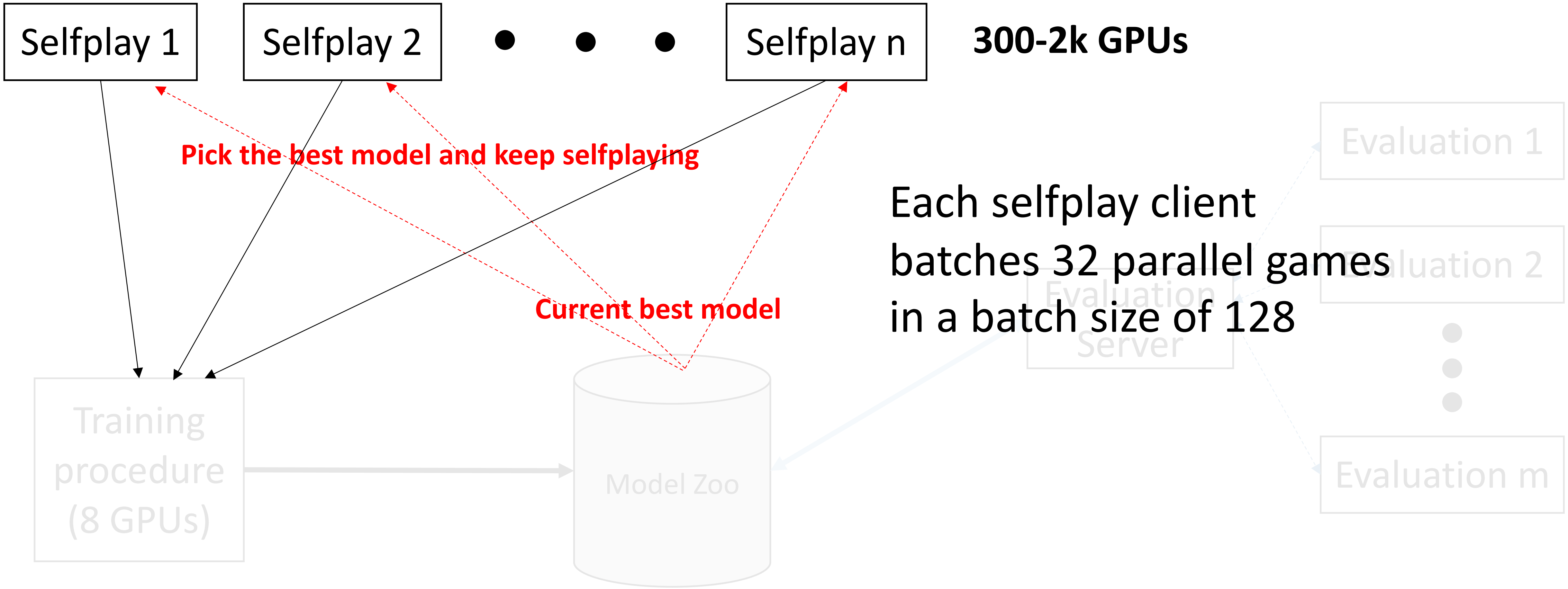
Distributed ELF (version 1, AlphaGoZero)



Distributed ELF (version 1)

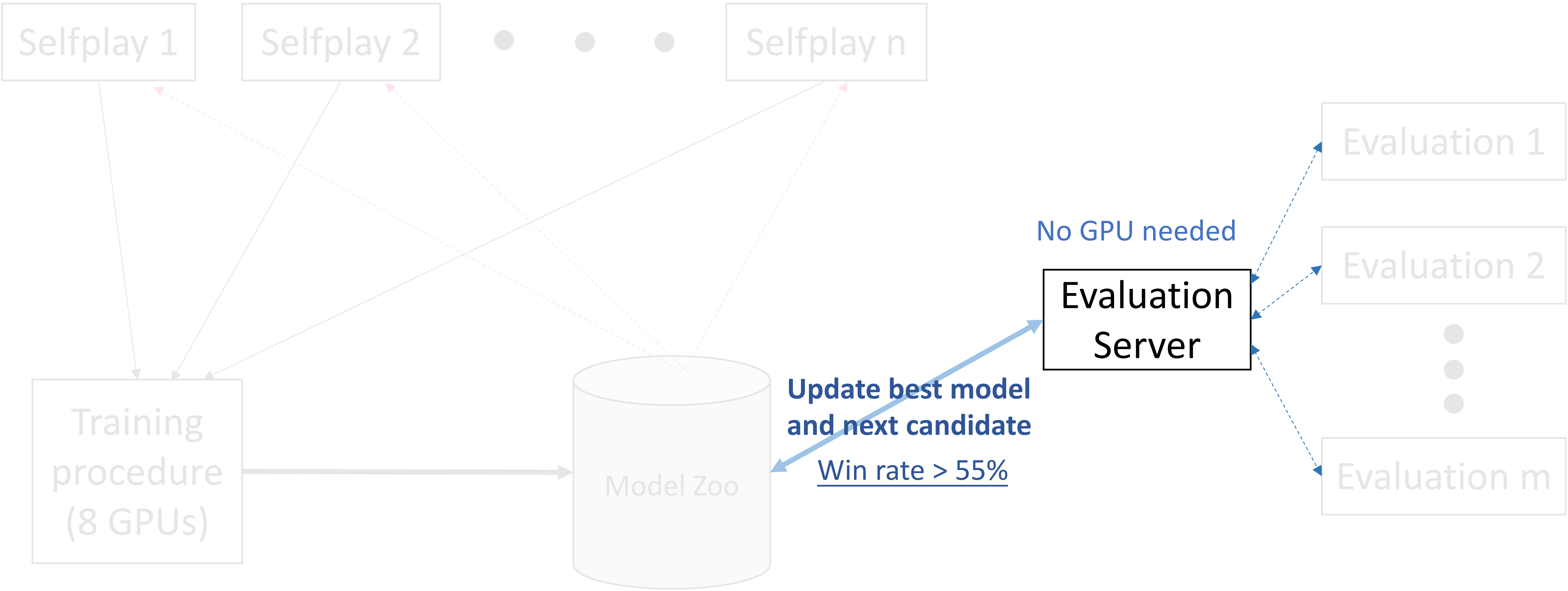


Distributed ELF (version 1)

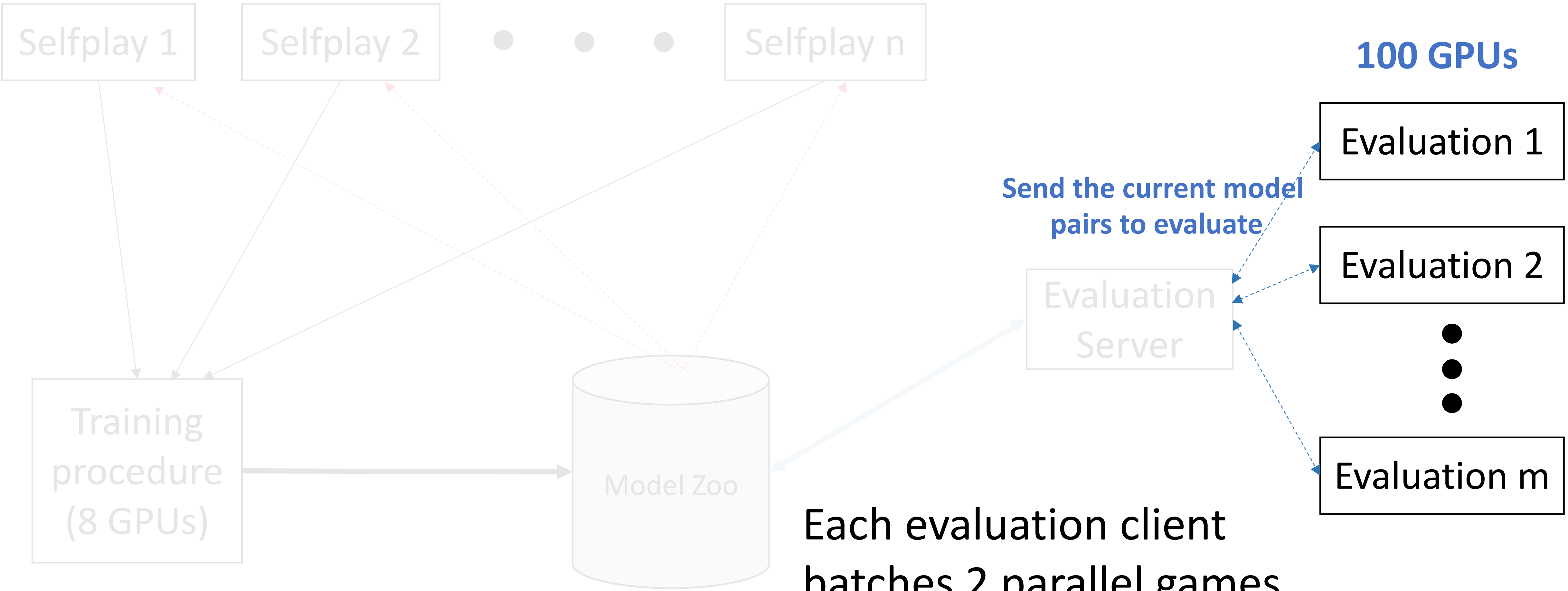


Each selfplay client batches 32 parallel games in a batch size of 128

Distributed ELF (version 1)

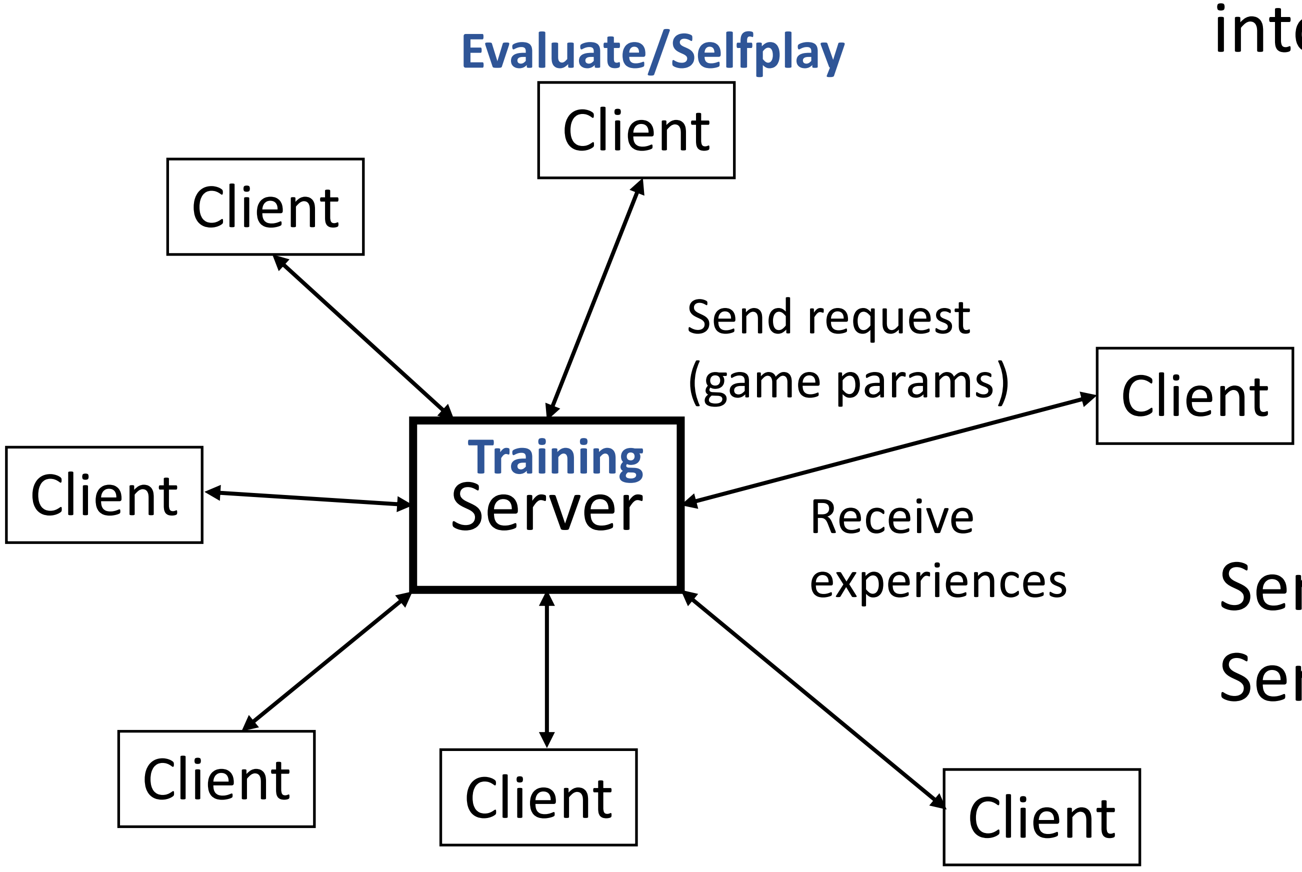


Distributed ELF (version 1)



Each evaluation client batches 2 parallel games

Distributed ELF (v2)



Putting AlphaGoZero and AlphaZero into the same framework

AlphaGoZero (more synchronization)
AlphaZero (less synchronization)

Server controls synchronization
Server also does training.

Next Step: **RL Assembly**

- Backbone infrastructure for ongoing projects (Hanabi, Bridge, etc)
- Reimplementation of SoTA off-policy RL methods like Ape-X and R2D2
- Incorporate OpenGo and SoTA implementation of MCTS.
- Efficient on single machine (SoTA training FPS so far)

Open source soon

Frame Per Second (FPS) on Atari Games

ReLA: 12.5 KFPS

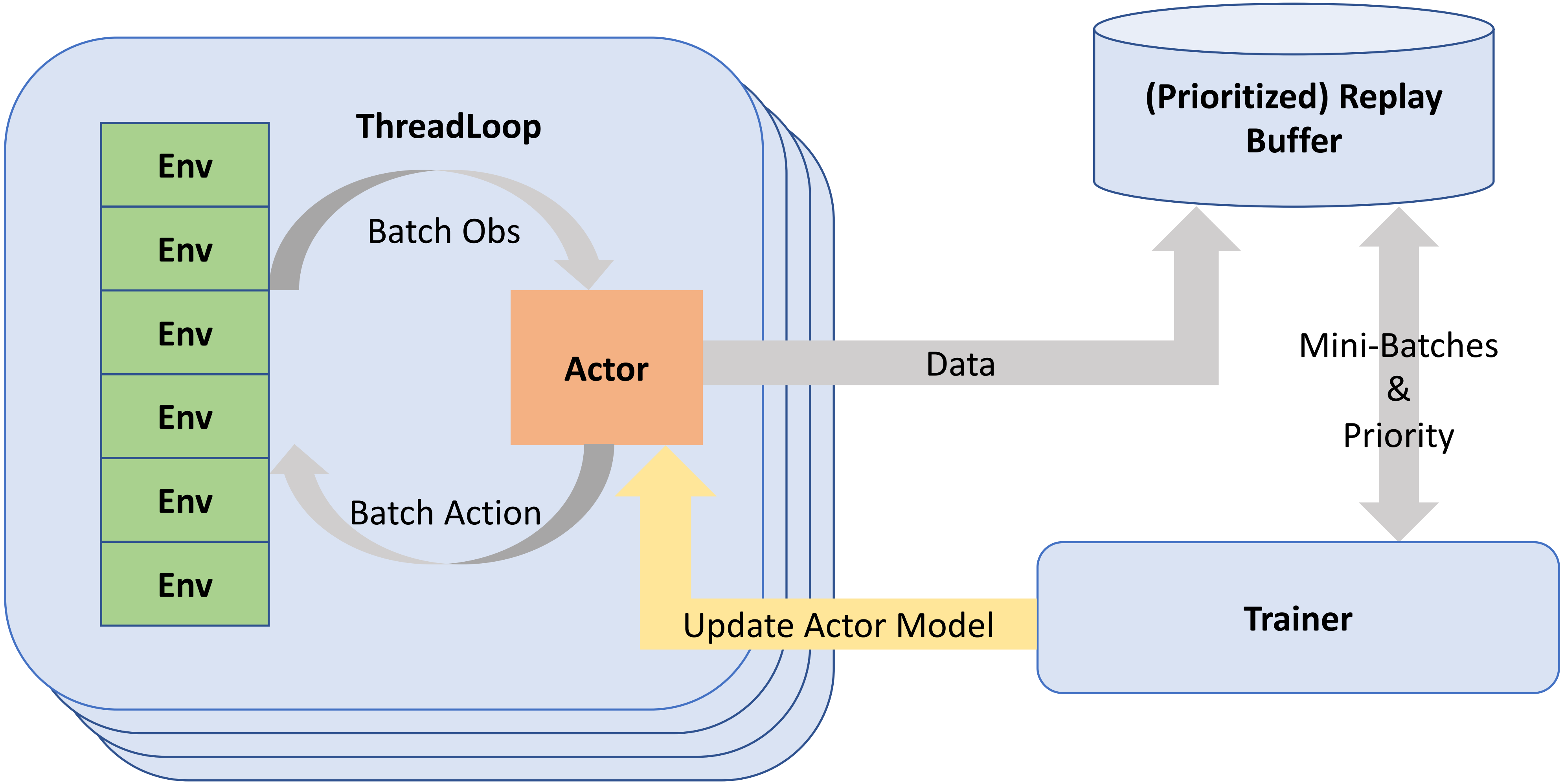
using 40 CPU cores + 2 GPU (P100) on a single machine

Ape-X: 12.5 KFPS

using 360 CPU cores + 1 GPU (distributed system)

- ReLA is GPU bound. Performance is better with more GPUs
- A few more improvements to achieve better performance when releasing.

Architecture



User Interface (API)

```python
env = rela.VectorEnv()
for _ in range(num_env_per_thread):
    game = create_atari(...)
    env.append(game)
actor = rela.DQNActor(...)
thread = rela.ThreadLoop(actor, env)
```

All objects (env, agent, replay buffer, etc) are created & configured in Python

User Interface (API)

```python
class ApexAgent(torch.jit.ScriptModule):
    @torch.jit.script_method
    def td_err(self, obs: Dict[str, torch.Tensor], ...) -> torch.Tensor:
        online_q = self.online_net(obs)
        online_qa = online_q.gather(1, action.unsqueeze(1)).squeeze(1)

        next_a = self.greedy_act(next_obs)
        bootstrap_q = self.target_net(next_obs)
        bootstrap_qa = bootstrap_q.gather(1, next_a.unsqueeze(1)).squeeze(1)
        target = reward + bootstrap * (self.gamma ** self.multi_step) * bootstrap_qa
        return target.detach() - online_qa
```

Model is written in **Python** with **PyTorch's TorchScript**,
and executed in **C++** with multi-threading for maximum throughput.

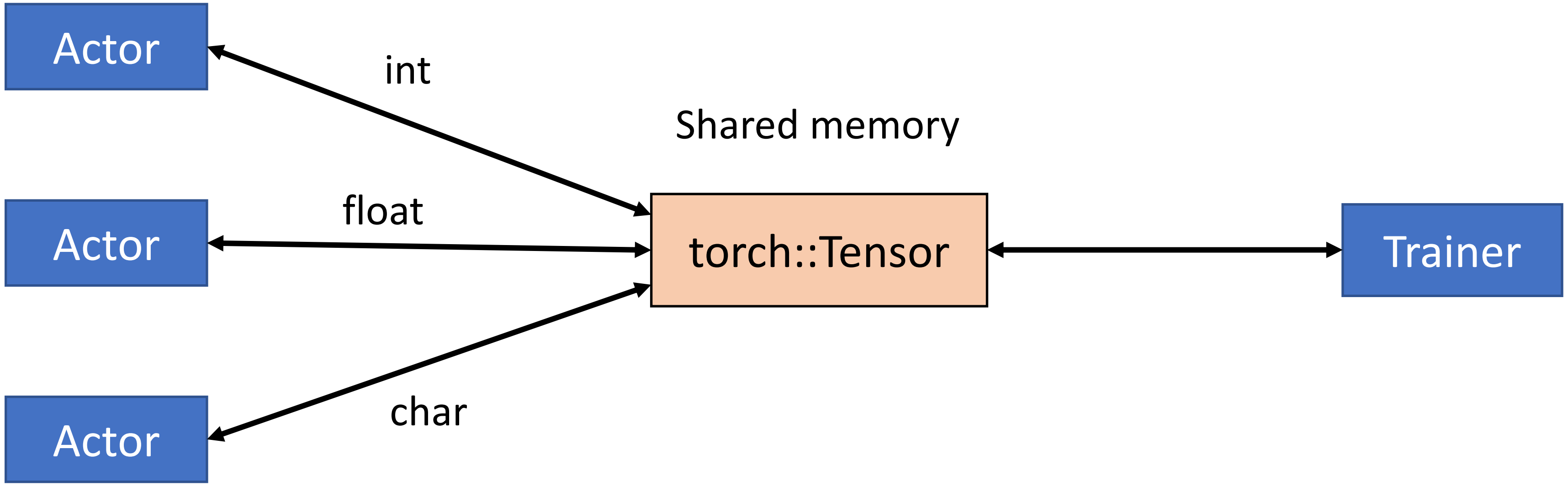
Native integration with PyTorch C++ API

- Simple/Intuitive manipulations of PyTorch tensors in C++
  - Same as/Similar to Python Interface
  - No extra library needed for operations like downsample/upsampling.

```cpp
torch::Tensor s = getObservation();
s = s.view({1, 3, height, width});
// rescale the image
s = torch::upsample_bilinear2d(s, {sHeight, sWidth}, true);
s = s.view({3, sHeight, sWidth});
// convert to grey scale
s = 0.21 * s[0] + 0.72 * s[1] + 0.07 * s[2];
```

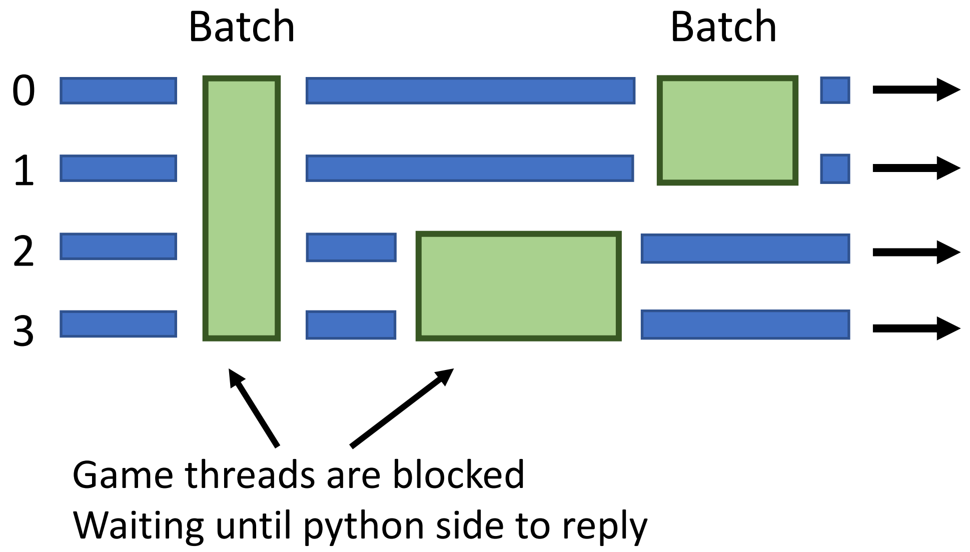
Native integration with PyTorch C++ API

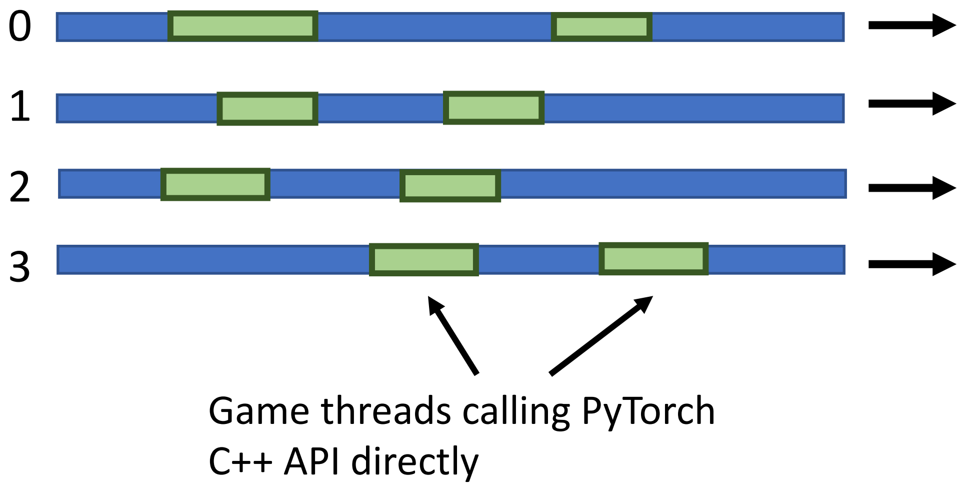
- Easier communication between threads/processes via Tensor.
  - No extra copy when sending data from/to environments.



Native integration with PyTorch C++ API

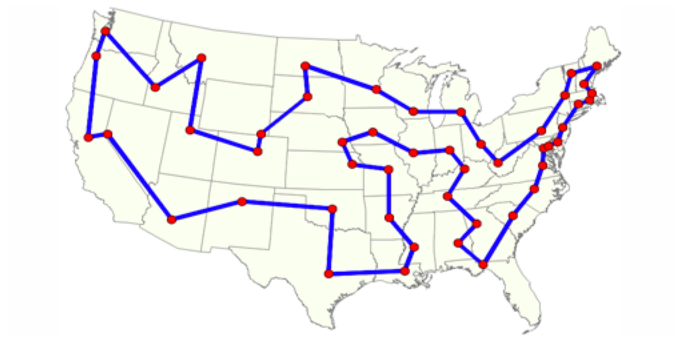
- Simultaneous network forwarding at different threads
  - Python GIL becomes irrelevant.
  - No need to block the environment
    - good for simple environments like Go, Bridge, Hanabi and others.



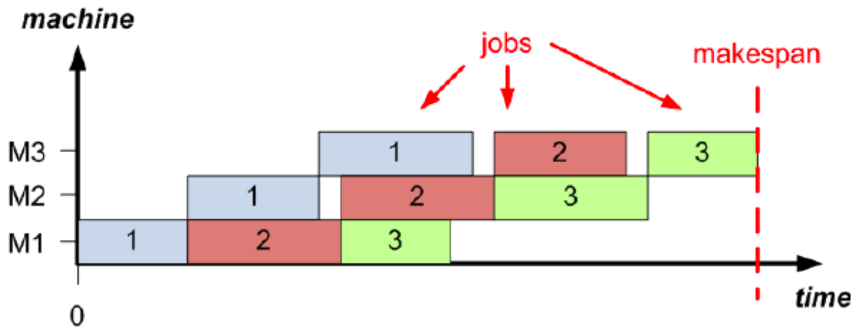


Learning Hand-tuned Heuristics with RL/ML

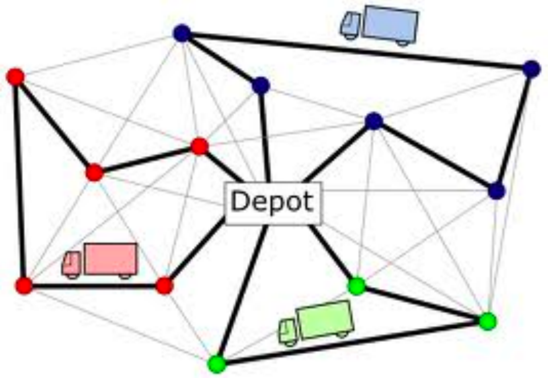
Combinatorial optimization



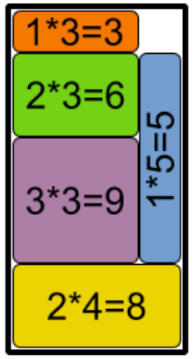
Travel Salesman Problem



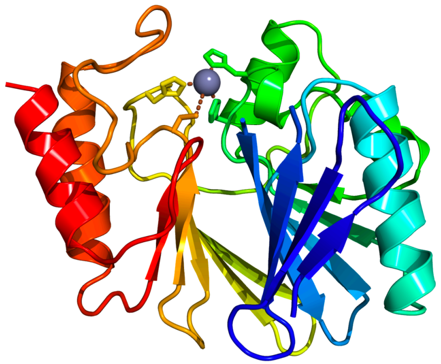
Job Scheduling



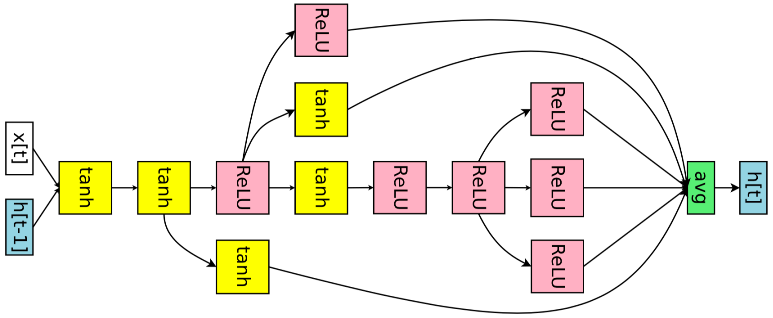
Vehicle Routing



Bin Packing



Protein Folding



Model-Search

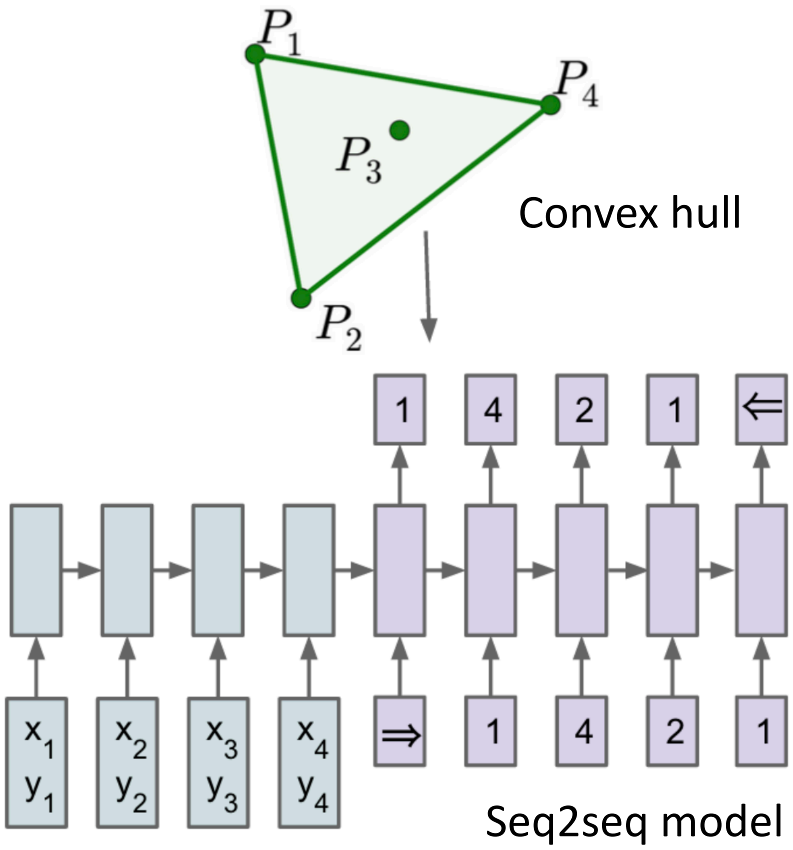
Wait...What?

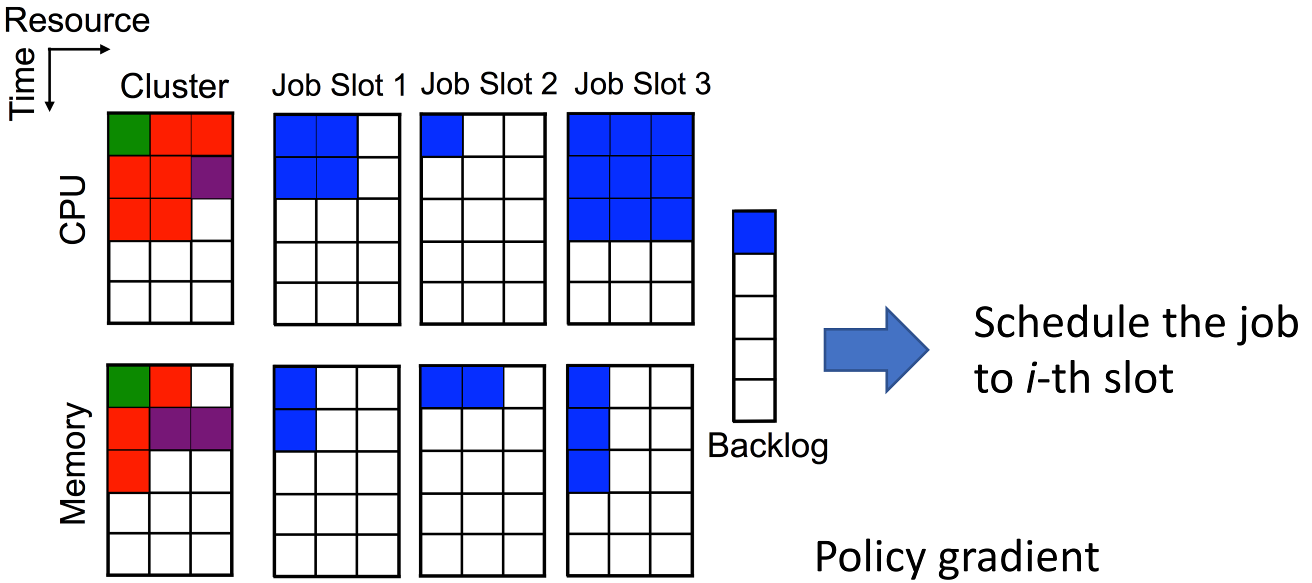
- These are NP-hard problems.
  - No good algorithm unless $P = NP$

- These guarantees are worst-case ones.
  - To prove a lower-bound, construct an adversarial example to fail the algorithm

- For specific distribution, there might be better heuristics.
  - Human heuristics are good but may not be suitable for everything

Direct predicting combinatorial solutions



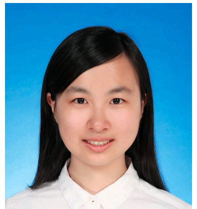


[H. Mao et al, Resource Management with Deep Reinforcement Learning, ACM Workshop on Hot Topics in Networks, 2016]

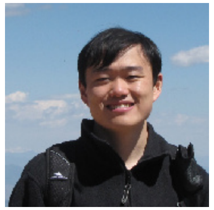
[O. Vinyals. et al, Pointer Networks, NIPS 2015]

Local Rewriting Framework

Code: https://github.com/facebookresearch/neural-rewriter



Xinyun Chen

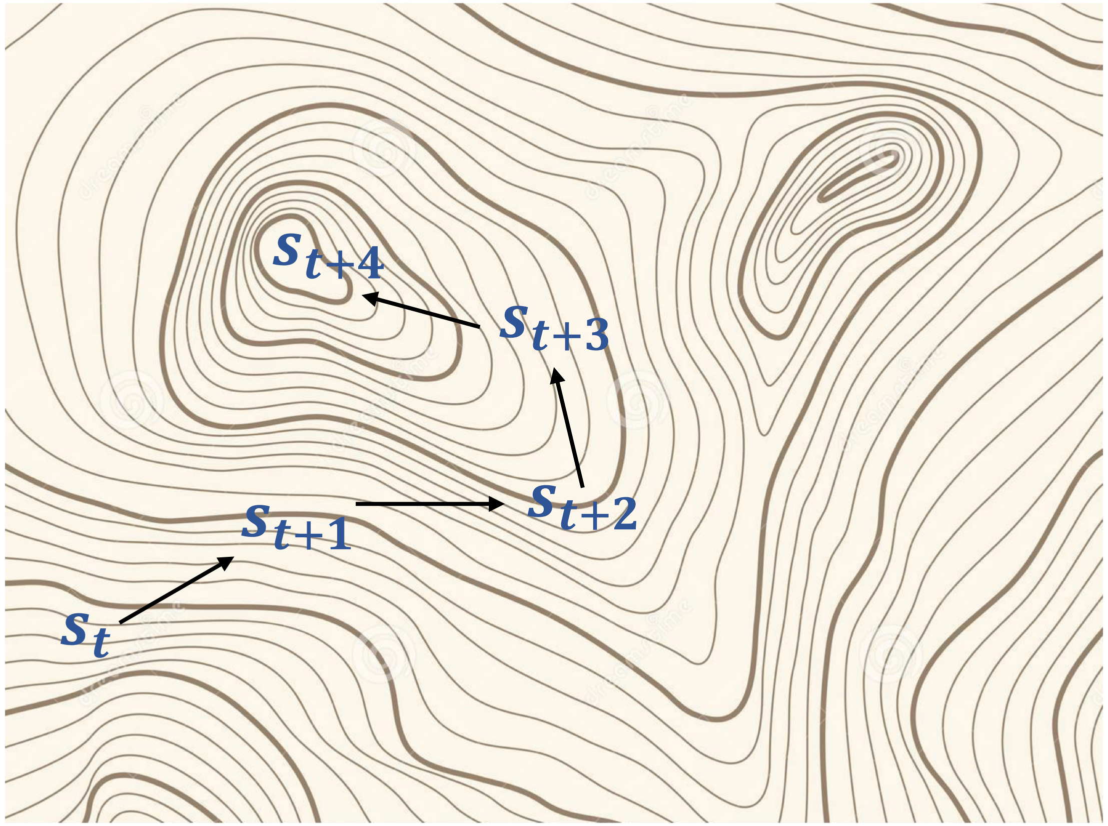


Yuandong Tian

A learned "gradient descent" that

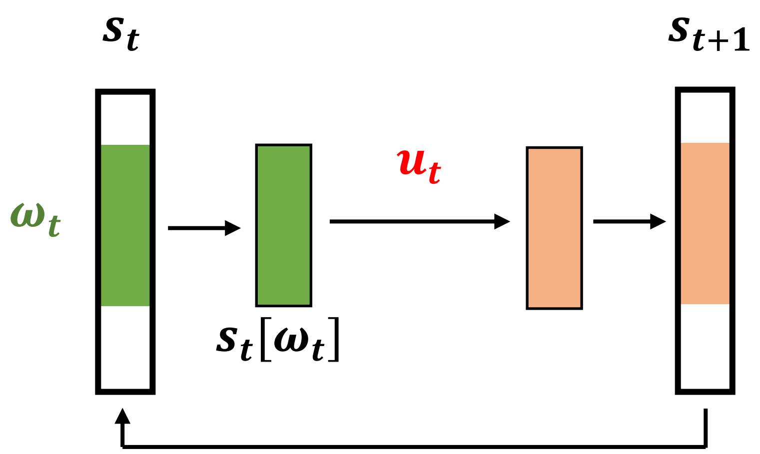
starts from a feasible solution
iteratively converges to a good solution

How to learn it?



 [Learning to Perform Local Rewriting for Combinatorial Optimization, X. Chen and Y. Tian, NeurIPS 2019]

Local Rewriting Framework



Current State (i.e. Solution)

Region-Picker

Rule-Picker

$$s_t \longrightarrow \omega_t \sim \pi_\omega(\cdot\,|s_t) \longrightarrow u_t \sim \pi_u(\cdot\,|s_t\,[\omega_t])$$

$$s_{t+1} = f(s_t, \omega_t, u_t)$$

Q-Actor-Critic Training

How to train two policies $\boldsymbol{\pi_\omega}(\cdot\,|\boldsymbol{s_t})$ and $\boldsymbol{\pi_u}(\cdot\,|\boldsymbol{s_t}\ [\boldsymbol{\omega_t}])$?

Learn Q to fit cumulative rewards:

$$L_\omega(\theta) = \frac{1}{T}\sum_{t=0}^{T-1}\Big(\sum_{t'=t}^{T-1}\gamma^{t'-t} r(s'_t, (\omega'_t, u'_t)) - Q(s_t, \omega_t; \theta)\Big)^2$$

$\boldsymbol{\pi_\omega}(\cdot\,|\boldsymbol{s_t})$: Q-learning with soft policy:

$$\pi_\omega(\omega_t|s_t;\theta) = \frac{\exp(Q(s_t, \omega_t; \theta))}{\sum_{\omega_t}\exp(Q(s_t, \omega_t; \theta))}$$

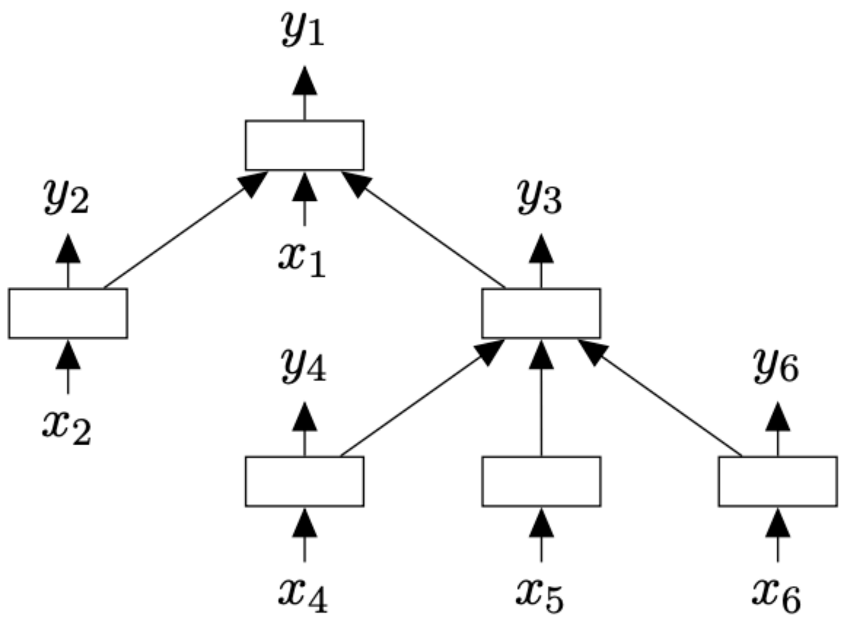
$\boldsymbol{\pi_u}(\cdot\,|\boldsymbol{s_t}\ [\boldsymbol{\omega_t}])$: Actor-Critic with learned Q:

$$L_u(\phi) = -\sum_{t=0}^{T-1}\Delta(s_t, (\omega_t, u_t))\log \pi_u(u_t|s_t[\omega_t];\phi)$$

Advantage:

$$\Delta(s_t, (\omega_t, u_t)) \equiv \sum_{t=t'}^{T-1}\gamma^{t'-t} r(s'_t, (\omega'_t, u'_t)) - Q(s_t, \omega_t; \theta)$$

How to encode Structure Data



Child-Sum LSTM

$$y_1 = f(y_2, y_3, x_1)$$

f can be very complicated:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k,$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right),$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right),$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right),$$

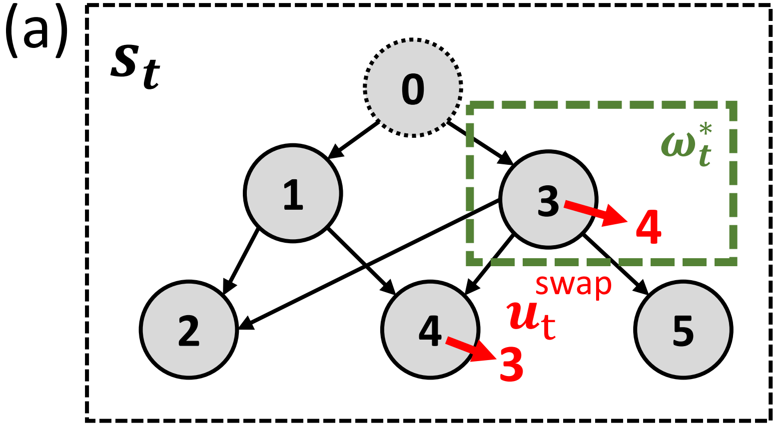
$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right),$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k,$$

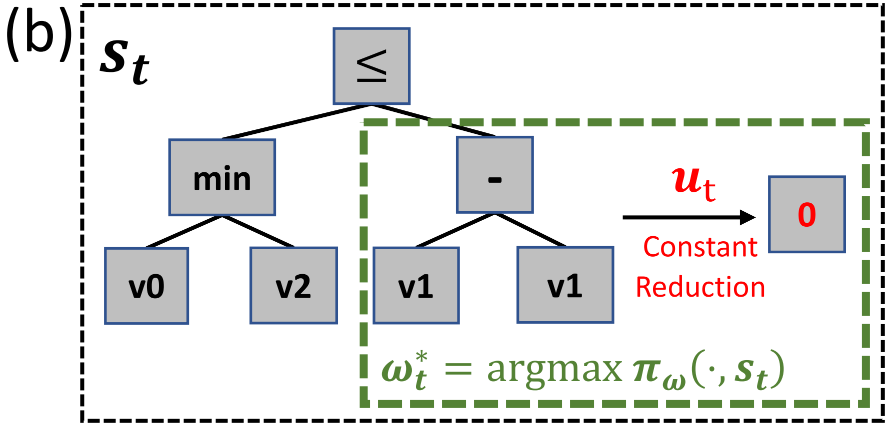
$$h_j = o_j \odot \tanh(c_j),$$

[Improved Semantic Representation From Tree-Structured Long Short-Term Memory Networks. K. Tai et al]

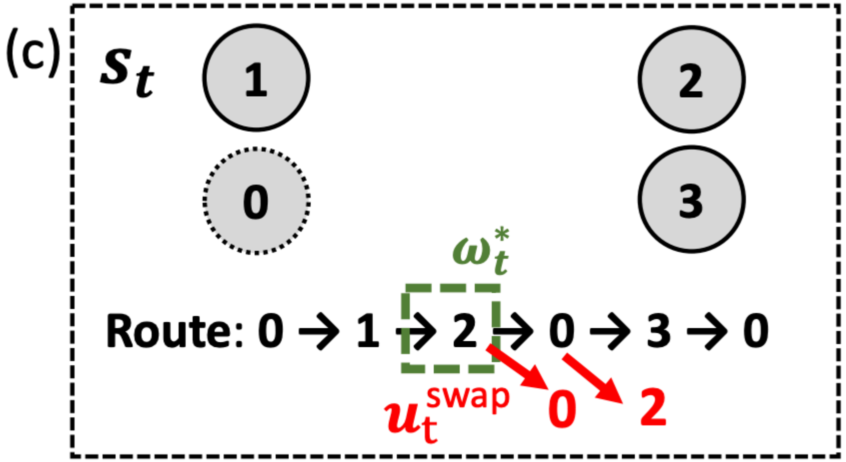
Applications



Online Job Scheduling

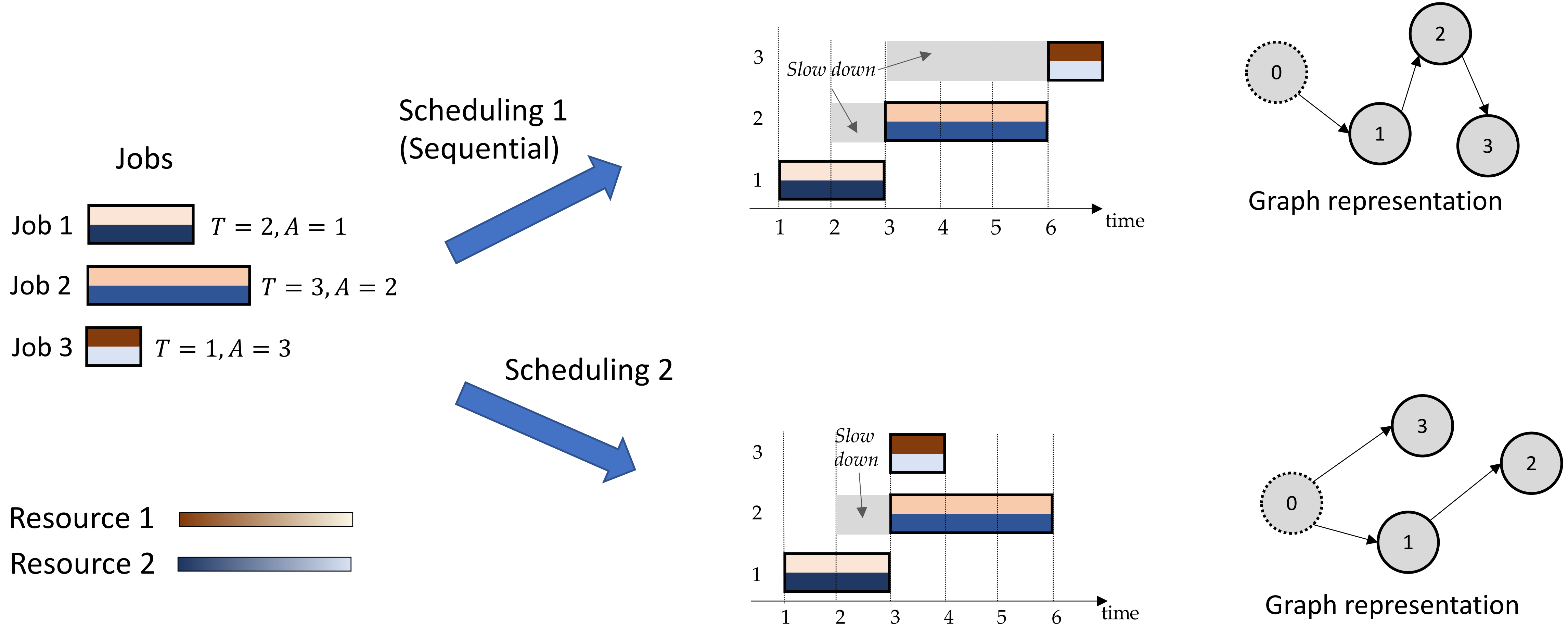


Expression Simplification

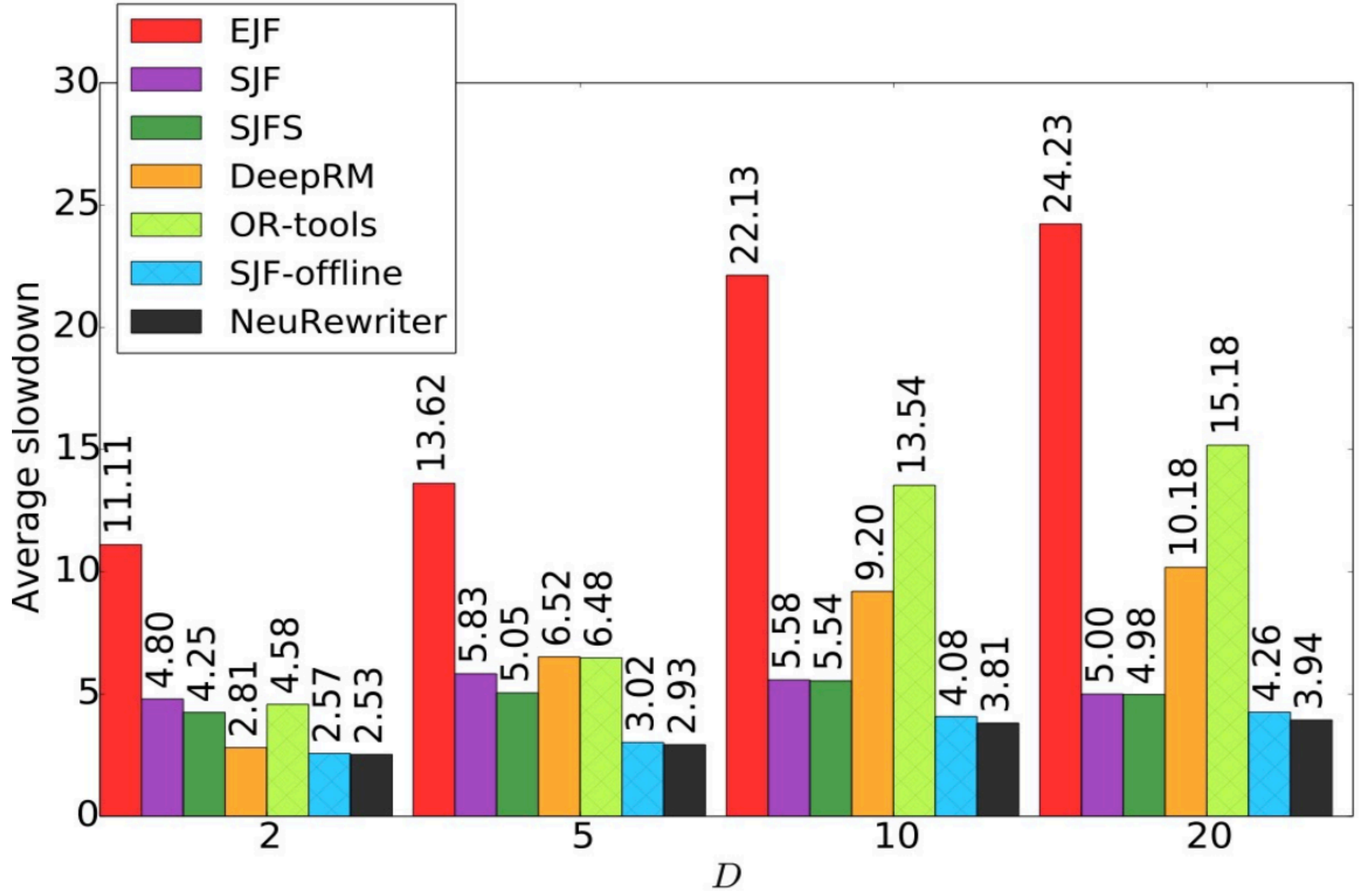


Vehicle Routing

Online Job Scheduling



Online Job Scheduling



D: Number of resources

Baselines:

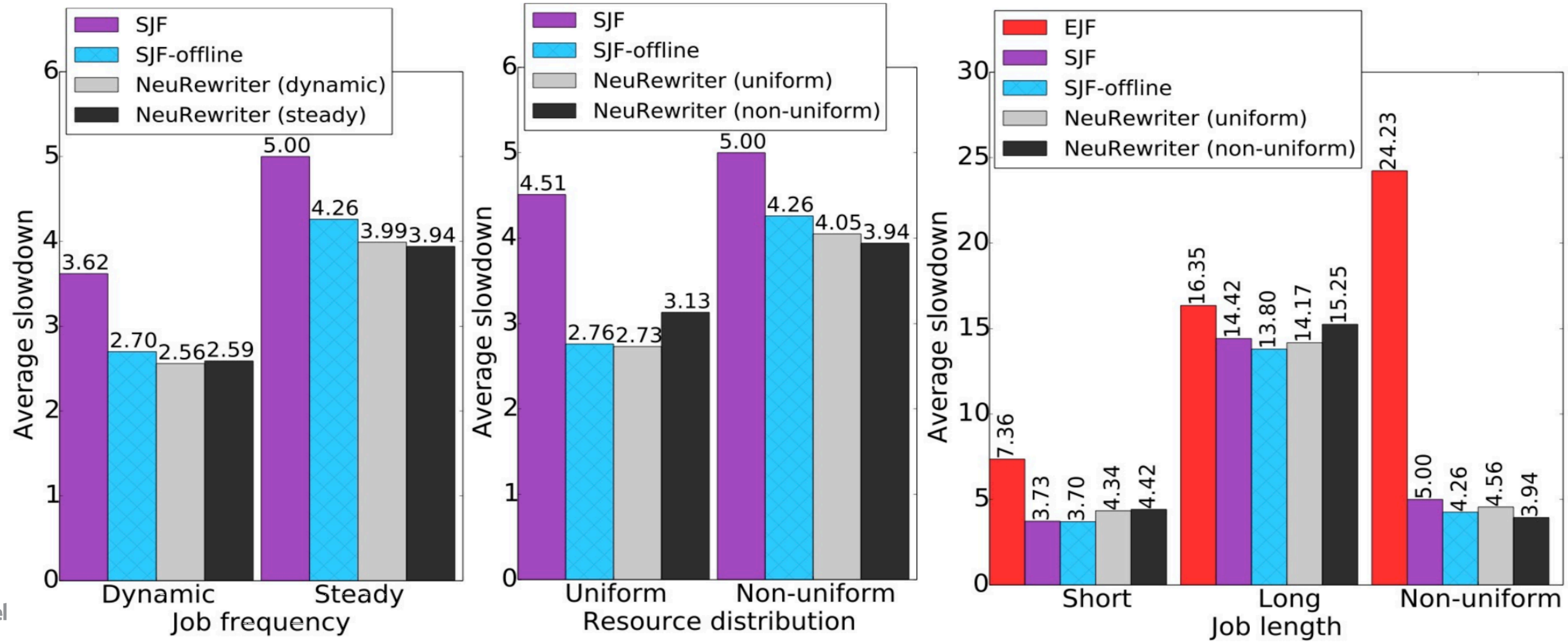
- Earliest Job First (EJF)
- Shortest Job First (SJF)
- Shortest First Search (SJFS)
- DeepRM

Offline baselines:

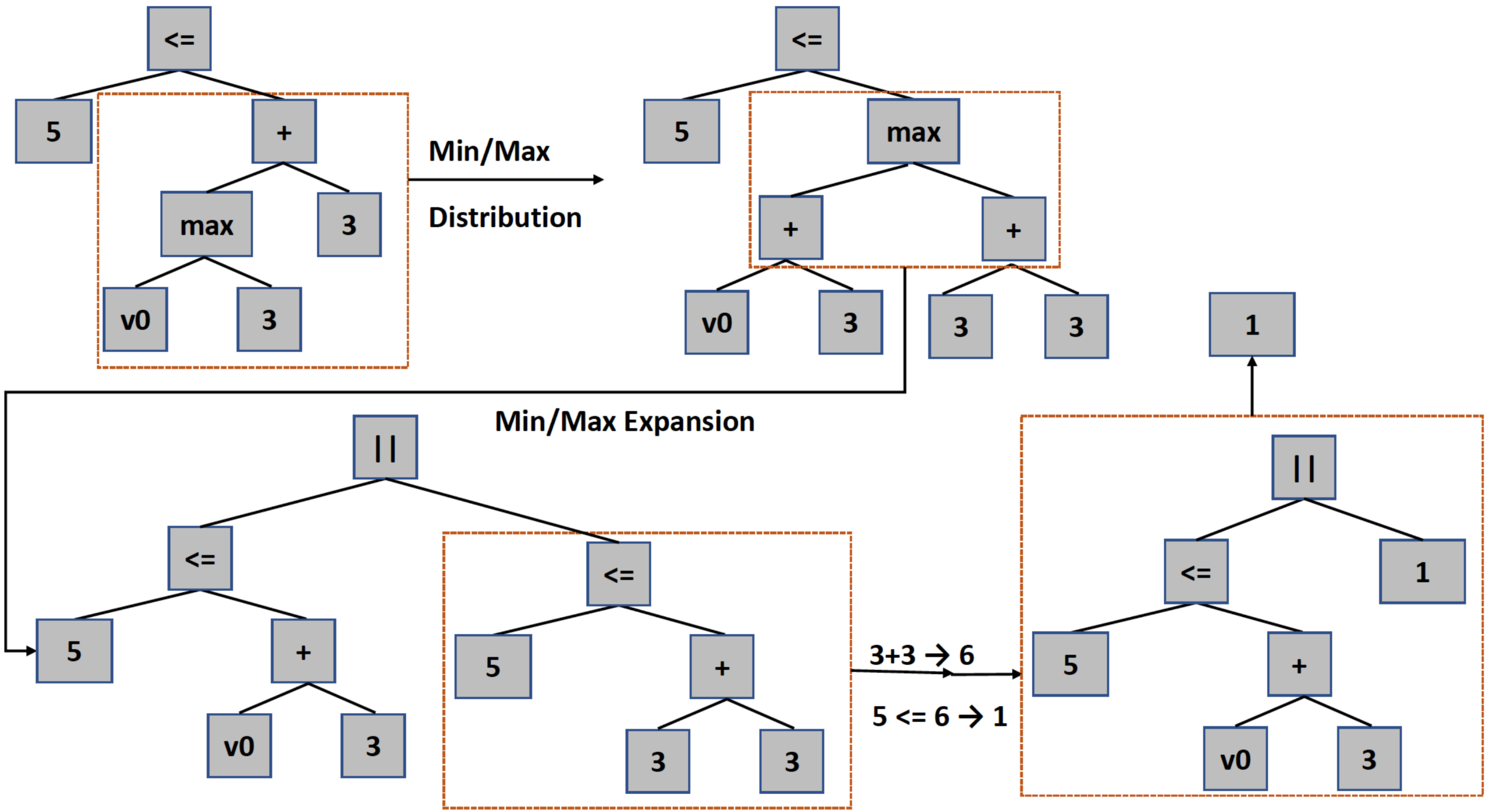
- Google OR-tools (OR-tools)
- SJF-offline

Online Job Scheduling: Ablation Study

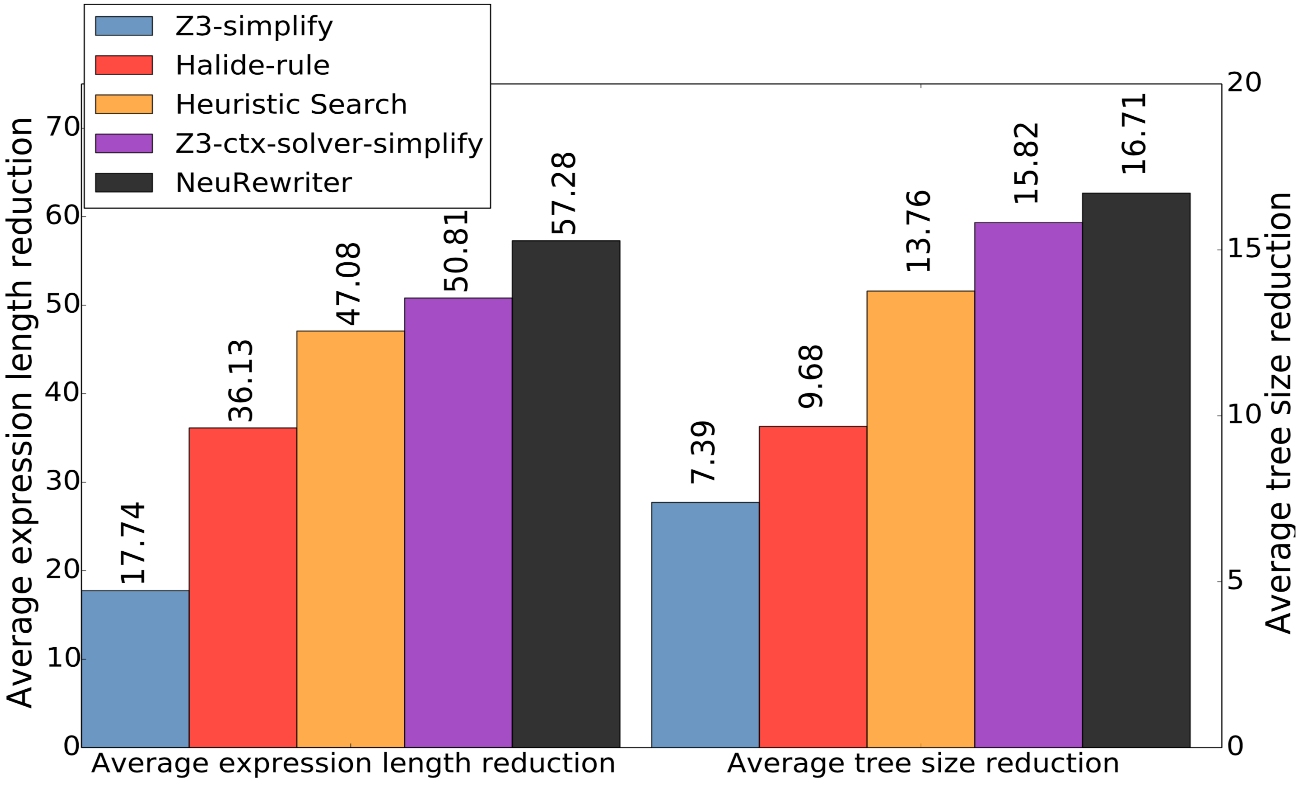
The learned model can generalize to different job distributions.



Expression Simplification



Expression Simplification



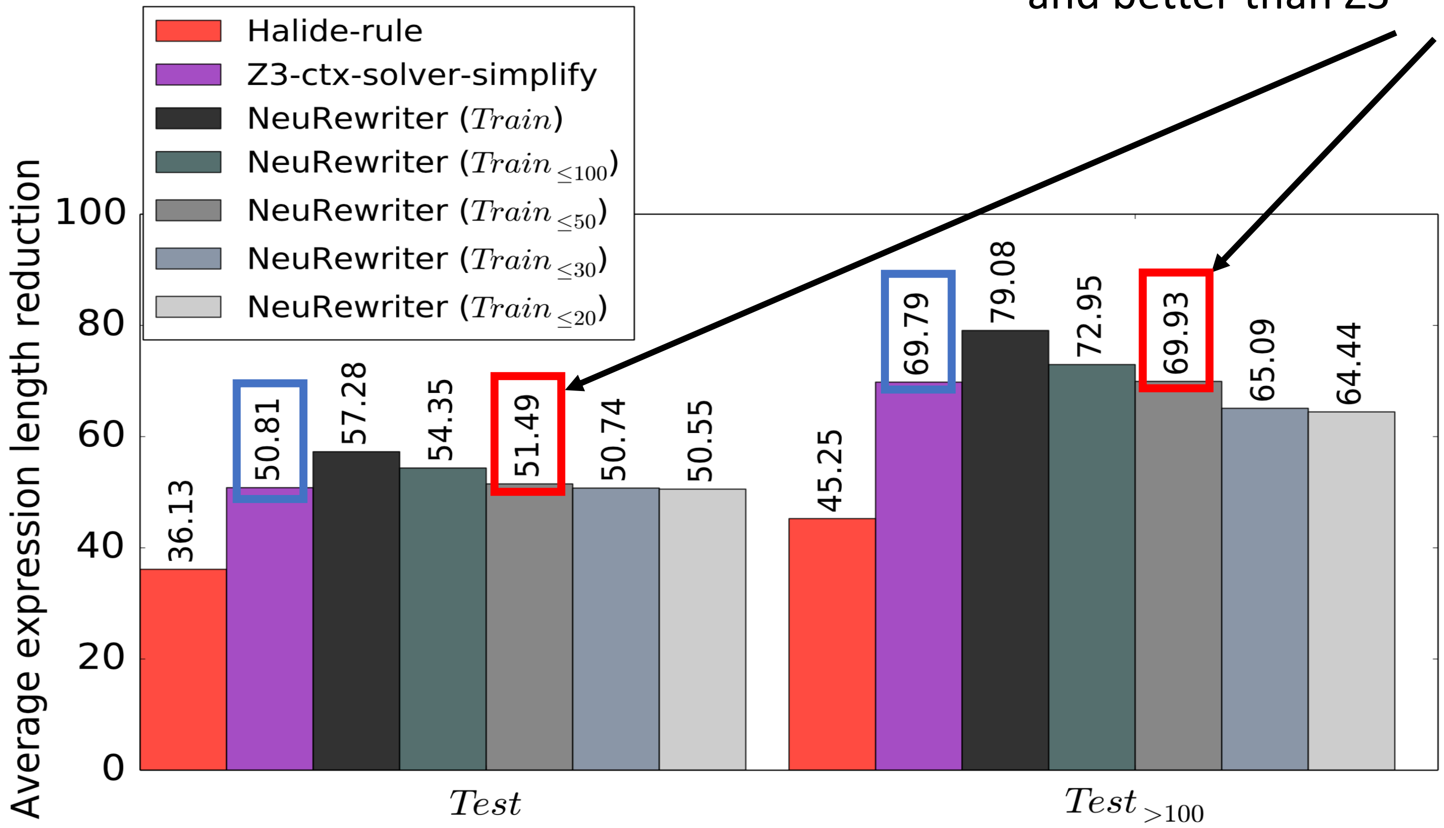
Baselines:

- Z3-simplify
- Z3-ctx-solver-simplify
- Heuristic Search
- Halide rules

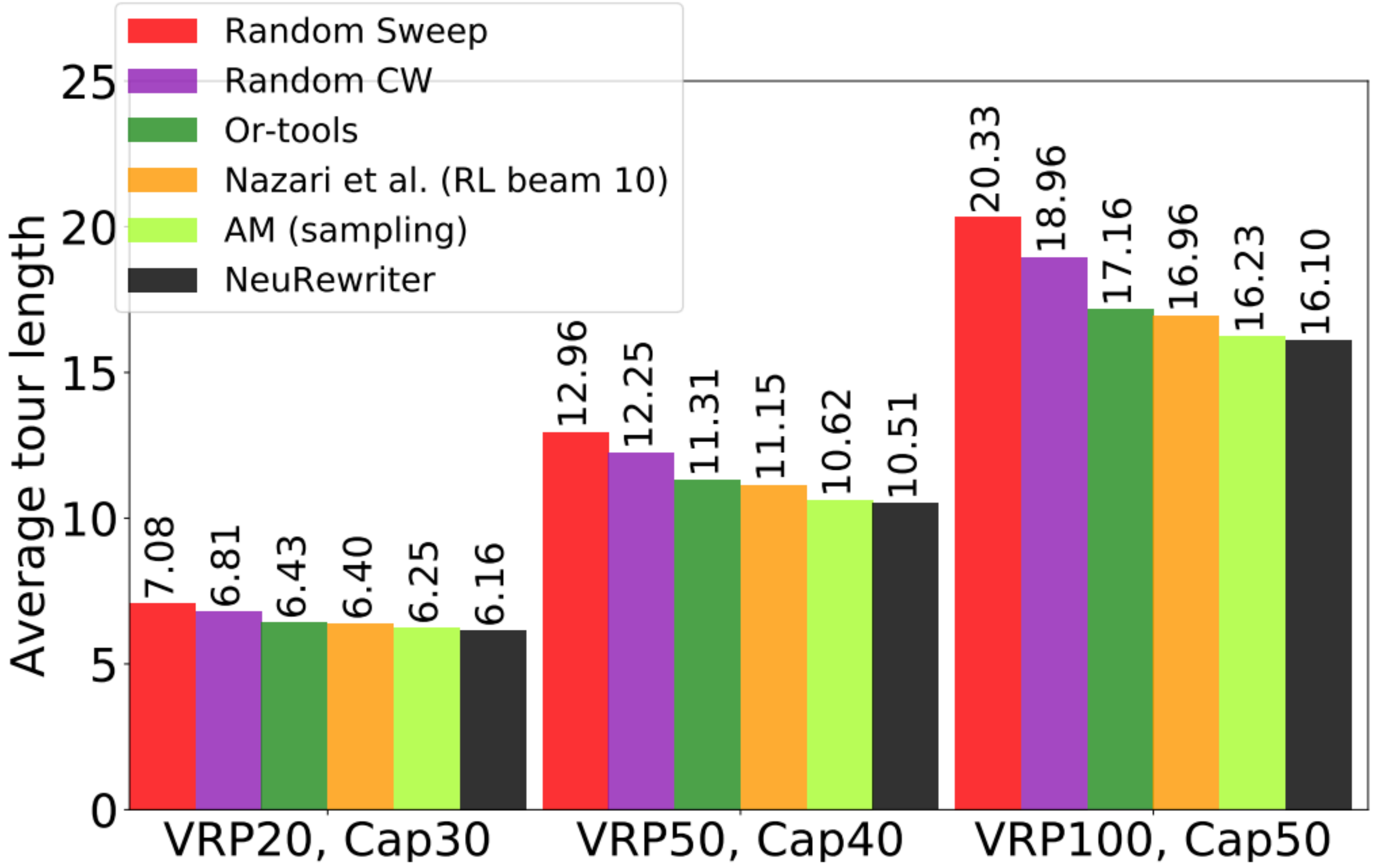
Expression Simplification

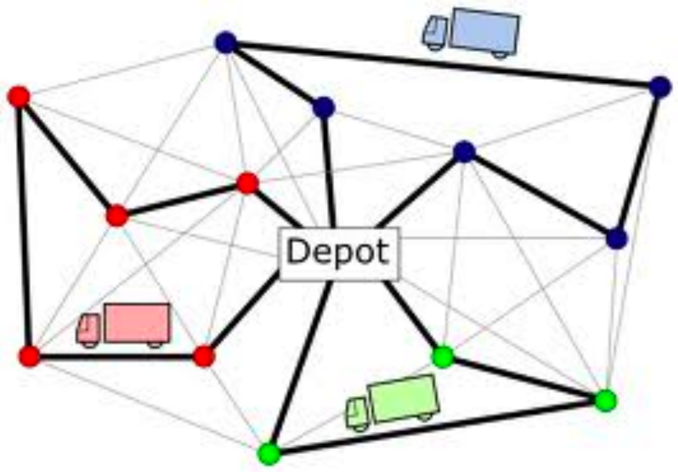
Transfer learning still works well.

A model trained with expression length ≤ 50 has good performance on test set with expression length ≥ 100, and better than Z3



Capacitated Vehicle Routing





fa

Coda: An End-to-End Neural Program Decomplier

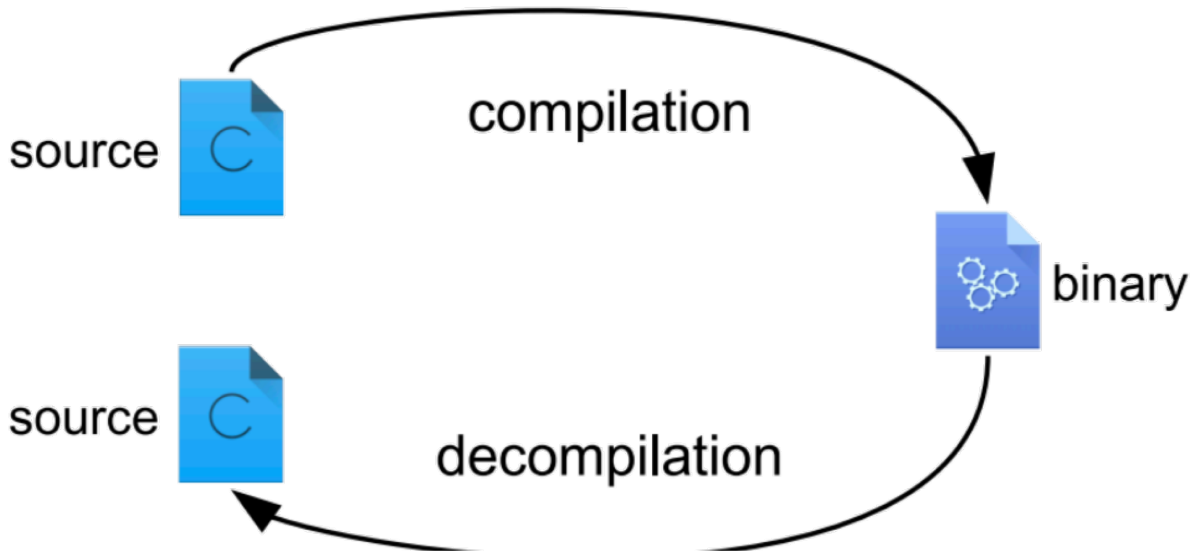
Cheng Fu[1], Huili Chen[1], Haolan Liu[1], Xinyun Chen[3], Yuandong Tian[2], Farinaz Koushanfar[1], Jishen Zhao[1]

[1]UC San Diego, [2]Facebook AI Research, [3]UC Berkeley

NeurIPS 2019

Background: Decompilation

- Goal of Decompilation
  - From Binary Execution to High-level program language



Challenges

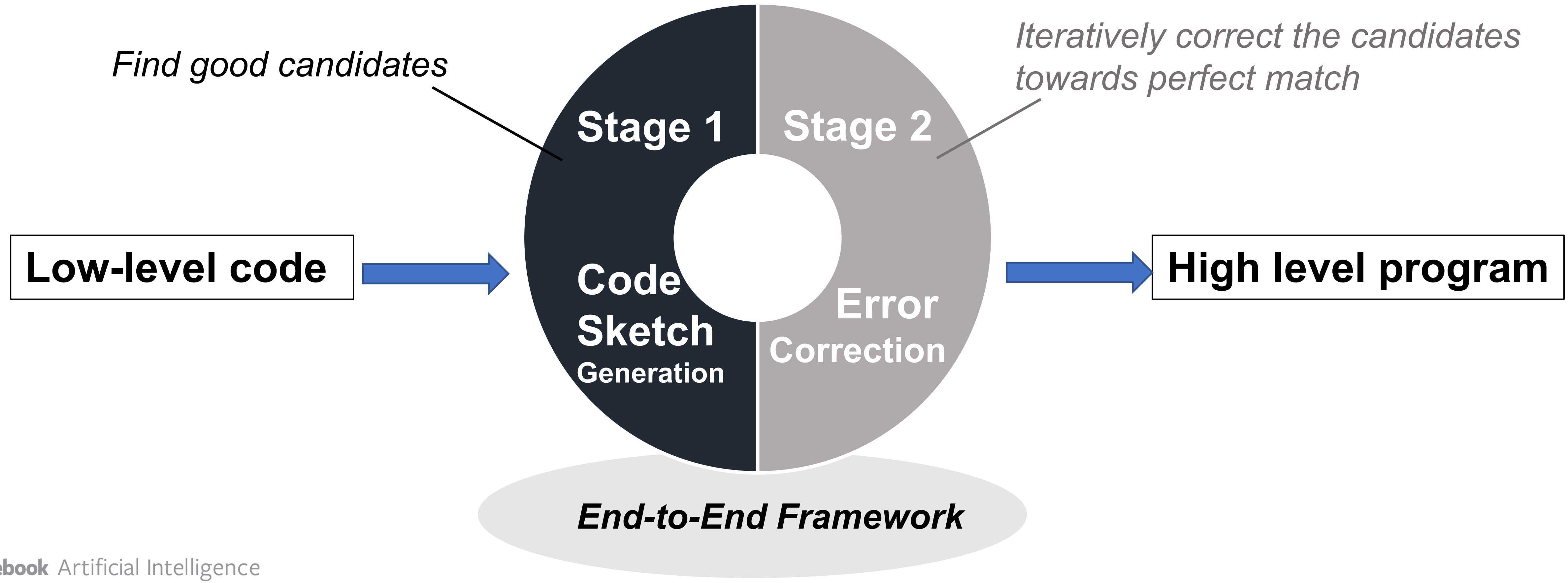
- Many hardware architectures (ISA): x86, MIPS, ARM

- Many Programming Languages (PL)
  - Extra Human effort to extend to the new version of the hardware architectures or programming languages

- Our goals:
  - Maintain both the functionality and semantics of the binary executables
  - Make the design process end-to-end (generalizable to various ISAs and PLs)

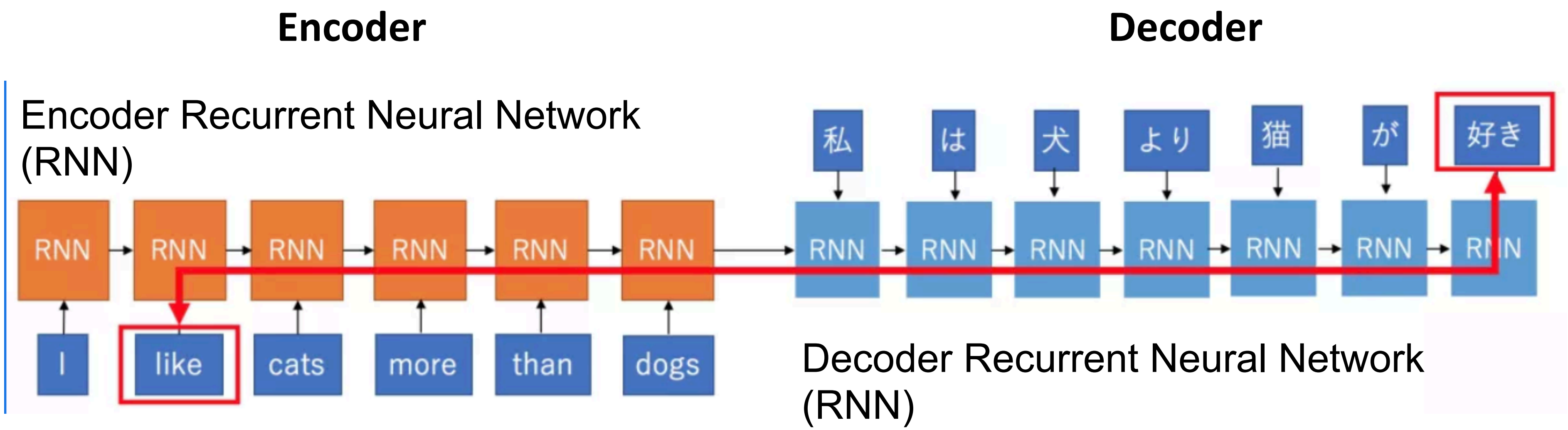
Coda Design

Leverage both syntax and dynamic information



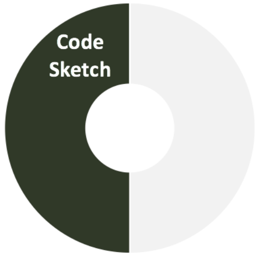
Stage 1: Coda Sketch Generation

- Is Decompilation simply a translation problem?



More than a translation problem!

Stage 1: Coda Sketch Generation

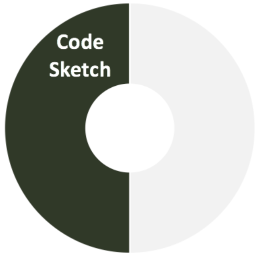


- **Encoder**
  - N-ary Tree Encoder to capture **inter** and **intra** dependencies of the low-level code.
  - Opcode and its operands are encoded together
  - Different encoder is used for different instruction types
    - memory (mem)
    - branch (br)
    - arithmetic (art).

```
# source C program
 a = b * c ;
 if( a > c ){
   c = a * c - b;
 }
```

```
lw     $1, 24($fp)      mem h0
lw     $2, 20($fp)      mem h1
mul    $1, $1, $2       art h2
sw     $1, 28($fp)      mem h3
lw     $1, 28($fp)      mem h4
lw     $2, 20($fp)      mem h5
slt    $1, $2, $1       art h6
beqz   $1, $BB0_3        br h7
j  $B2                   br h8
$B2:                     br h9
lw     $1, 28($fp)      mem h10
lw     $2, 20($fp)      mem h11
mul    $1, $1, $2       art h12
lw     $2, 24($fp)      mem h13
subu   $1, $1, $2       art h14
j      $B3               br h15
sw     $1, 20($fp)      mem h16
```

Stage 1: Coda Sketch Generation



- **Decoder**
  - Generate Abstract Syntax Tree (AST)
  - AST can be equivalently translated into its corresponding high level Program
  - Advantages:
    - Prevent error propagation/ Preserve node dependency / capture PL grammar
    - Boundaries are more explicit (terminal nodes)
  - Using Attention Mechanism

Stage 2: Iterative Error Correction

- The sketch generated in Stage 1 may contain errors:
  - mispredicted tokens, missing lines, redundant lines

Golden program

```
If( a > c ) {
    a = b + c * a;
    b = a – c;
}
```

Wrongly predicted

```
If( a > b ) {
    a = b + c * a;
    b = a - b;
}
```

Missing lines

```
If( a > c ) {
    a = b + c * a;
}
```

Redundant lines

```
If( a > c ) {
    a = b + c * a;
    b = a;
    b = a;
}
```

Stage 2: Iterative Error Correction

- Correct the error by iterative Error Predictor (EP)
  - Iterative rewriting!
  - Spot errors in the generated assembly codes
  - Fix errors and recompile
  - Repeat 10 times

Experimental Setup

- Compiler configuration: Clang **–O0** (no code optimization)
- Benchmarks:
  - Synthetic programs:
    - **Karel library (Karel)** – only function calls
    - **Math library (Math)** – function calls with arguments
    - **Normal expressions (NE)** – (^,&,*,-,<<,>>,|,% ....)
    - **Math library + Normal expressions (Math+NE)** – replaces the variables in NE with a return value of math function.
- Metrics:
  - Token Accuracy
  - Program Accuracy

Result – Stage 1 Performance

- Token accuracy (%) across benchmarks

Benchmarks	Seq2Seq	Seq2Seq+Attn	Seq2AST+Attn	Inst2seq+Attn	Inst2AST+Attn
Karel_S	51.61	97.13	99.81	98.83	**99.89**
Math_S	23.12	94.85	99.12	96.20	**99.72**
NE_S	18.72	87.36	90.45	88.48	**94.66**
$(\text{Math+NE})_S$	14.14	87.86	91.98	89.67	**97.90**
Karel_L	33.54	94.42	98.02	98.12	**98.56**
Math_L	11.32	91.94	96.63	93.16	**98.63**
NE_L	11.02	81.80	85.92	85.97	**91.92**
$(\text{Math+NE})_L$	6.09	81.56	85.32	86.16	**93.20**

- Highest token accuracy across all benchmarks (96.8% on average) compared to baselines.
- 10.1% and 80.9% margin over a naive Seq2Seq model with and without attention.
- More tolerant to the growth of program length.

Result – Stage 2 Performance

- Program accuracy (%)

BenchMarks	(i) Error Detection		(ii) Befor EC		After EC	
	$s2s$,10	$i2a$,10	$s2s$	$i2a$	$s2s$	$i2a$
Math_S	91.4	94.2	40.1	64.8	91.2	**100.0**
NE_S	83.5	88.7	6.6	12.2	53.0	**78.6**
$(\mathrm{Math+NE})_S$	83.6	90.1	3.5	43.2	63.6	**89.2**
Math_L	87.5	91.3	21.7	51.8	83.9	**99.5**
NE_L	78.1	84.5	0.2	2.6	33.1	**56.4**
$(\mathrm{Math+NE})_L$	80.2	85.3	0.1	4.9	38.3	**67.2**

Baseline

Ours

s2s = sequence-to-sequence with attention

I2a = instruction encoder to AST decoder with attention

Summarization and Future Works

- Summary
  - Gives examples of scalable RL system
  - RL/ML can be used to learn heuristics for system

- Large Open Space Ahead
  - ML captures statistics regularity and leads to better solutions
  - Application to large-scale systems?
  - Theoretical Guarantees?

Thanks!