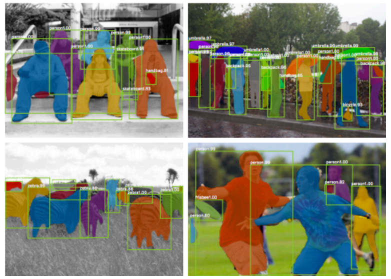
AI in Games: Achievements and Challenges

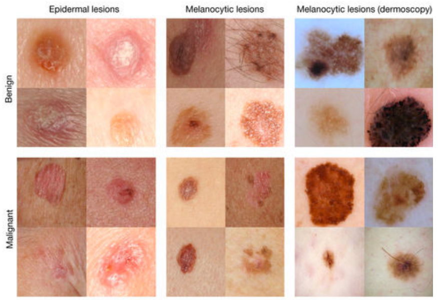
Yuandong Tian

Facebook AI Research

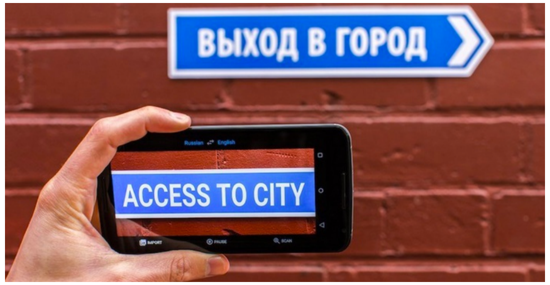
AI works in a lot of situations



Object Recognition

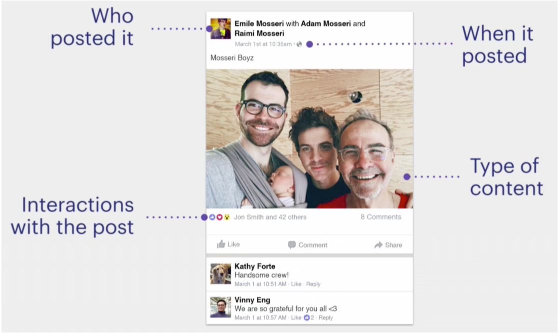


Medical



Translation

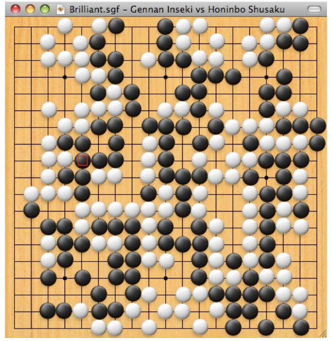
Speech Recognition



Personalization

Surveillance

Smart Design

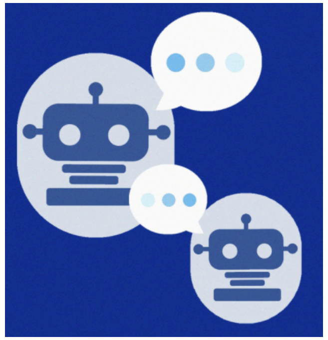


Board game

What AI still needs to improve

Home Robotics

Autonomous Driving



ChatBot

StarCraft

Question Answering

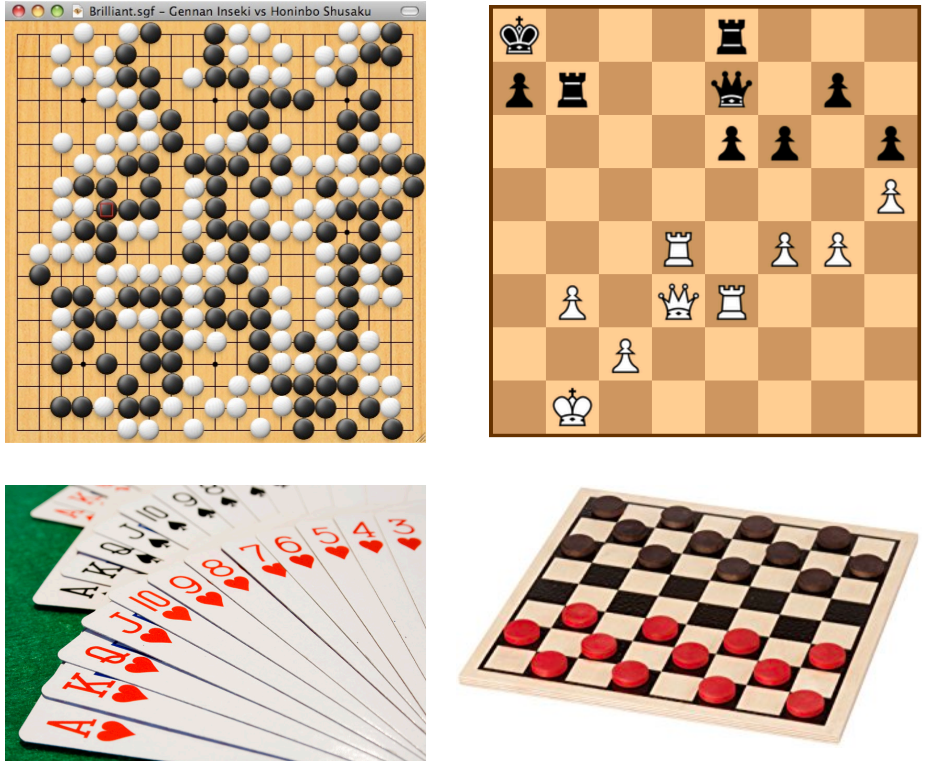
Exponential space to explore

Very few supervised data

Complicated/unknown environments with lots of corner cases.

Common Sense

The Charm of Games



Complicated long-term strategies.

Realistic Worlds

Game as a Vehicle of AI

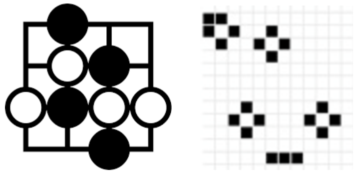
Infinite supply of *fully* labeled data

Controllable and replicable

Low cost per sample

Faster than real-time

Less safety and ethical concerns

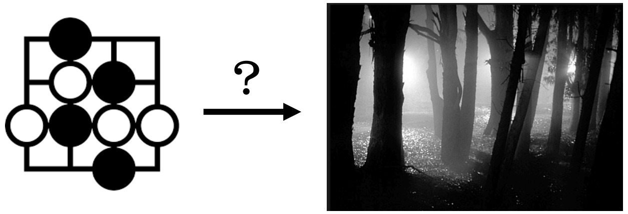


Complicated dynamics with simple rules.

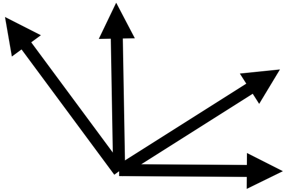
Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Require a lot of resources.



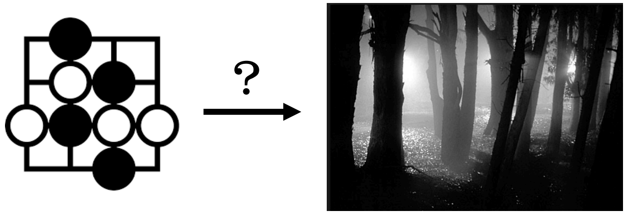
Abstract game to real-world



Hard to benchmark the progress

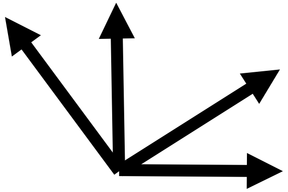
Game as a Vehicle of AI

Algorithm is slow and data-inefficient



Abstract game to real-world

Require a lot of resources.



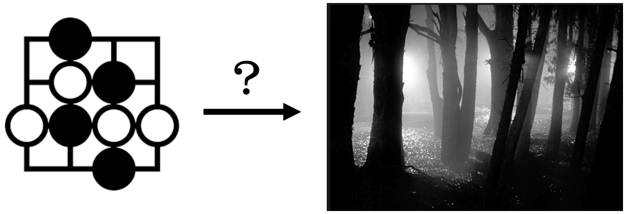
Hard to benchmark the progress

Better Algorithm/System

Better Environment

Game as a Vehicle of AI

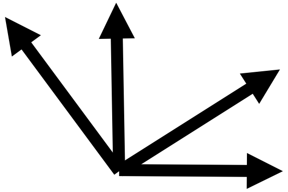
Algorithm is slow and data-inefficient



Abstract game to real-world

Better Algorithm/System

Require a lot of resources.



Hard to benchmark the progress

Better Environment

Our work

Better Environment

ELF

ELF: Extensive Lightweight and Flexible Framework
(Yuandong Tian et al, NIPS17)

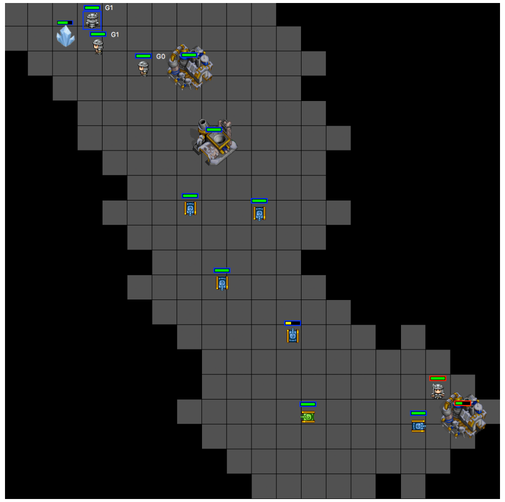
House3D: An interactive 3D environment
for navigation
(Yi Wu, Georgia Gkioxari, Yuxin Wu, Yuandong Tian)

Our work

Better Algorithm/System

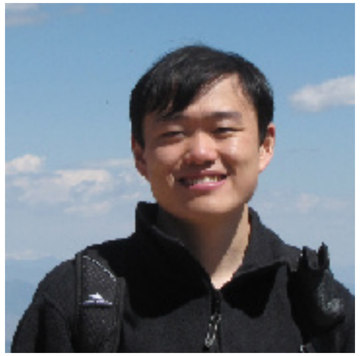
DarkForest Go Engine
(Y. Tian, Y. Zhu, ICLR16)

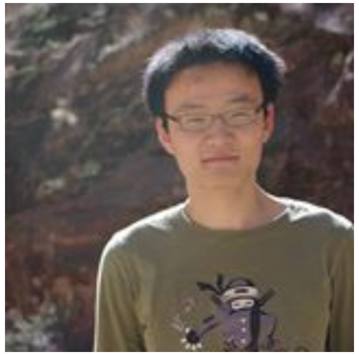
Doom AI
(Yuxin Wu, Y. Tian, ICLR17)

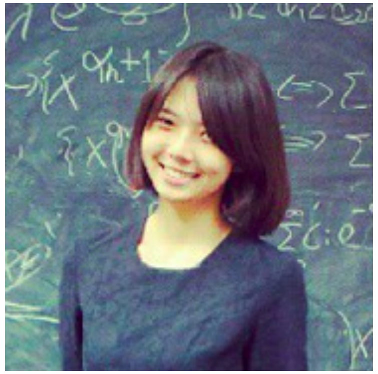


MiniRTS
(Y. Tian, Q. Gong, W. Shang)

ELF: **E**xtensive, **L**ightweight and **F**lexible Framework for Game Research


Yuandong Tian


Qucheng Gong

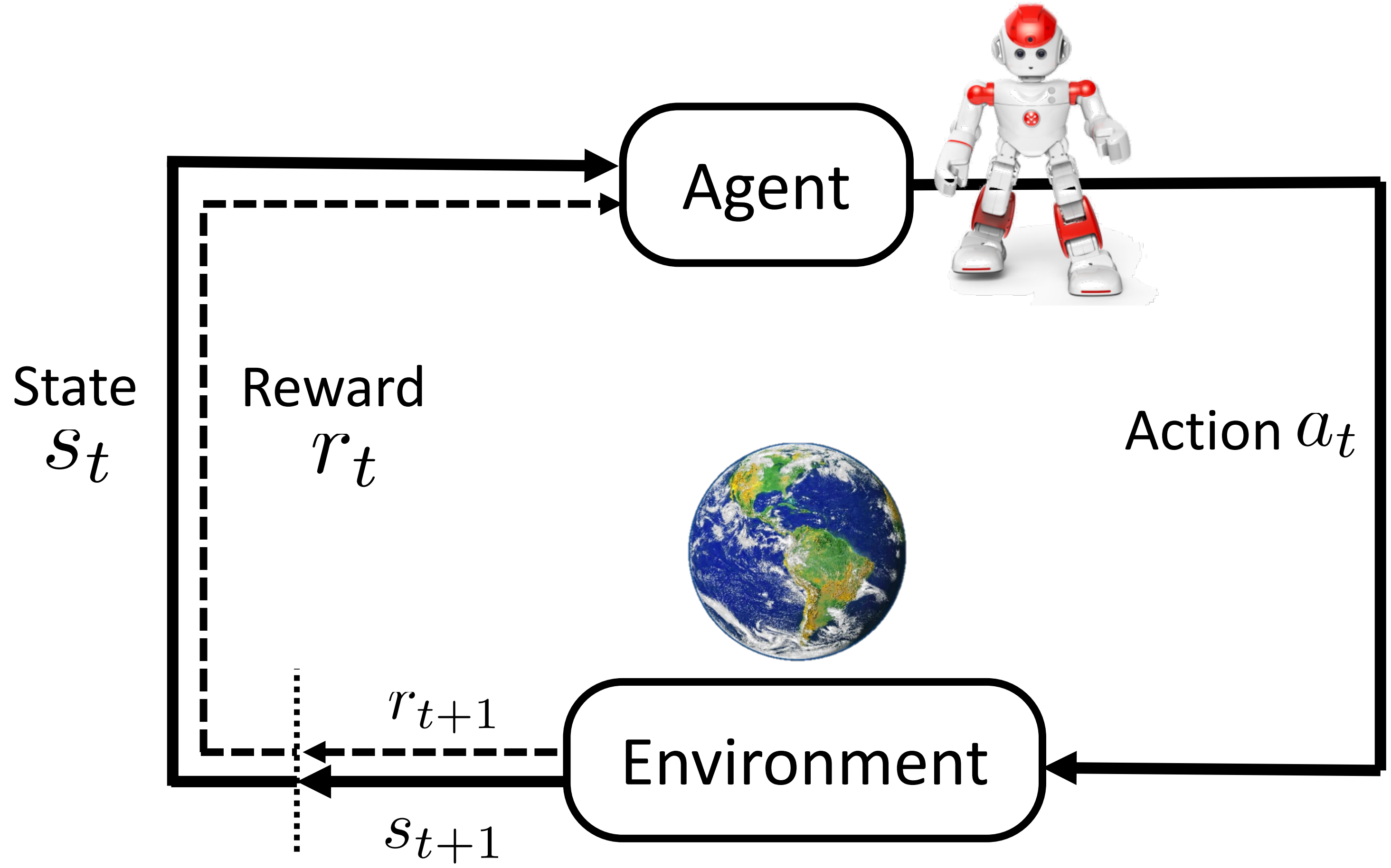

Wenling Shang


Yuxin Wu


Larry Zitnick

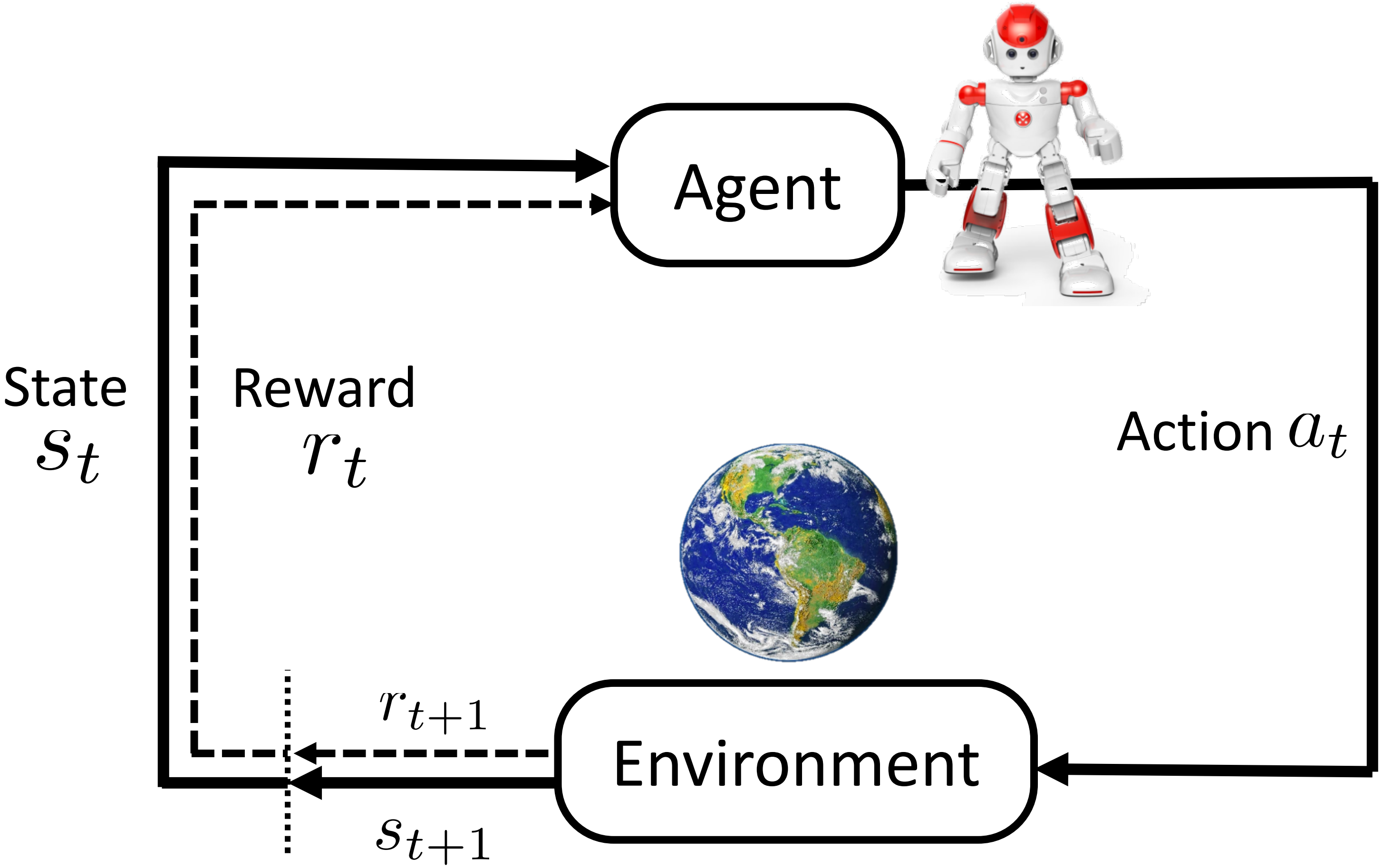
Facebook AI Research

Reinforcement Learning: Ideal and Reality



[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]

Reinforcement Learning: Ideal and Reality

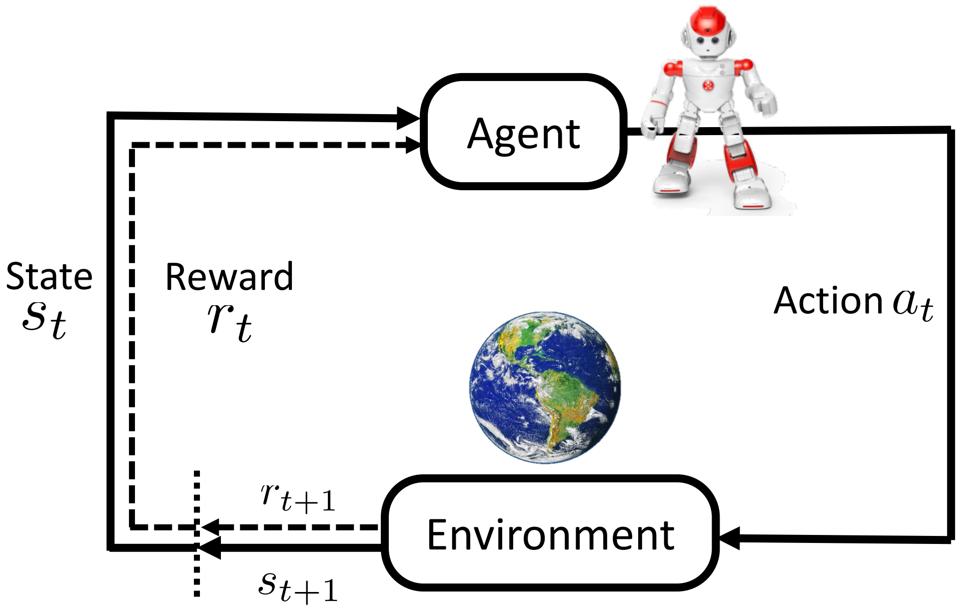


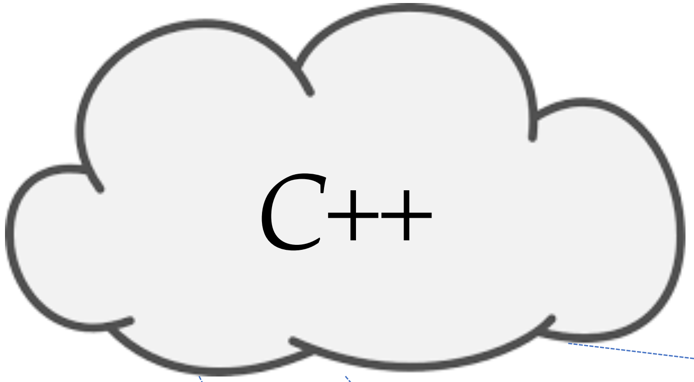
Design Choices:

CPU, GPU?
Simulation, Replays
Concurrency

[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]

ELF: A simple for-loop





Python

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

ELF Characteristics

Extensive

Any games with C++ interfaces
can be incorporated.

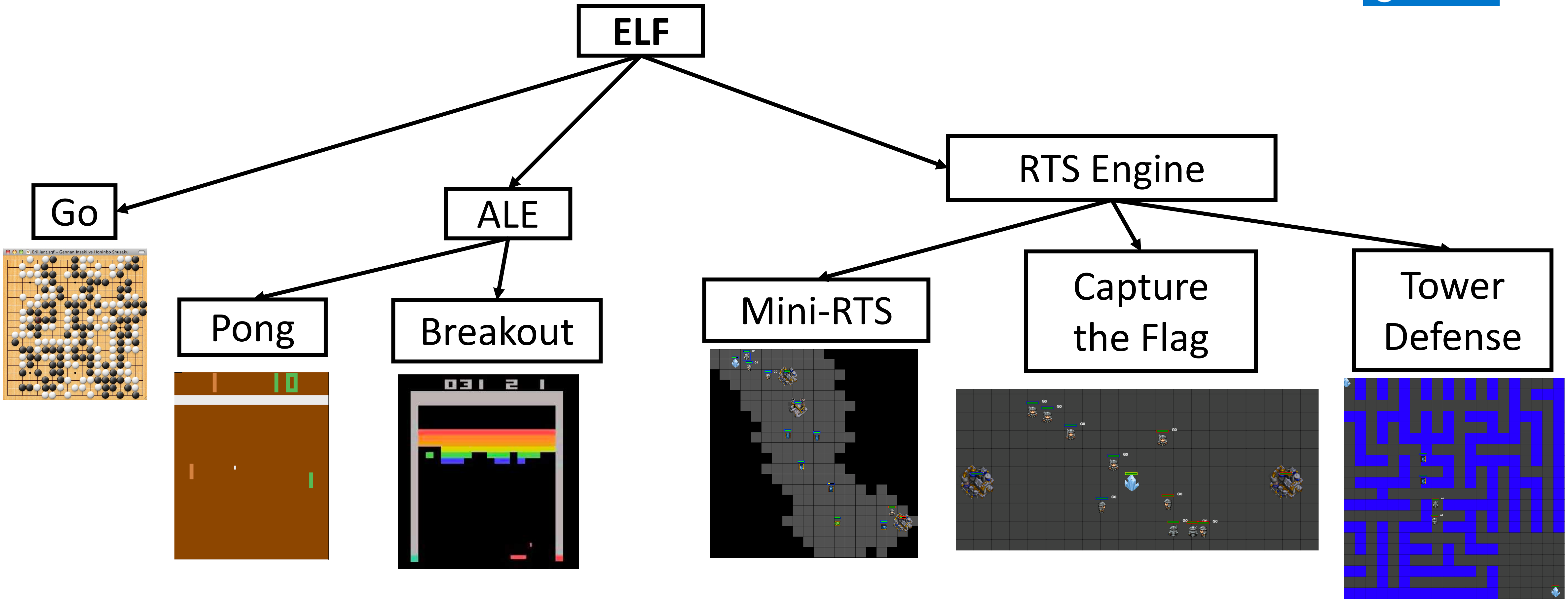
Flexible

Environment-Actor topology
Parametrized game environments.
Choice of different RL methods.

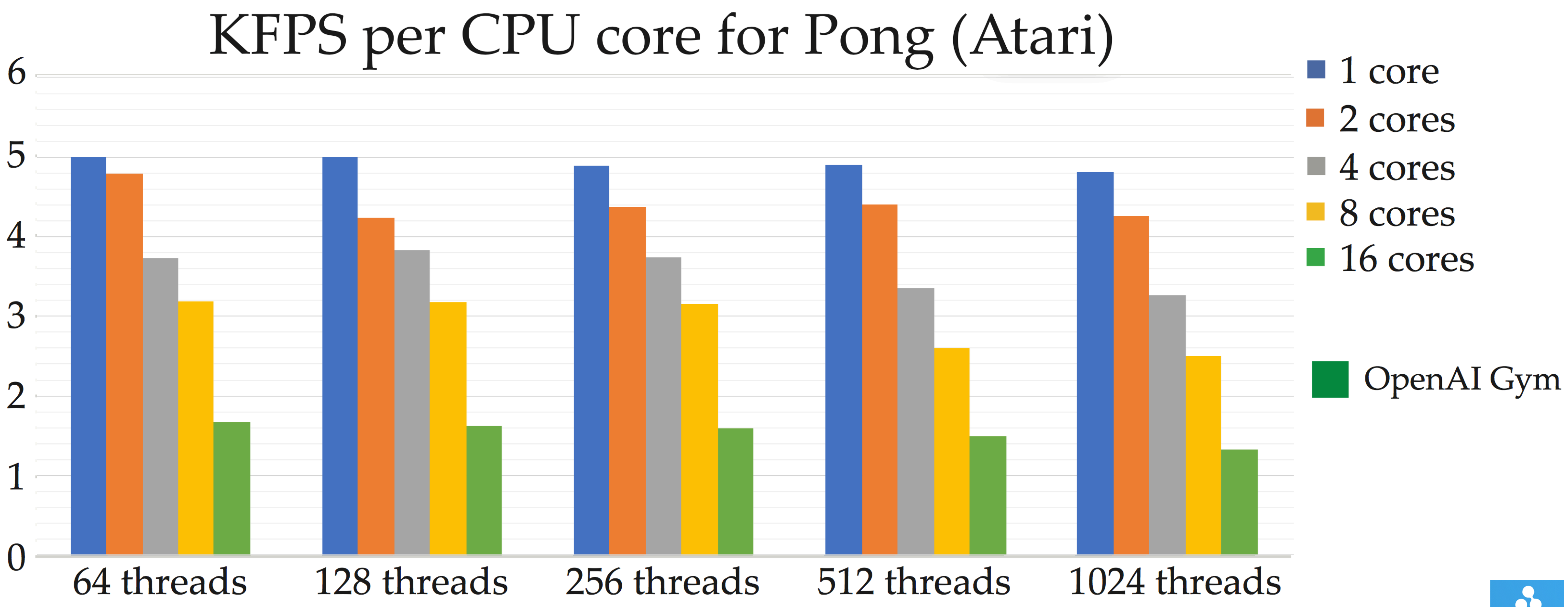
Lightweight

Fast. Mini-RTS (40K FPS per core)
Minimal resource usage (1GPU+several CPUs)
Fast training (half a day for a RTS game)

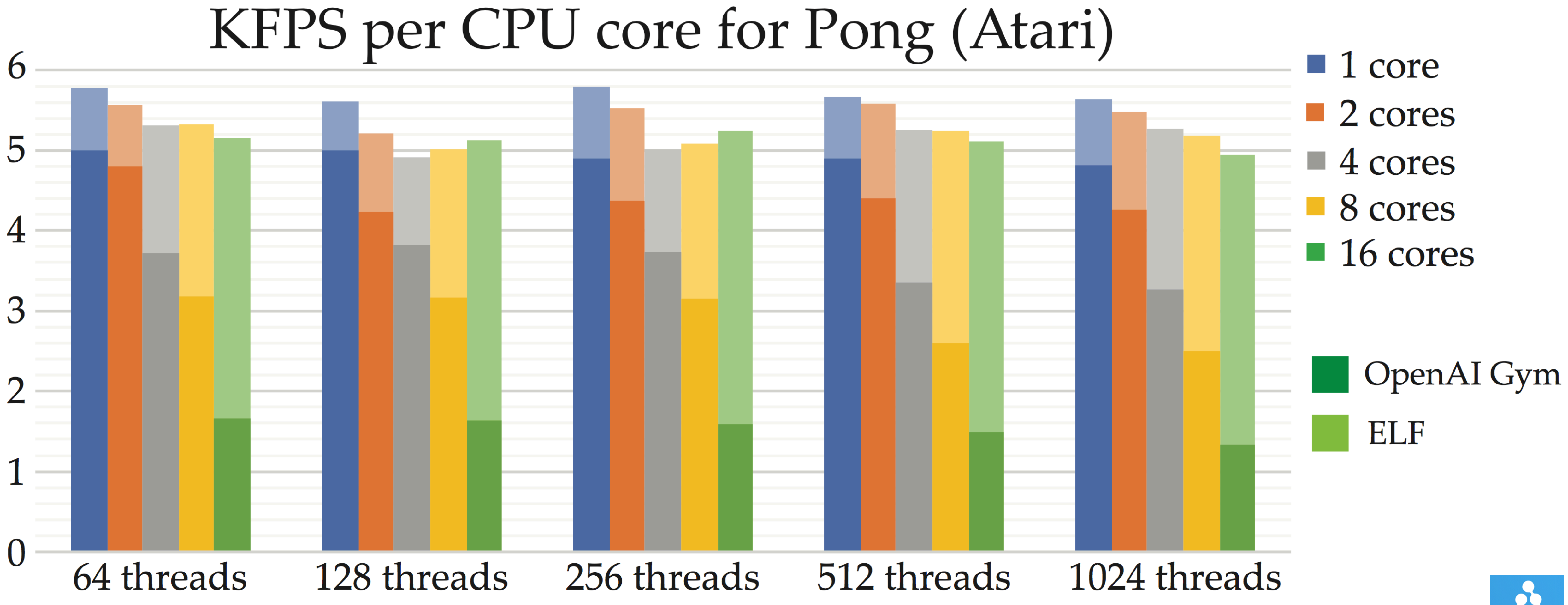
Extensibility



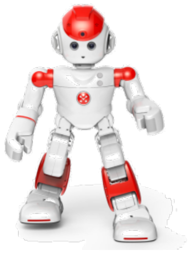
Lightweight

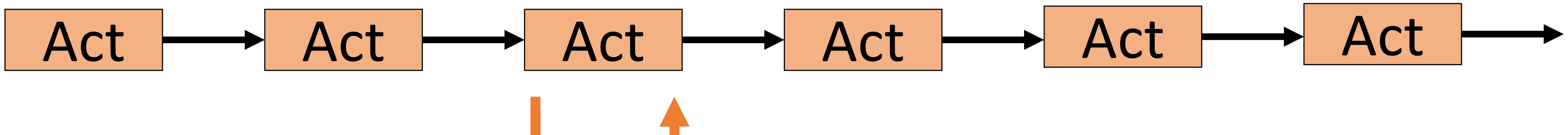


Lightweight



Flexibility

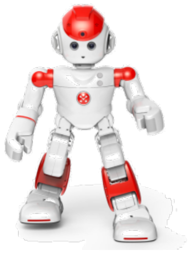


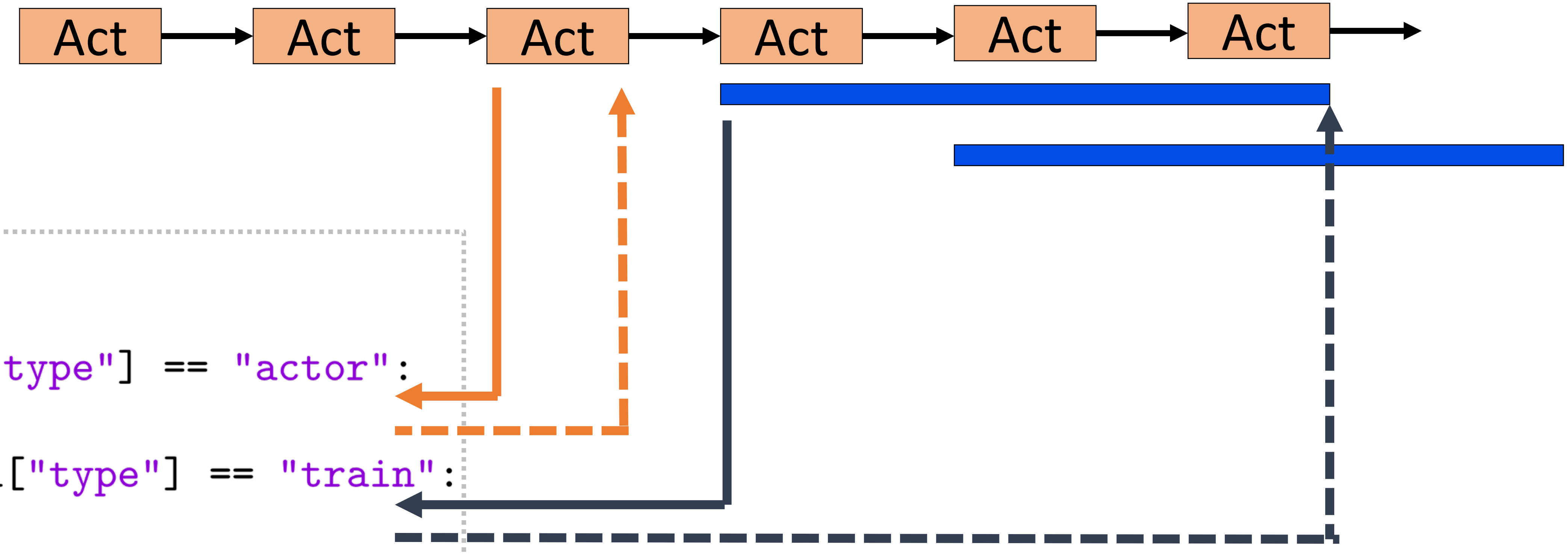



```python
while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
```

Evaluation

Flexibility

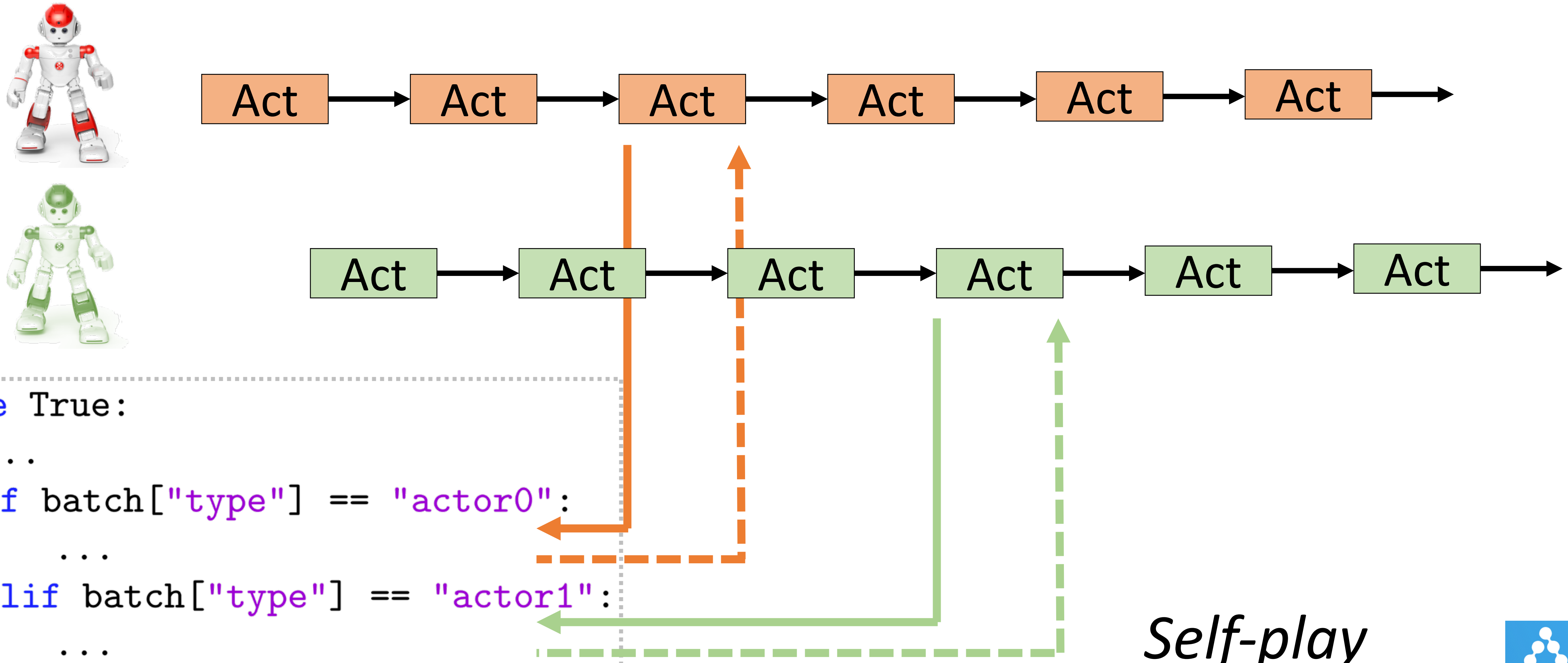





```python
while True:
    ...
    if batch["type"] == "actor":
        ...
    elif batch["type"] == "train":
        ...
```

Training

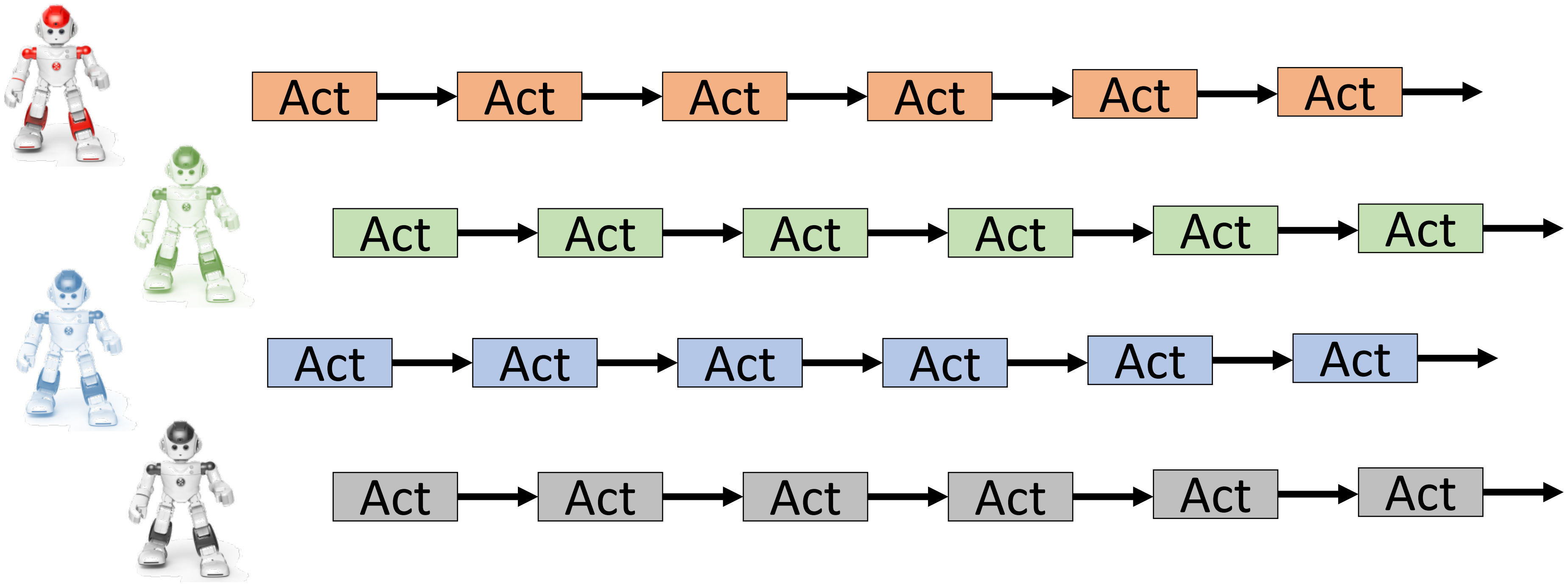
Flexibility




```
while True:
    ...
    if batch["type"] == "actor0":
        ...
    elif batch["type"] == "actor1":
        ...
```

Self-play

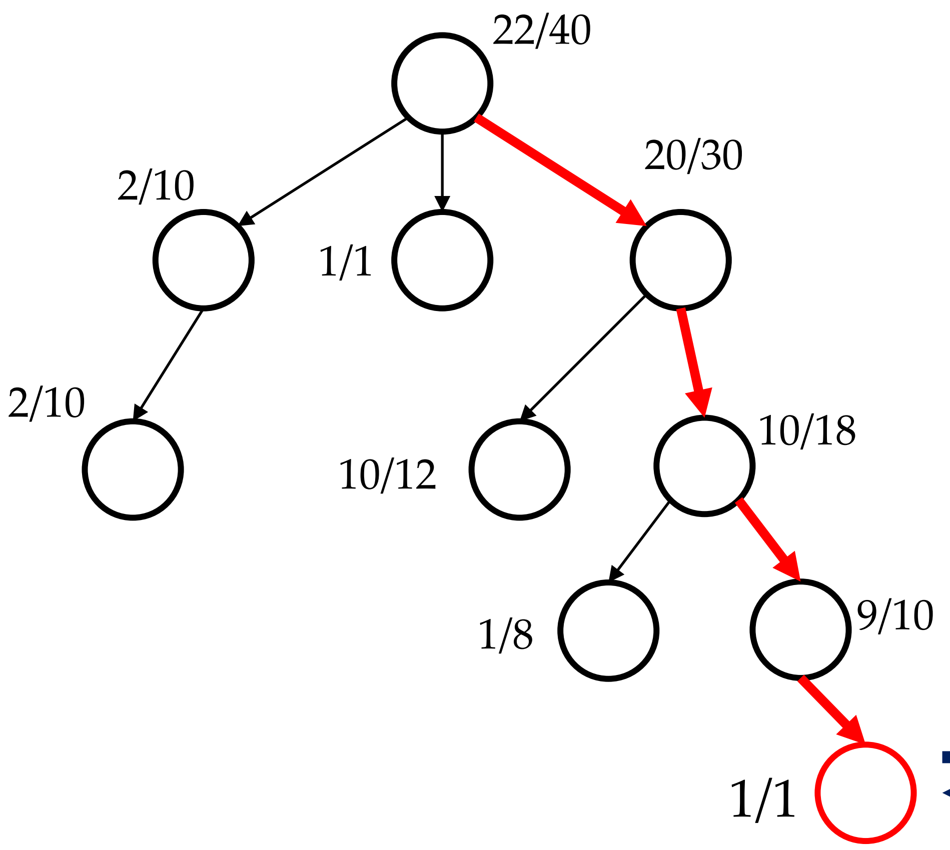
Flexibility




```python
while True:
    ...
    for i in range(n):
        if batch["type"] == "actor%d" % i:
            ...
```

Multi-agent

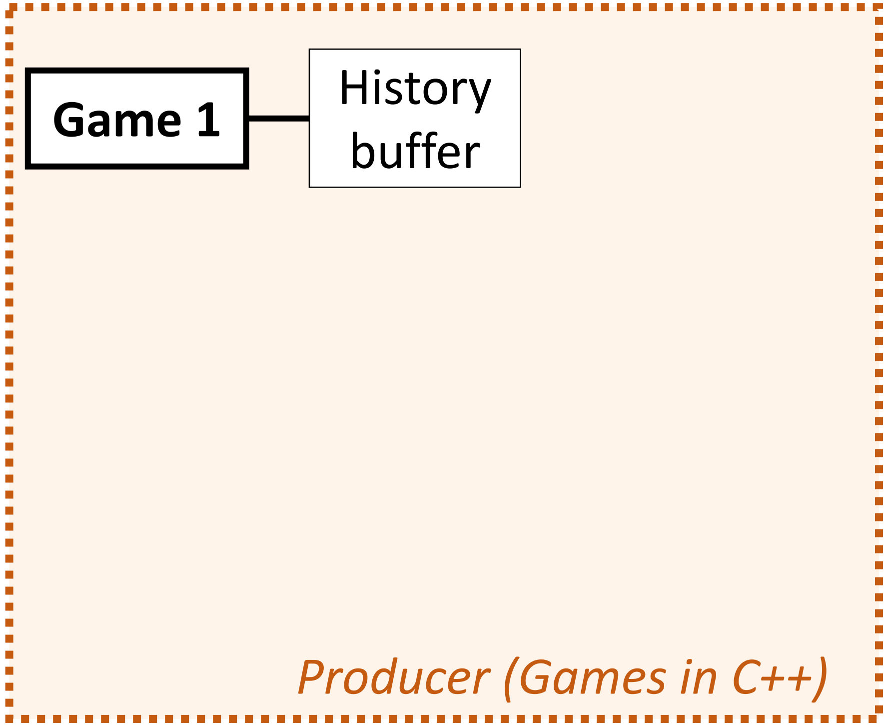
Flexibility



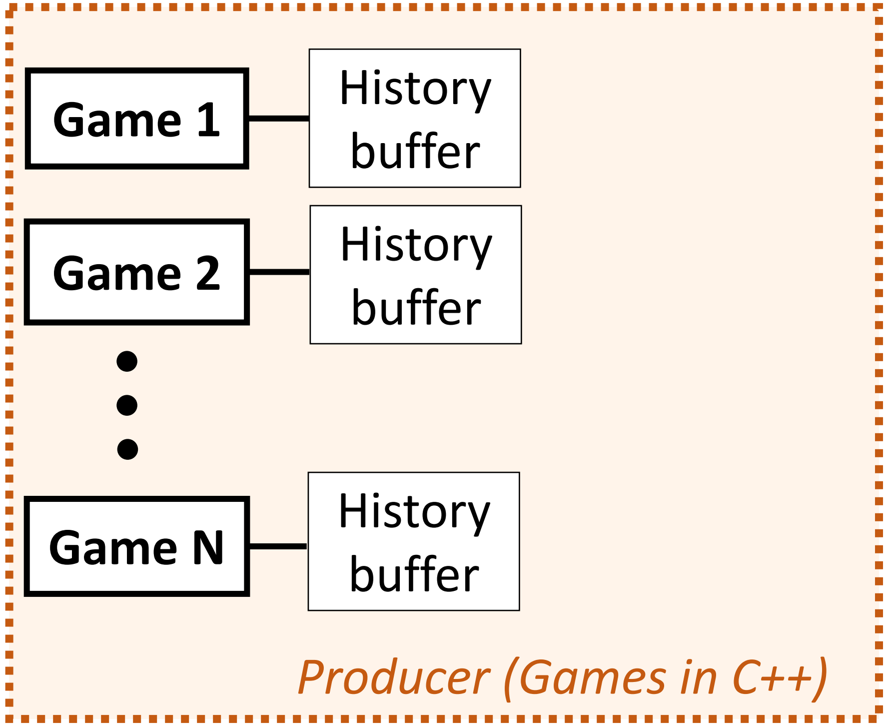

```python
while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
```

Monte-Carlo Tree Search

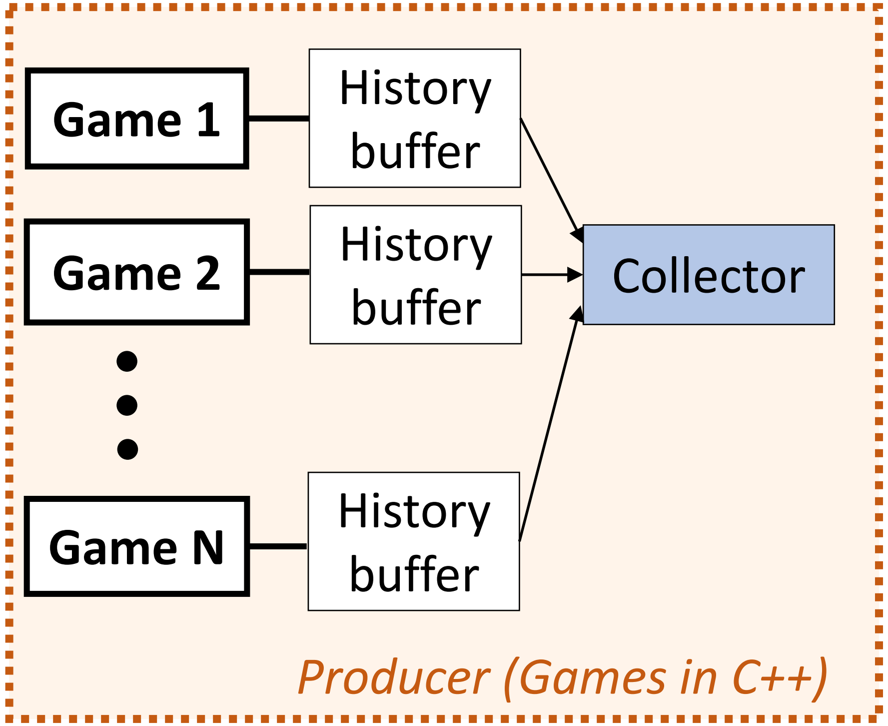
ELF design



ELF design

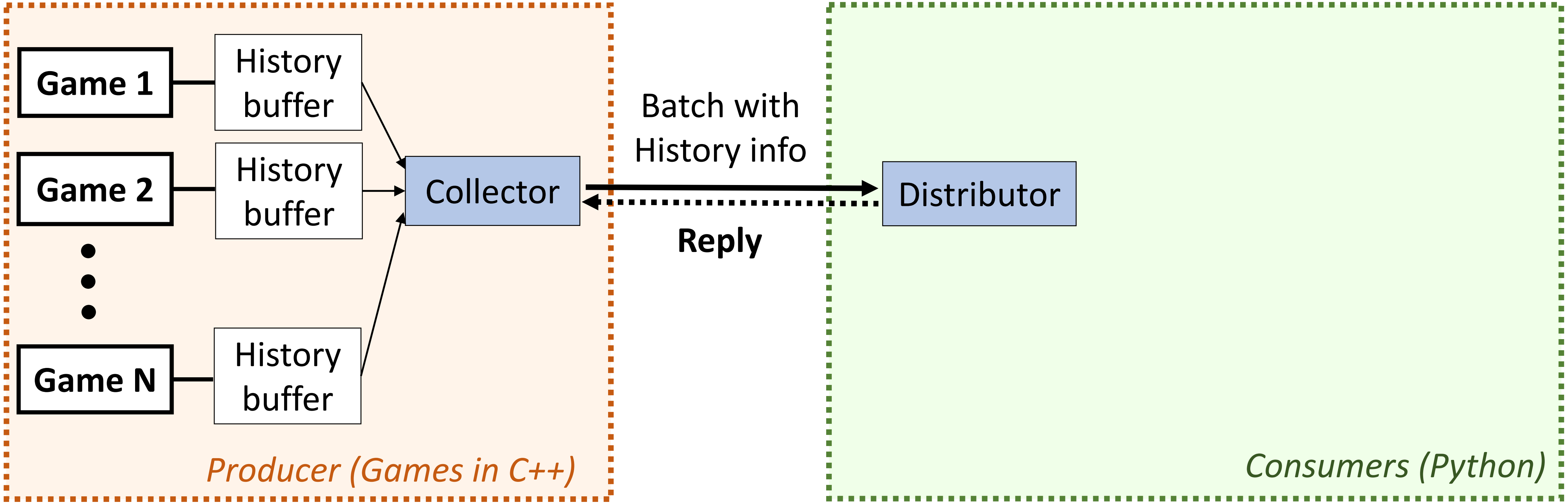

Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Producer (Games in C++)

ELF design



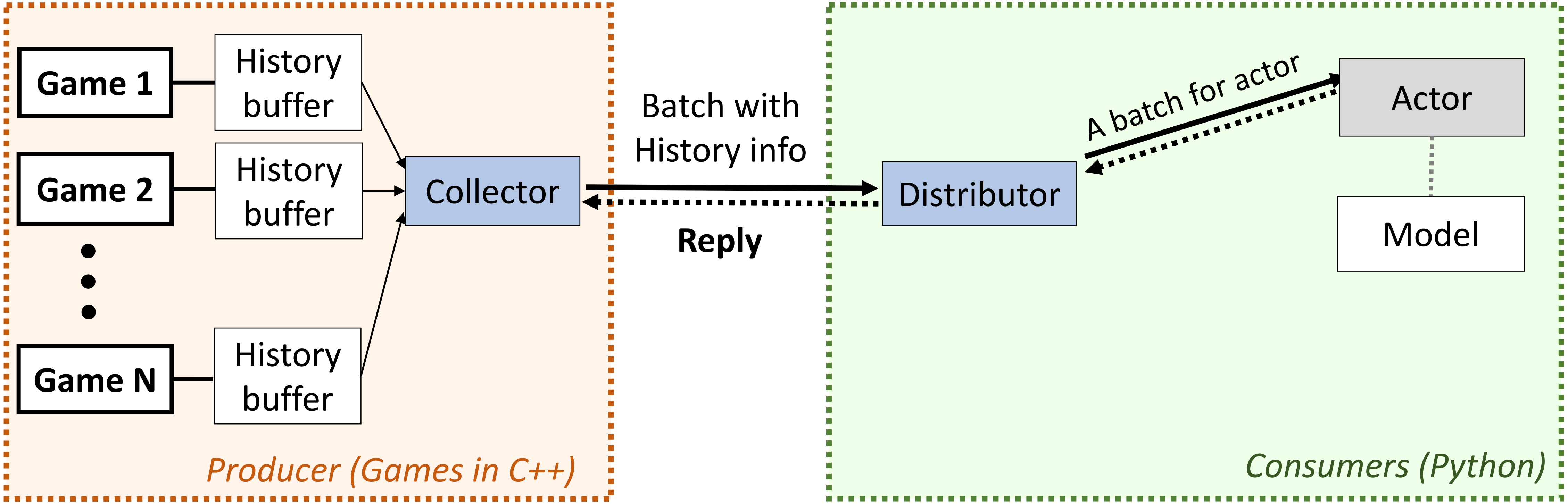
Game 1
History buffer
Game 2
History buffer
Collector
Game N
History buffer
Producer (Games in C++)

ELF design

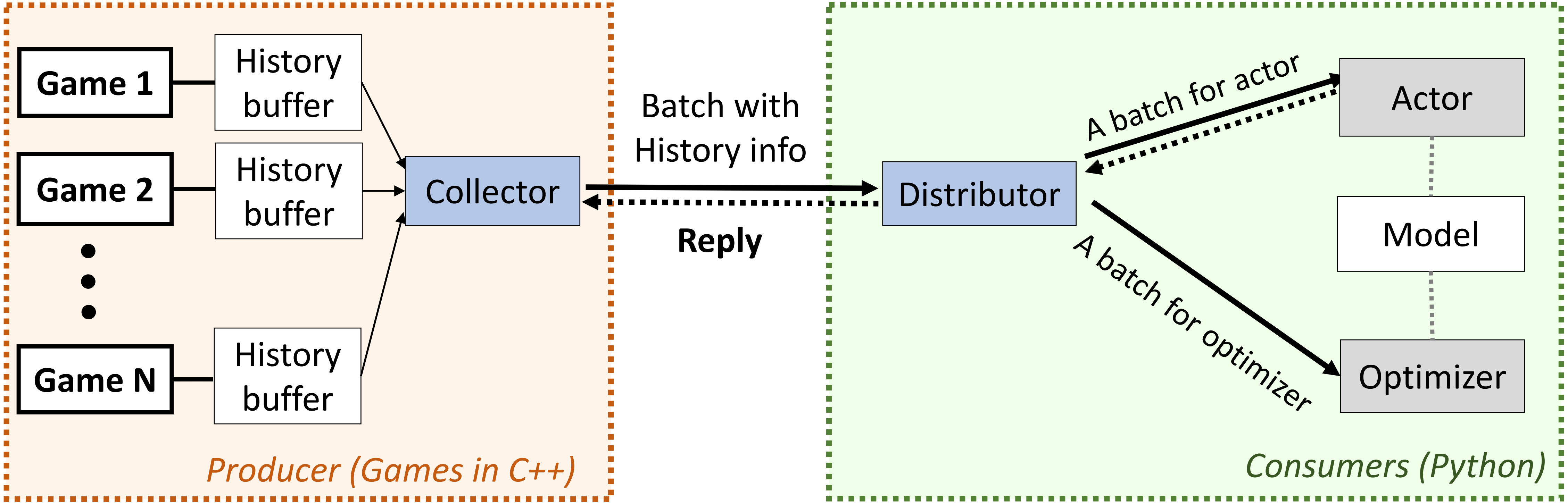


Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
Consumers (Python)

ELF design

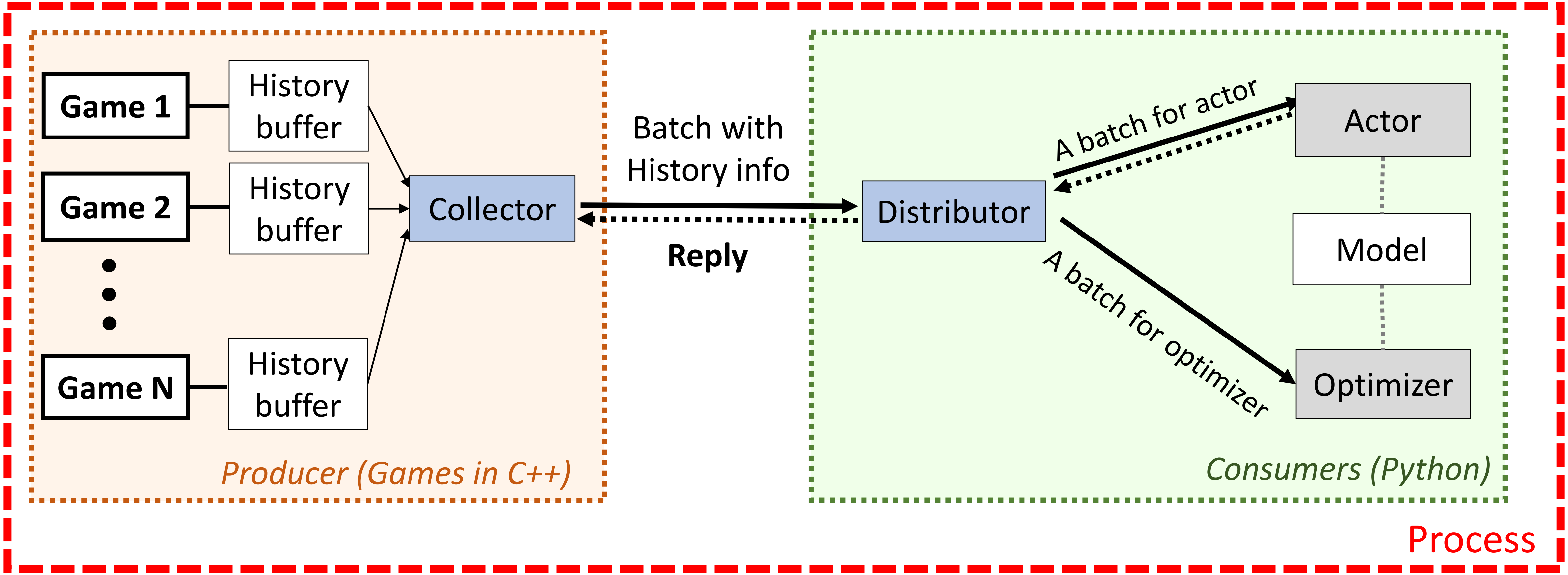

Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
A batch for actor
Actor
Model
Consumers (Python)

ELF design

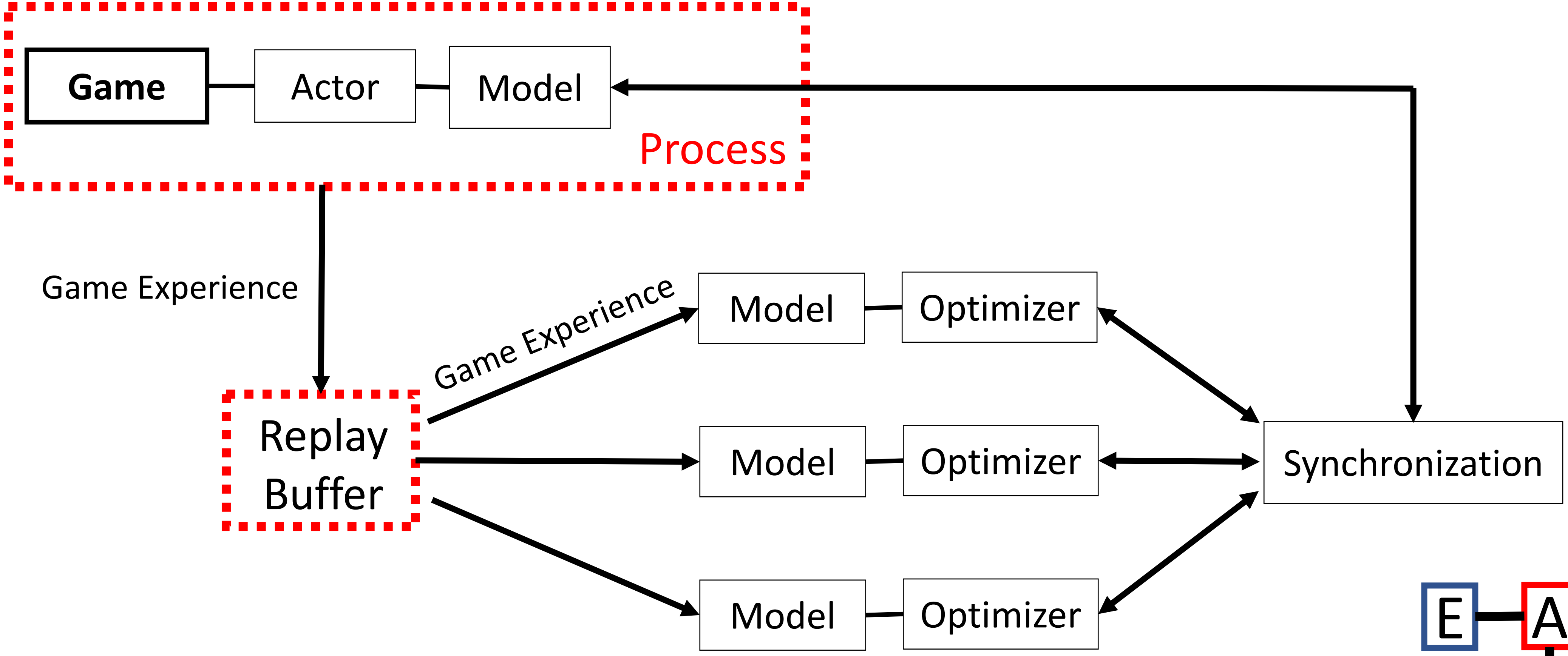


Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
A batch for actor
A batch for optimizer
Actor
Model
Optimizer
Consumers (Python)

ELF design

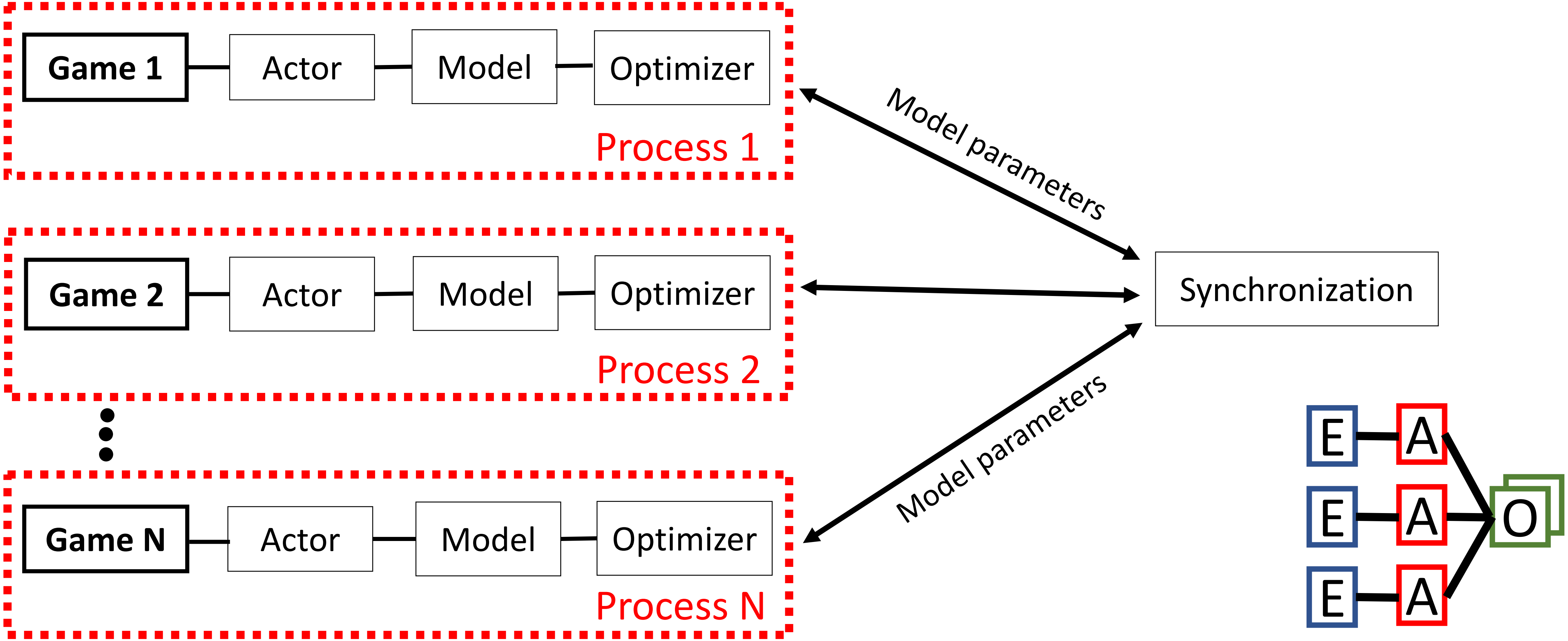

Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
A batch for actor
A batch for optimizer
Actor
Model
Optimizer
Consumers (Python)
Process

Gorilla



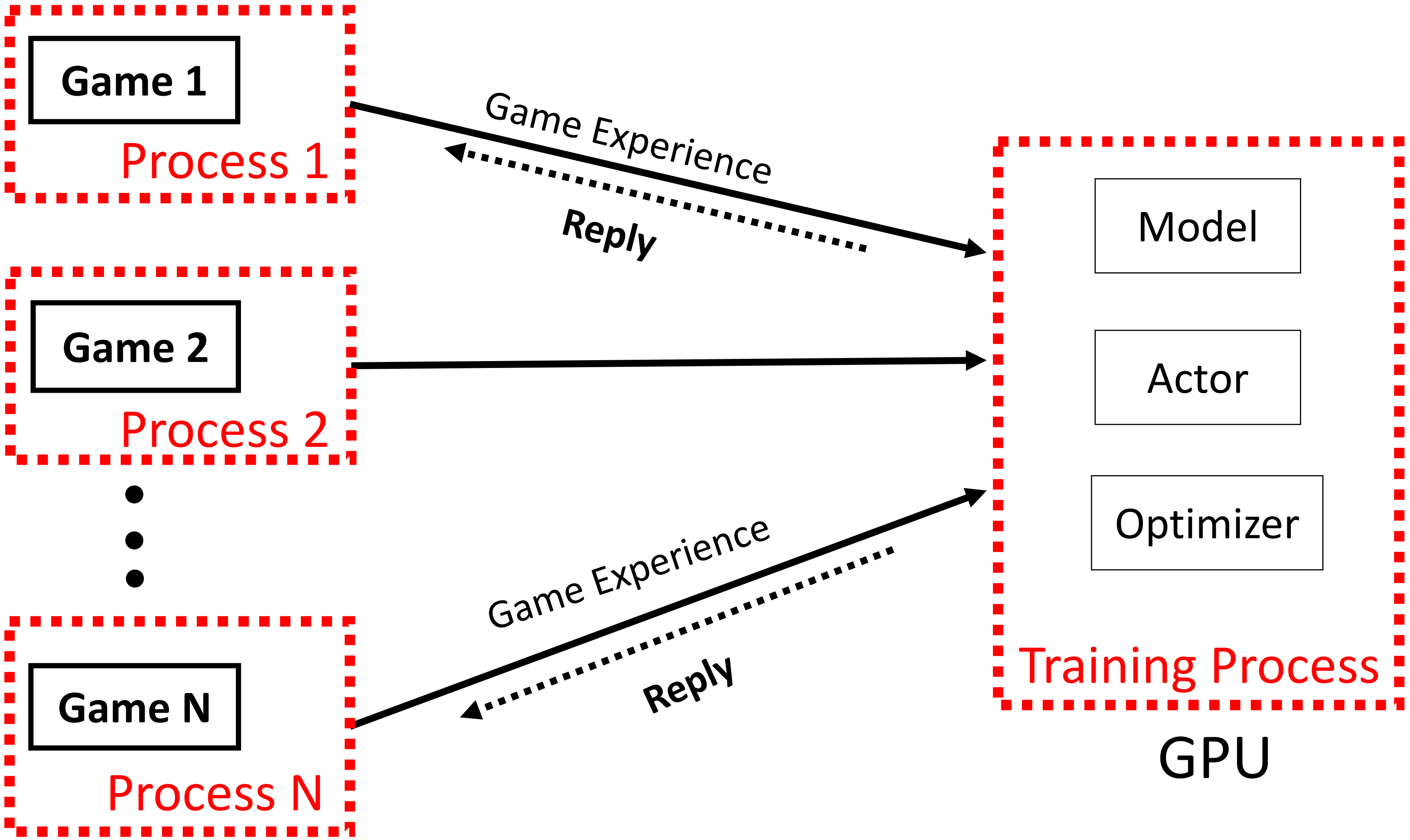
[Nair et al, Massively Parallel Methods for Deep Reinforcement Learning, ICML 2015]

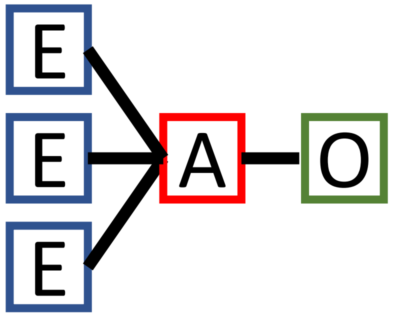
Asynchronized Advantageous Actor-Critic (A3C)



[Mnih et al, Asynchronous Methods for Deep Reinforcement Learning, ICML 2016]

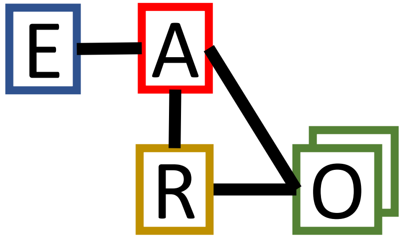
GA3C / BatchA2C





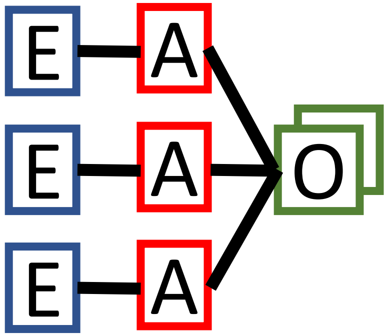
[Babaeizadeh et al, Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU, ICLR 2017]

ELF: A unified framework



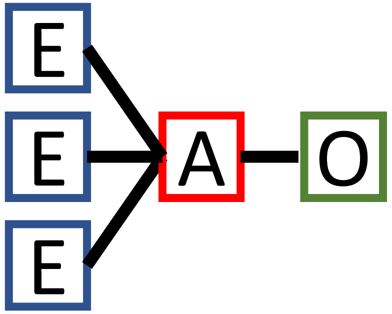
Off-policy training

Deep Q-learning



One-to-One

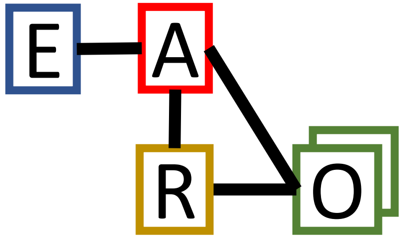
Vanilla A3C



Many-to-One

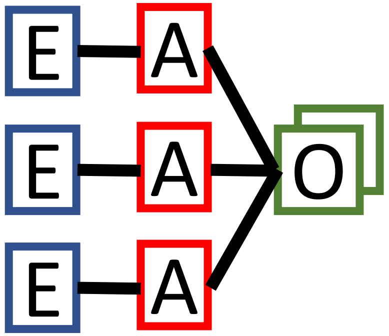
BatchA2C, GA3C

ELF: A unified framework



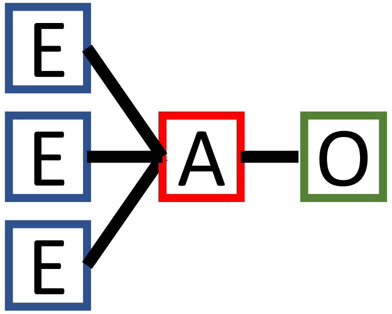
Off-policy training

Deep Q-learning



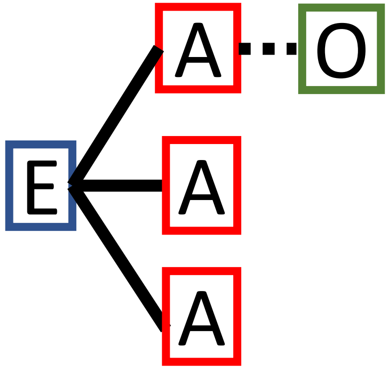
One-to-One

Vanilla A3C



Many-to-One

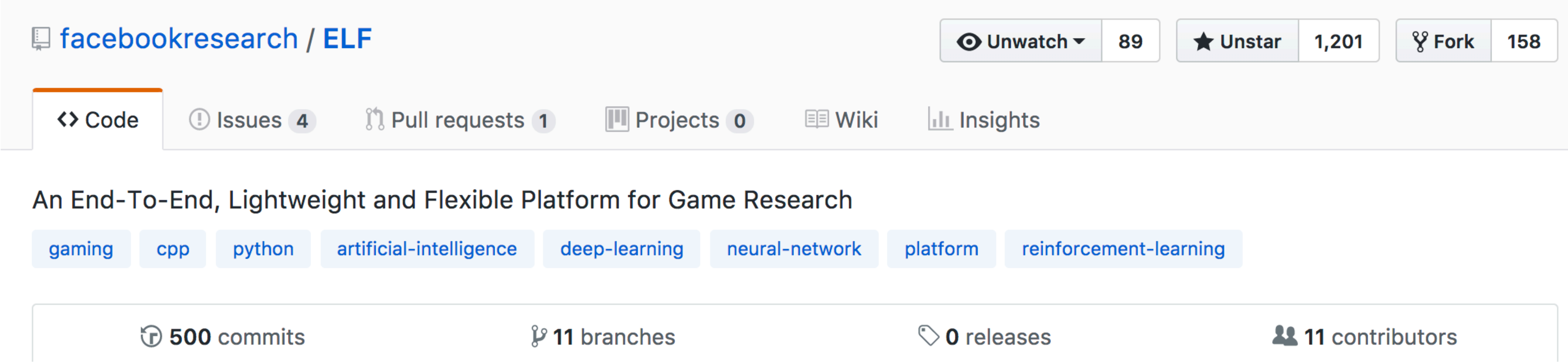
BatchA3C, GA3C



One-to-Many

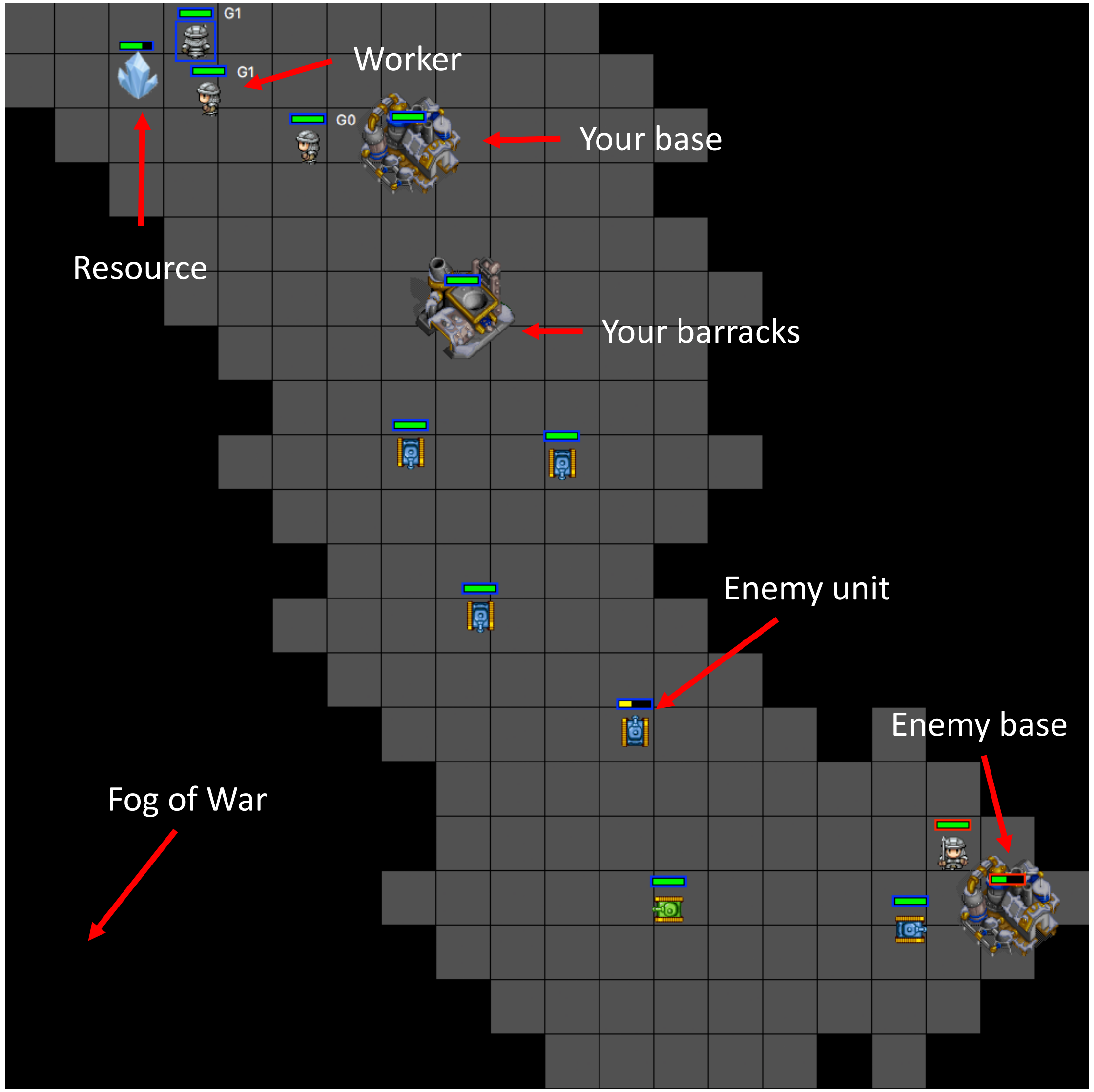
Self-Play,
Monte-Carlo Tree Search

Open Source



https://github.com/facebookresearch/ELF

MiniRTS: A miniature RTS engine



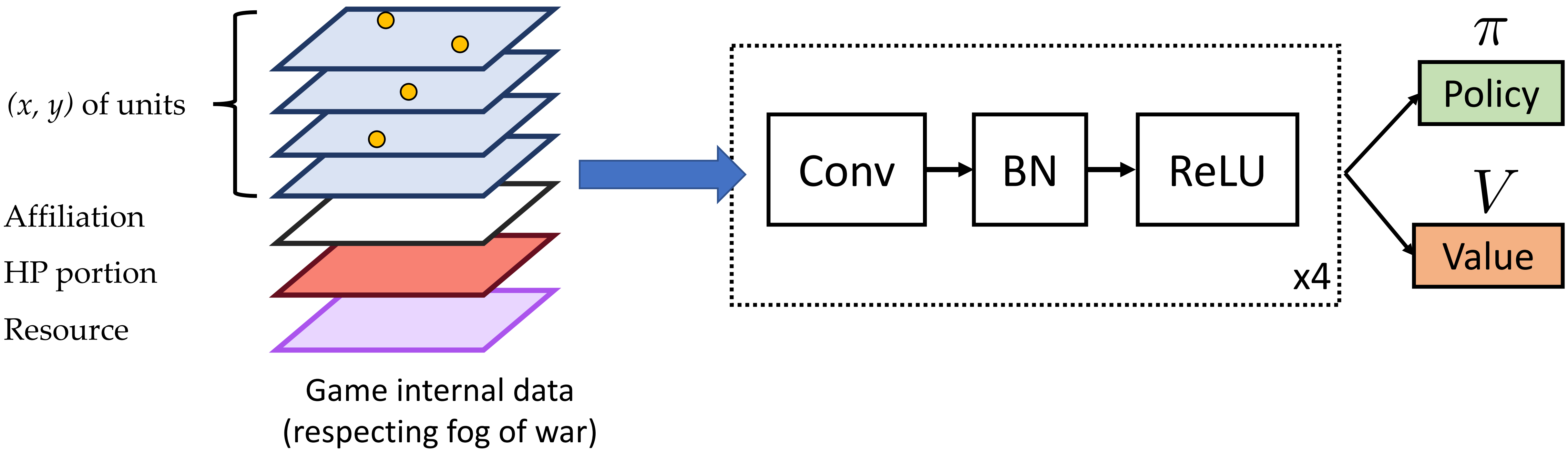
Platform	Frame per second
ALE	6,000
Open AI Universe	60
Malmo	120
DeepMind Lab	287*/866**
VizDoom	7,000
TorchCraft	2,000
MiniRTS	40,000

* Using CPU only ** Using CPUs and GPU

MiniRTS

Base		Build workers and collect resources.
Resource		Contains 1000 minerals.
Barracks		Build melee attacker and range attacker.
Worker		Build barracks and gather resource. Low speed in movement and low attack damage.
Melee Tank		High HP, medium movement speed, short attack range, high attack damage.
Range Tank		Low HP, high movement speed, long attack range and medium attack damage.

Training AI

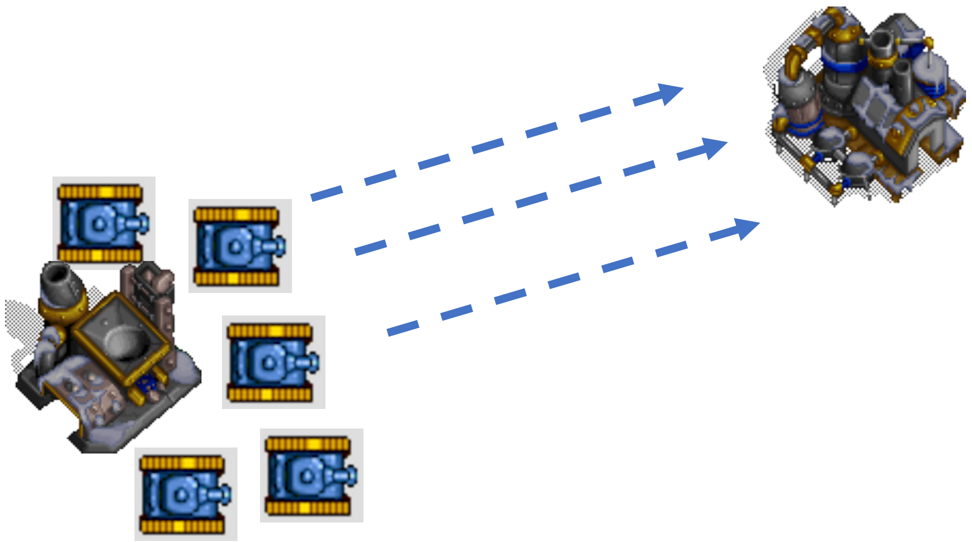


Using Internal Game data and Actor-Critic Models.
Reward is only available once the game is over.

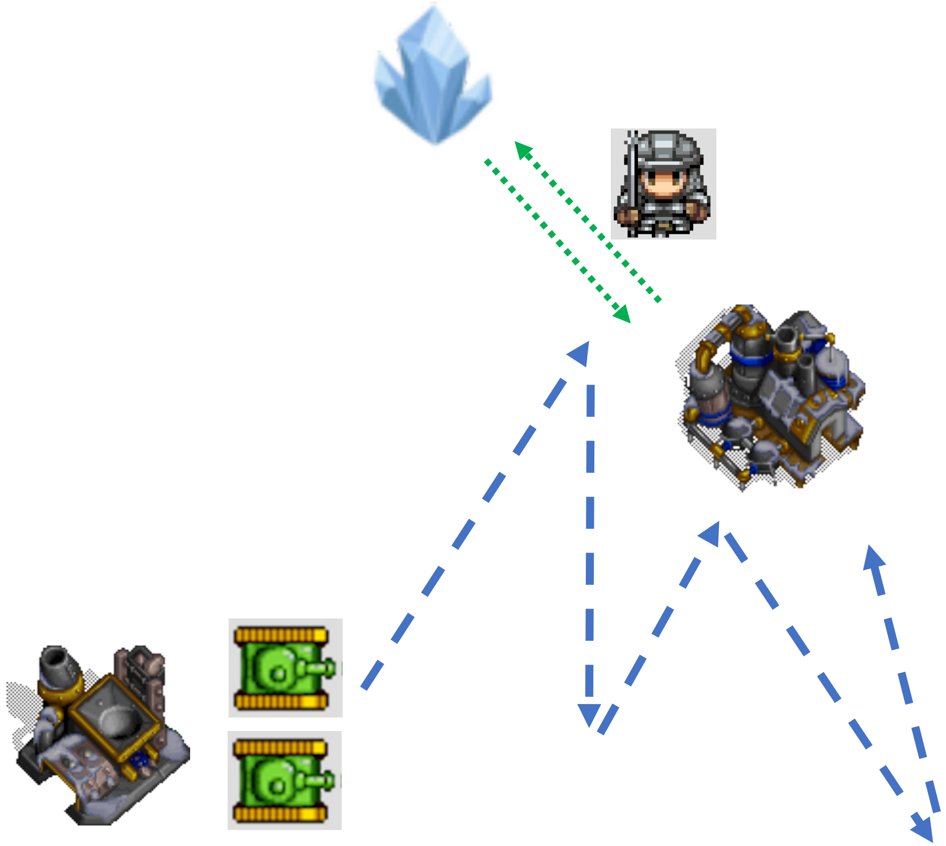
9 Discrete Strategic Actions

No.	Action name	Descriptions
1	IDLE	Do nothing
2	BUILD WORKER	If the base is idle, build a worker
3	BUILD BARRACK	Move a worker (gathering or idle) to an empty place and build a barrack.
4	BUILD MELEE ATTACKER	If we have an idle barrack, build an melee attacker.
5	BUILD RANGE ATTACKER	If we have an idle barrack, build a range attacker.
6	HIT AND RUN	If we have range attackers, move towards opponent base and attack. Take advantage of their long attack range and high movement speed to hit and run if enemy counter-attack.
7	ATTACK	All melee and range attackers attack the opponent's base.
8	ATTACK IN RANGE	All melee and range attackers attack enemies in sight.
9	ALL DEFEND	All troops attack enemy troops near the base and resource.

Rule-based AIs



AI_SIMPLE
Build 5 tanks and attack



AI_HIT_AND_RUN
Build 2 tanks and harass

<u>MiniRTS trains with a single GPU and 6 CPUs in half a day.</u>


Trained AI
B0
G1
AI_SIMPLE


Trained AI
G2
G0
AI_SIMPLE

Win rate against rule-based AI

Network Architecture

Conv → BN → ReLU

Win Rate (10K games)	SIMPLE (median)	SIMPLE (mean/std)	HIT_AND_RUN (median)	HIT_AND_RUN (mean/std)
ReLU	52.8	54.7(±4.2)	60.4	57.0(±6.8)
Leaky ReLU	59.8	61.0(±2.6)	60.2	60.3(±3.3)
ReLU + BN	61.0	64.4(±7.4)	55.6	57.5(±6.8)
Leaky ReLU + BN	**72.2**	**68.4(±4.3)**	**65.5**	**63.6(±7.9)**

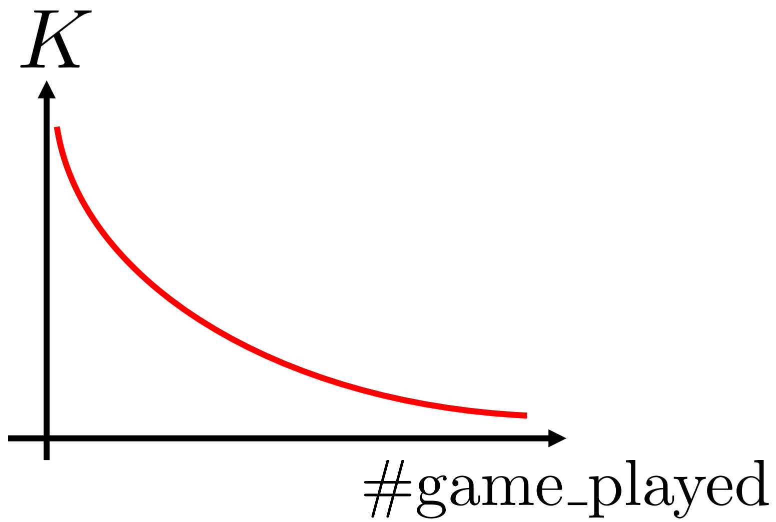
Curriculum Training

Win Rate	Without curriculum training	With curriculum training
AI_SIMPLE	66.0 (±2.4)	**68.4 (±4.3)**
AI_HIT_AND_RUN	54.4 (±15.9)	**63.6 (±7.9)**

First k decisions made by `AI_SIMPLE`
then made by trained AI

$$k \sim \mathrm{Uniform}[0, K]$$

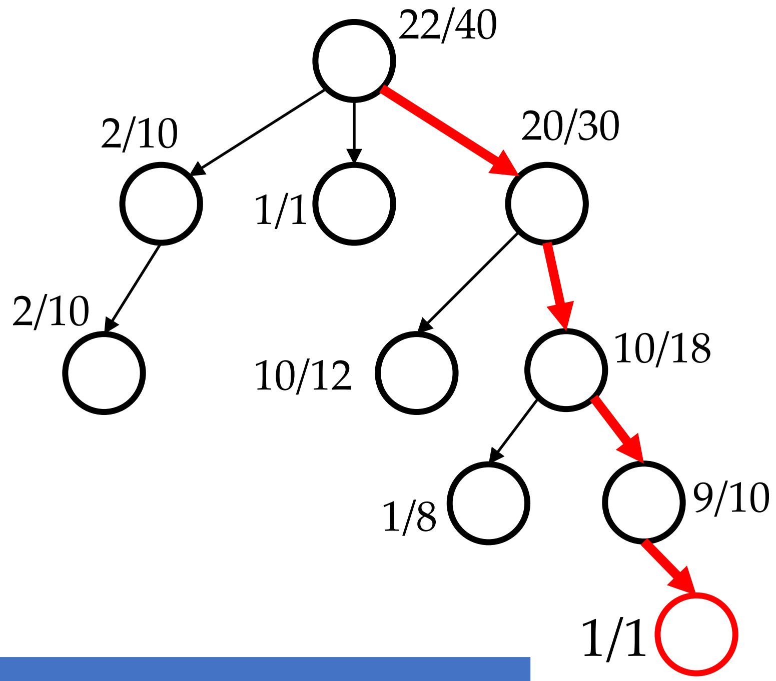
$$K \propto \beta^{-\#\mathrm{game\_played}}$$



Transfer Learning

Win Rate	AI_SIMPLE	AI_HIT_AND_RUN	Combined (50%SIMPLE+50% H&R)
SIMPLE	**68.4 (±4.3)**	26.6(±7.6)	47.5(±5.1)
HIT_AND_RUN	34.6(±13.1)	**63.6 (±7.9)**	49.1(±10.5)
Combined	51.8(±10.6)	54.7(±11.2)	**53.2(±8.5)**

Monte Carlo Tree Search

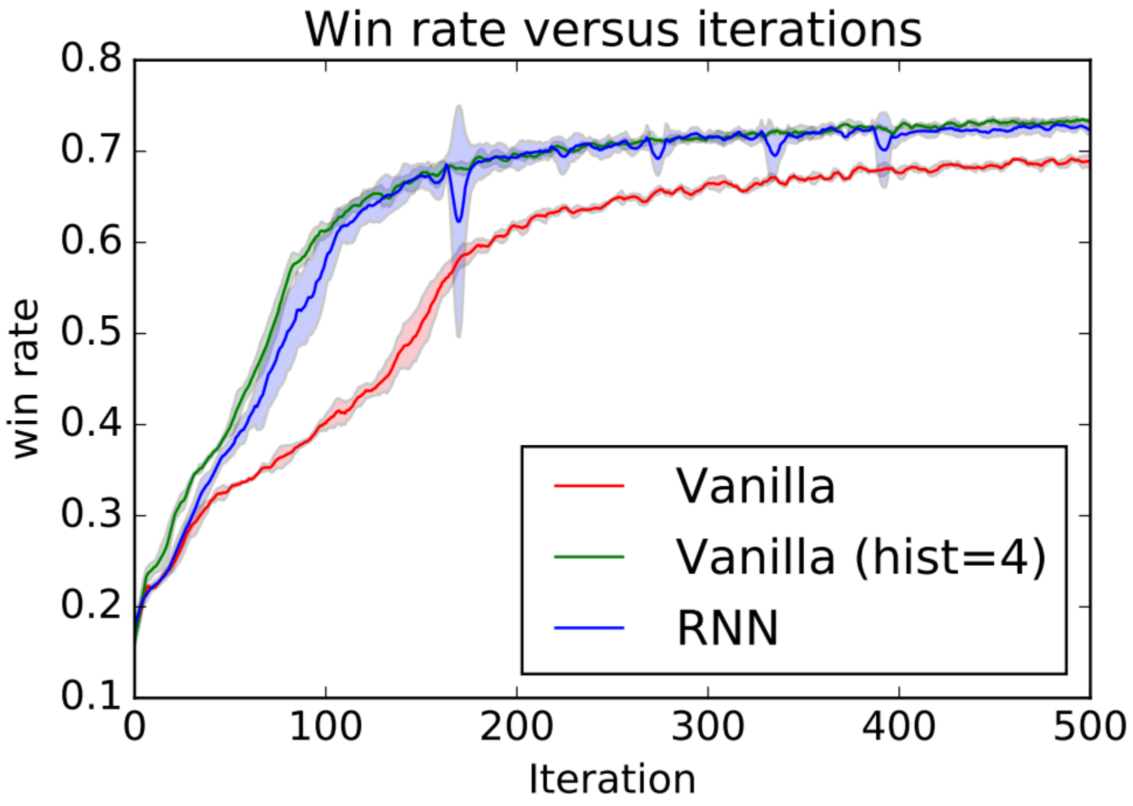


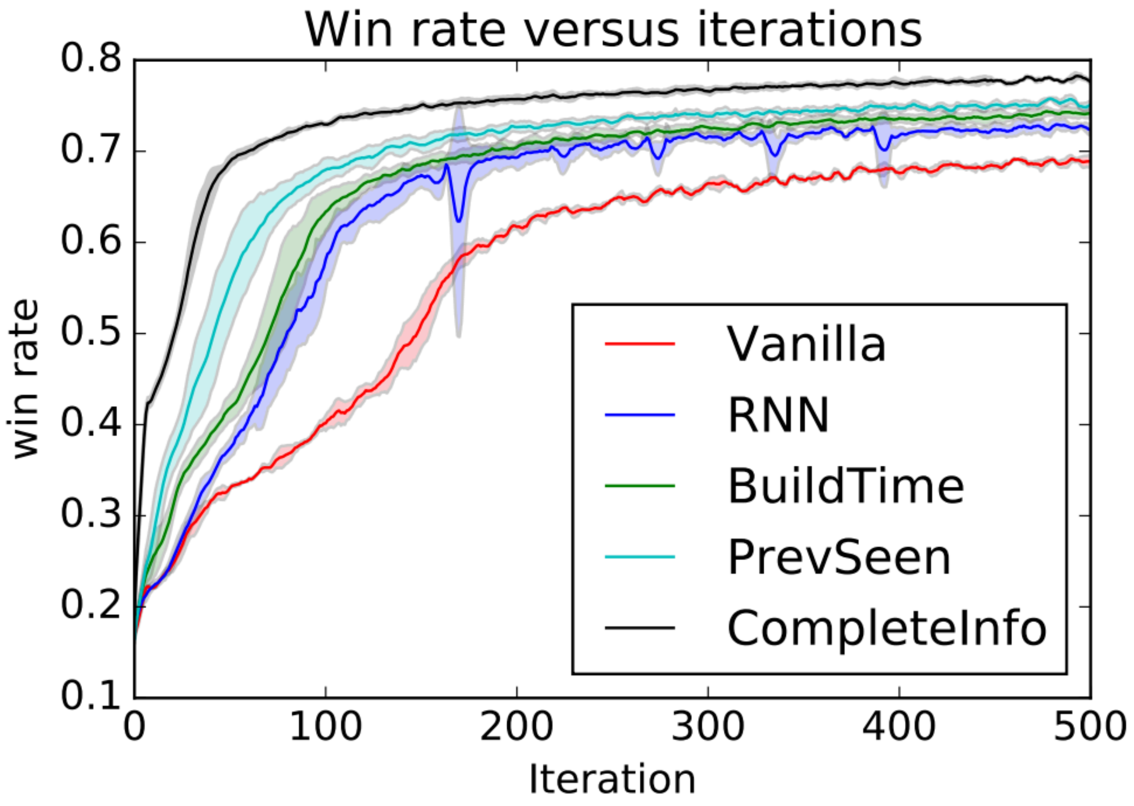
Win Rate	AI_SIMPLE	AI_HIT_AND_RUN
Random	24.2 (±3.9)	25.9 (±0.6)
MCTS*	73.2 (±0.6)	62.7 (±2.0)
Trained AI	**68.4(±4.3)**	**63.6(±7.9)**

* repeat on 1000 games, each using 800 rollouts.

MCTS uses complete information and perfect dynamics

Recent Update





Method	*Vanilla*	*Vanilla*(hist=4)	RNN	*BuildHistory*	*PrevSeen*	Complete Info
Win rate	72.9±1.8	79.8±0.7	79.7±1.3	80.8±1.7	81.4±0.8	81.7±0.7

Ongoing Work

Engineering

- Richer game scenarios for MiniRTS.
  - LUA scripting support
  - Multiple bases (Expand? Rush? Defending?)
  - More complicated units.
- Realistic action space
  - One command per unit

Research

- Model-based Reinforcement Learning
- Hierarchical RL
- Self-Play (Trained AI versus Trained AI)

LUA Interface for MiniRTS

- Easy to change game dynamics
  - Don’t need to touch C++.
- Comparable speed to C++
  - 1.5x slower than compiled code.

```lua
g_funcs = { }
function g_funcs.attack(env, cmd)
    local target = env:unit(cmd.target)
    local u = env:self()

    if target:isdead() or not u:can_see(target) then
        -- c_print("Task finished!")
        return global.CMD_COMPLETE
    end
    local att_r = u:att_r()
    local in_range = env:dist_sqr(target:p()) <= att_r * att_r
    if u:cd_expired(global.CD_ATTACK) and in_range then
        -- print("Attacking .. ")
        -- Then we need to attack.
        if att_r <= 1.0 then
            env:send_cmd_melee_attack(cmd.target, u:att())
        else
            env:send_cmd_emit_bullet(cmd.target, u:att())
        end
        env:cd_start(global.CD_ATTACK)
    else
        if not in_range then
            -- print("Moving towards target .. ")
            env:move_towards(target)
        end
   end
   -- print("Done with Attacking .. ")
end
```

RLPytorch

- A RL platform in PyTorch
- A3C in 30 lines.

```python
# A3C
def update(self, batch):
    ''' Actor critic model '''
    R = deepcopy(batch["V"][T - 1])
    batchsize = R.size(0)
    R.resize_(batchsize, 1)

    for t in range(T - 2, -1, -1):
        # Forward pass
        curr = self.model_interface.forward("model", batch.hist(t))

        # Compute the reward.
        R = R * self.args.discount + batch["r"][t]
        # If we see any terminal signal, do not backprop
        for i, terminal in enumerate(batch["terminal"][t]):
            if terminal: R[t][i] = curr["V"].data[i]

        # We need to set it beforehand.
        self.policy_gradient_weights = R - curr["V"].data

        # Compute policy gradient error:
        errs = self._compute_policy_entropy_err(curr["pi"], batch["a"][t])
        # Compute critic error
        value_err = self.value_loss(curr["V"], Variable(R))

        overall_err = value_err + errs["policy_err"]
        overall_err += errs["entropy_err"] * self.args.entropy_ratio
        overall_err.backward()
```

House3D: A rich and realistic 3D environment

Yi Wu

Georgia Gkioxari

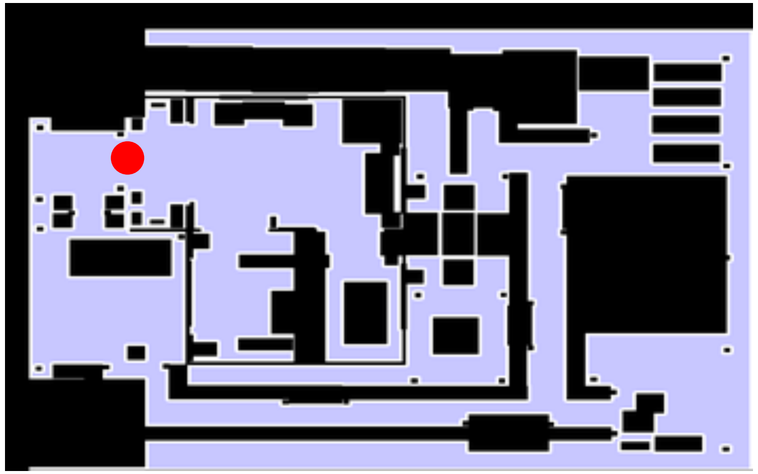
Yuxin Wu

[Yi Wu et al, Building Generalizable Agents with a Realistic and Rich 3D Environment, ICLR 2018 submission]

SUNCG Dataset

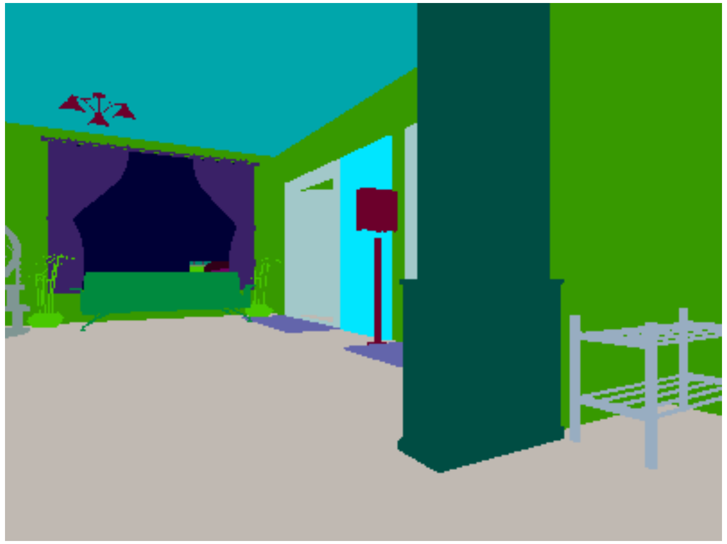
SUNCG dataset, 45K scenes, all objects are fully labeled.

Multi-modality



Top-down map

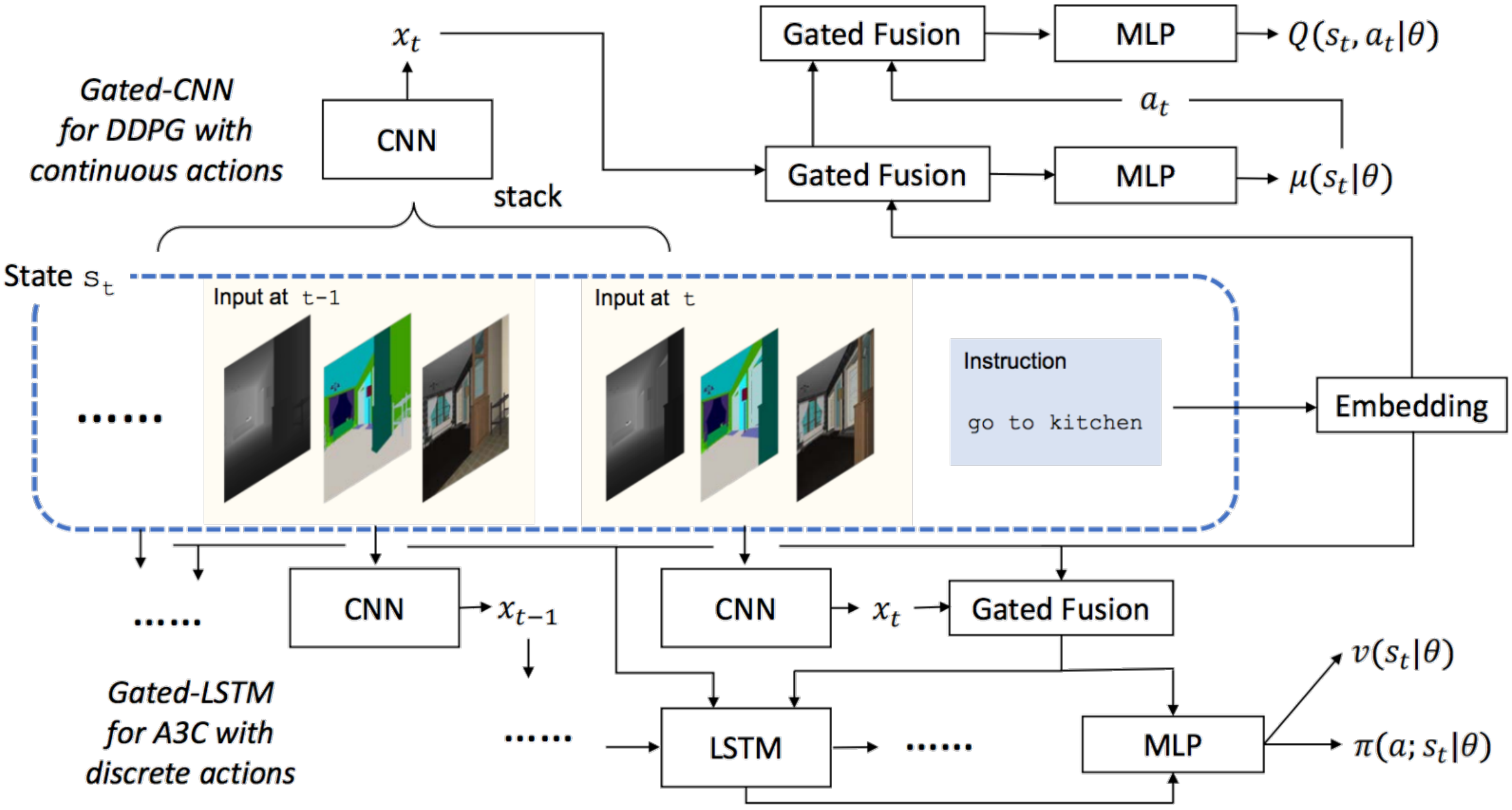
Depth



Segmentation mask

RGB image

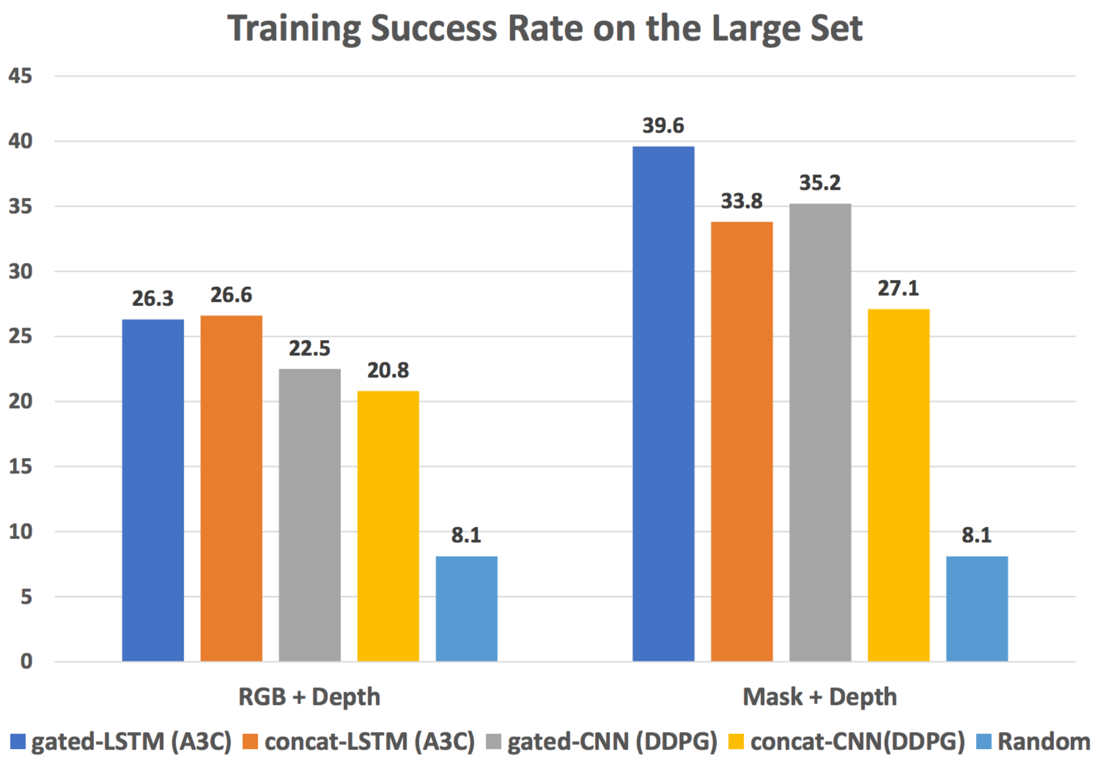
Architecture



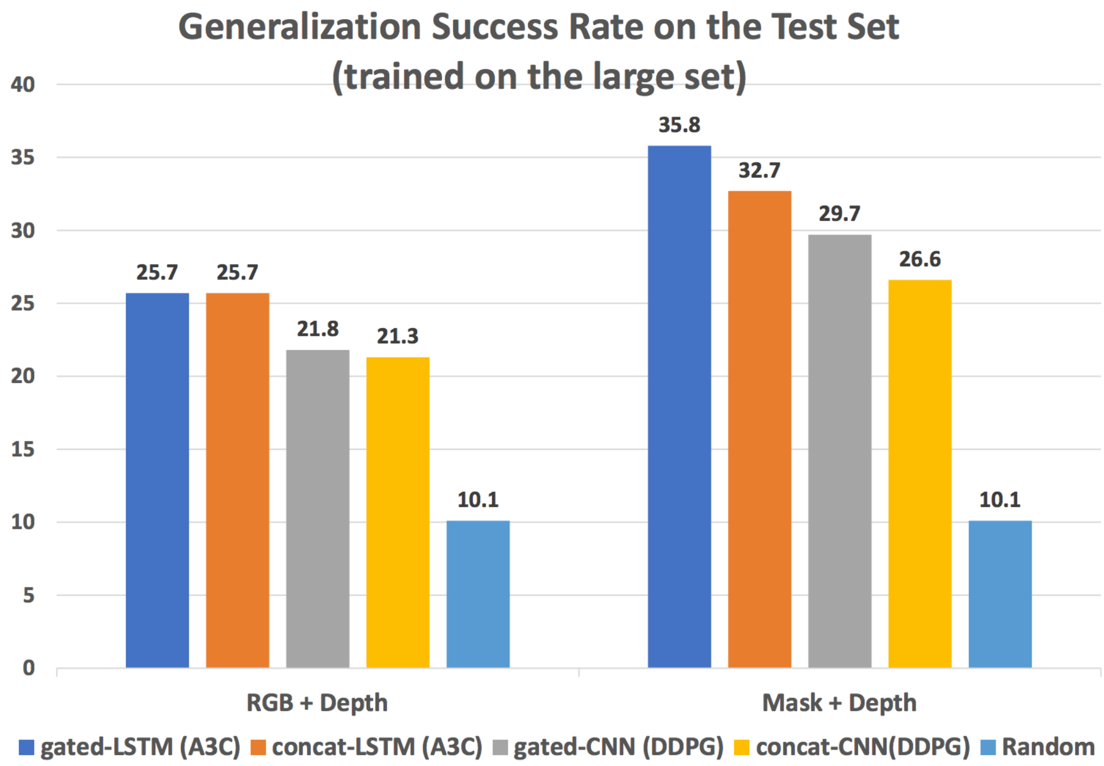
Comparison

Environment	3D	Realistic	Large-scale	Fast-speed	Customizable
Atari (Bellemare et al., 2013)				●	
OpenAI Universe (Shi et al., 2017)		●	●		●
Malmo (Johnson et al., 2016)	●		●	●	●
DeepMind Lab (Beattie et al., 2016)	●			●	●
VizDoom (Kempka et al., 2016)	●			●	●
AI2-THOR (Zhu et al., 2017)	●	●		●	
House3D	●	●	●	●	●

Successful Rate

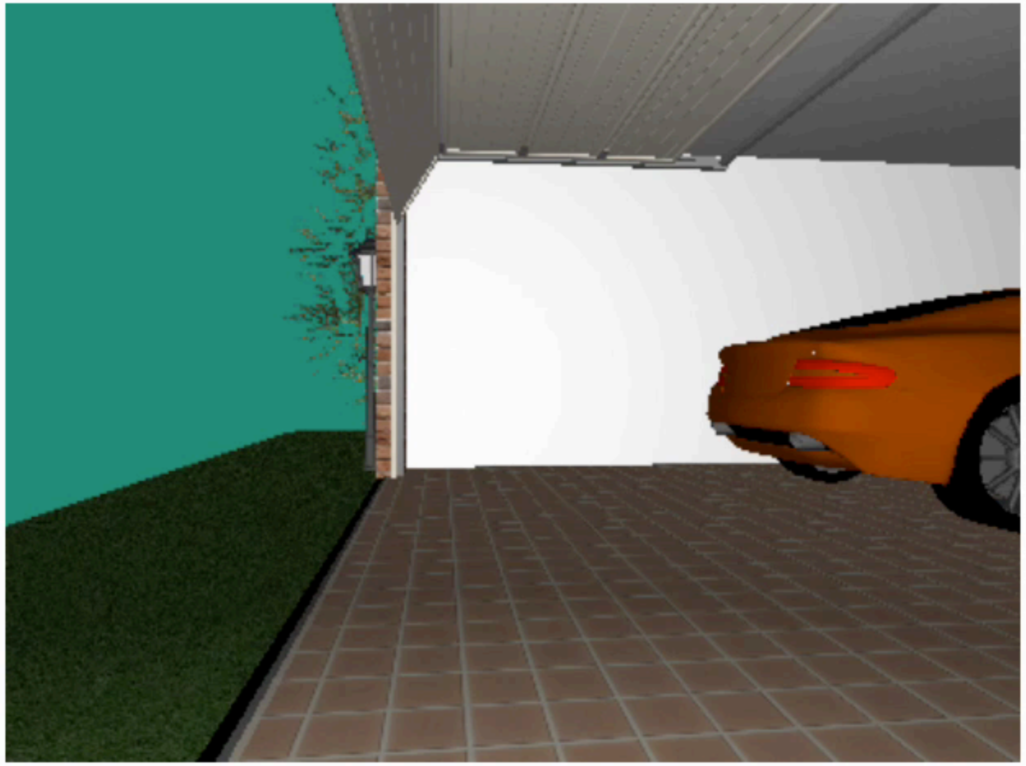


(a) Training performances



(b) Generalization performances on the test set

Videos

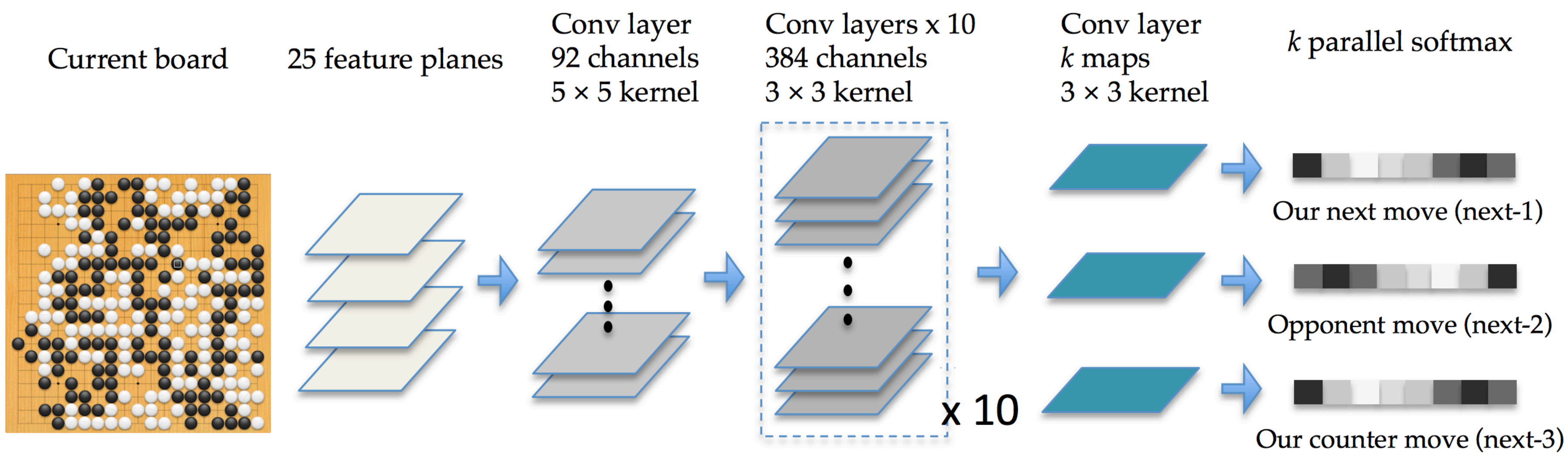


DarkForest: Go engine

Yuandong Tian and Yan Zhu, ICLR 2016

- DCNN as a tree policy
  - Predict next k moves (rather than next move)
  - Trained on 170k KGS dataset/80k GoGoD, ***57.1%*** accuracy.
  - KGS 3D without search (0.1s per move)
  - Release 3 month before AlphaGo, < 1% GPUs (from Aja Huang)

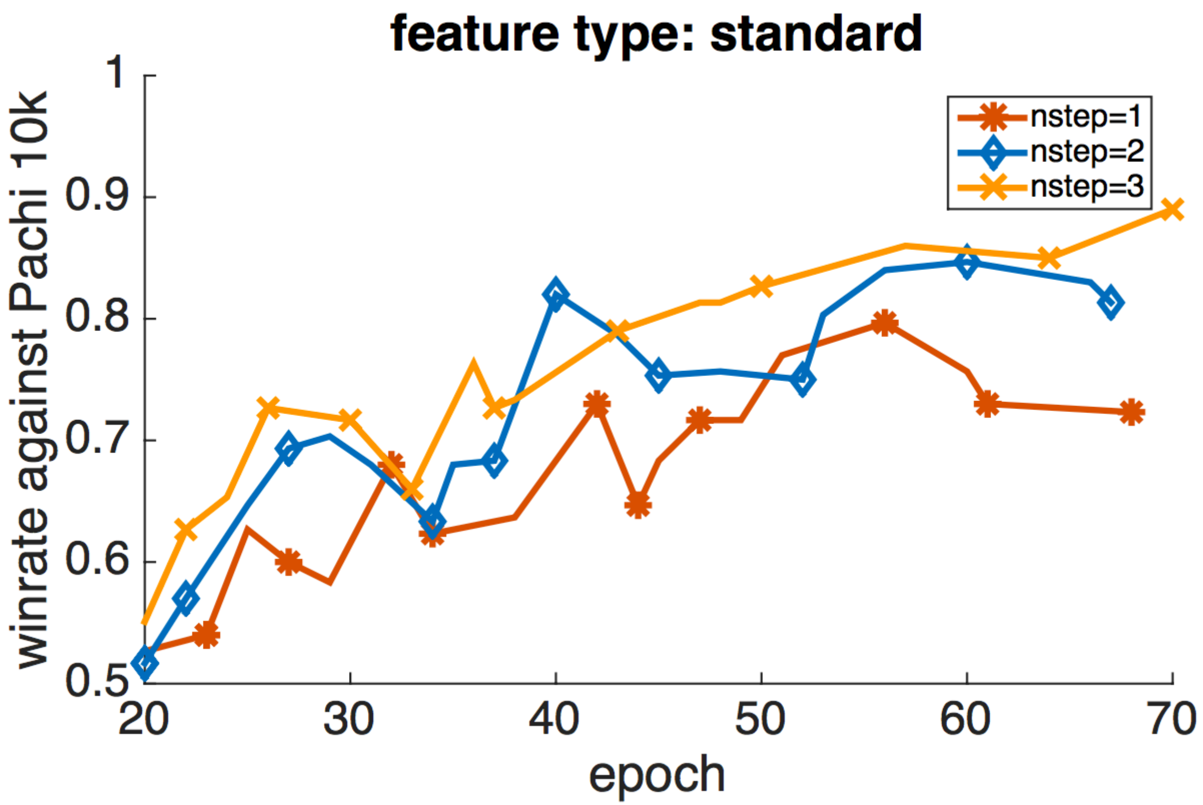
Yan Zhu



Our computer Go player: DarkForest

Name
Our/enemy liberties
Ko location
Our/enemy stones/empty place
Our/enemy stone history
Opponent rank

Feature used for DCNN



Pure DCNN

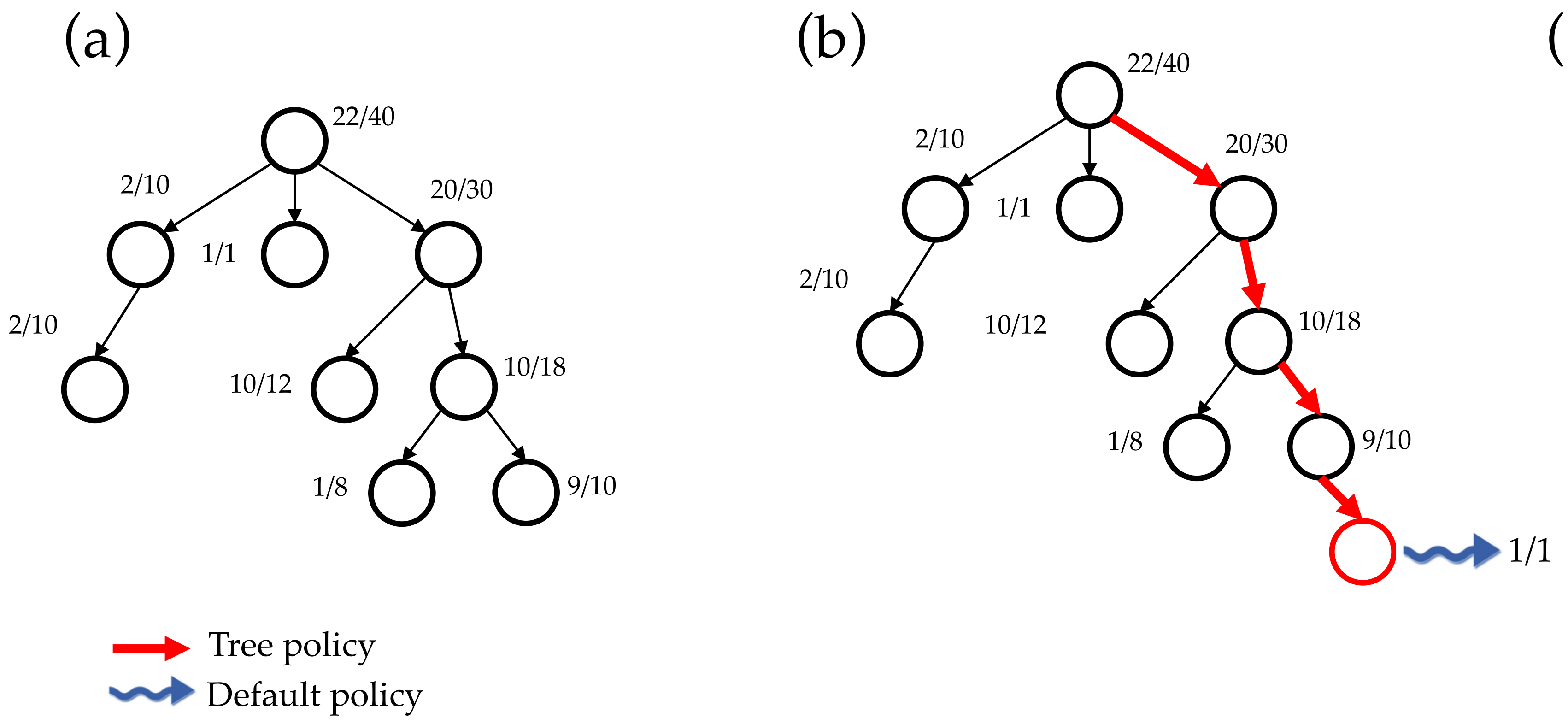
darkforest: Only use top-1 prediction, trained on KGS
darkfores1: Use top-3 prediction, trained on GoGoD
darkfores2: *darkfores1* with fine-tuning.

	GnuGo (level 10)	Pachi 10k	Pachi 100k	Fuego 10k	Fuego 100k
Clark & Storkey (2015)	91.0	-	-	14.0	
Maddison et al. (2015)	97.2	47.4	11.0	23.3	12.5
darkforest	98.0 ± 1.0	71.5 ± 2.1	27.3 ± 3.0	84.5 ± 1.5	56.7 ± 2.5
darkfores1	99.7 ± 0.3	88.7 ± 2.1	59.0 ± 3.3	93.2 ± 1.5	78.0 ± 1.7
darkfores2	$\mathbf{100 \pm 0.0}$	$\mathbf{94.3 \pm 1.7}$	$\mathbf{72.6 \pm 1.9}$	$\mathbf{98.5 \pm 0.1}$	$\mathbf{89.7 \pm 2.1}$

Win rate between DCNN and open source engines.

Monte Carlo Tree Search

Aggregate win rates, and search towards the good nodes.



DCNN + MCTS

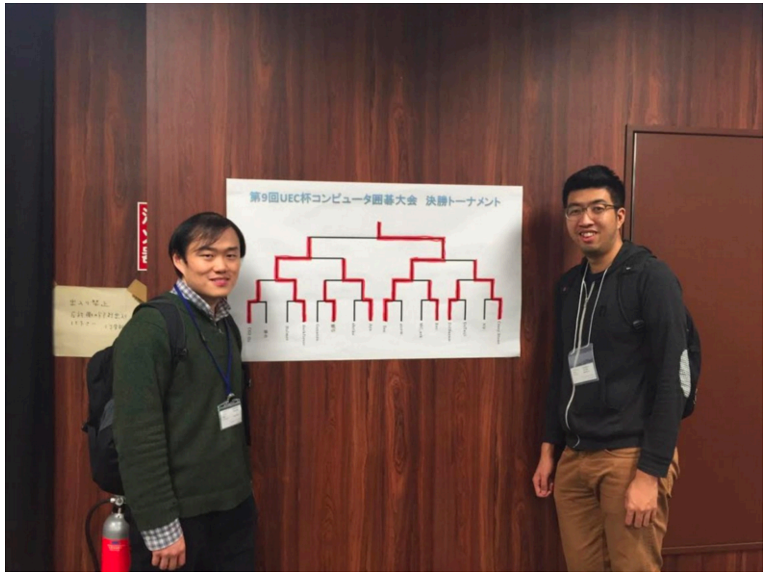
darkfmcts3: Top-3/5, 75k rollouts, ~12sec/move, KGS 5d

	darkforest+MCTS	darkfores1+MCTS	darkfores2+MCTS
Vs pure DCNN (1000rl/top-20)	84.8%	74.0%	62.8%
Vs pure DCNN (1000rl/top-5)	89.6%	76.4%	68.4%
Vs pure DCNN (1000rl/top-3)	91.6%	89.6%	~~79.2%~~ 94.2%
Vs pure DCNN (5000rl/top-5)	96.8%	94.3%	82.3%
Vs Pachi 10k (pure DCNN baseline)	71.5%	88.7%	94.3%
Vs Pachi 10k (1000rl/top-20)	91.2% (+19.7%)	92.0% (+3.3%)	95.2% (+0.9%)
Vs Pachi 10k (1000rl/top-5)	88.4% (+16.9%)	94.4% (+5.7%)	97.6% (+3.3%)
Vs Pachi 10k (1000rl/top-3)	95.2% (+23.7%)	98.4% (+9.7%)	99.2% (+4.9%)
Vs Pachi 10k (5000/top-5)	98.4%	99.6%	100.0%

Win rate between DCNN + MCTS and open source engines.

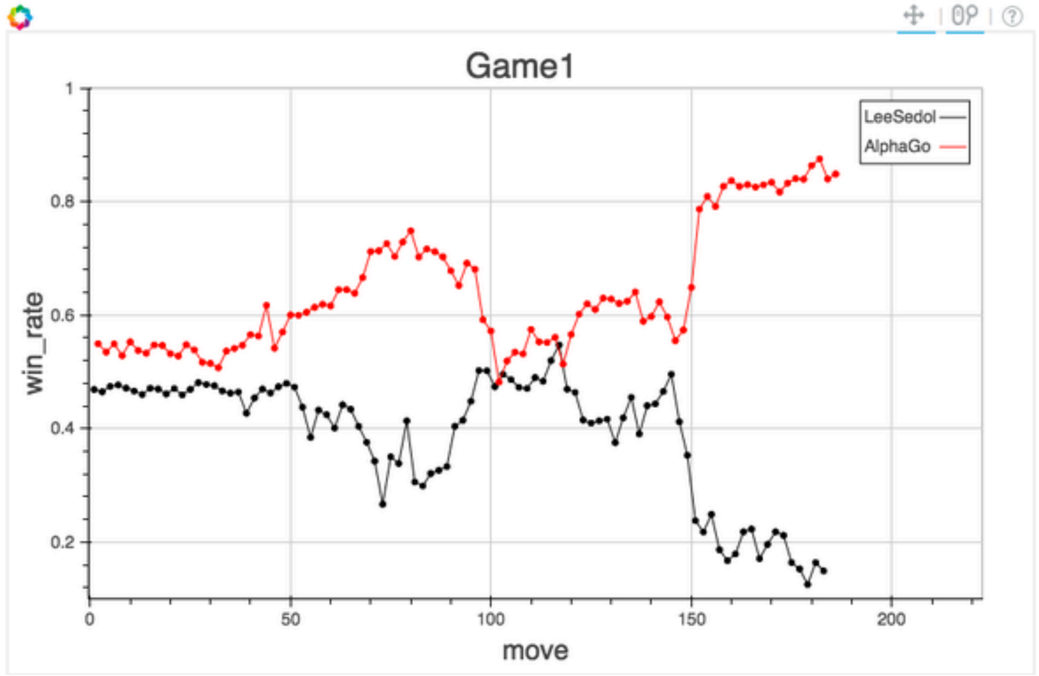
DarkForest

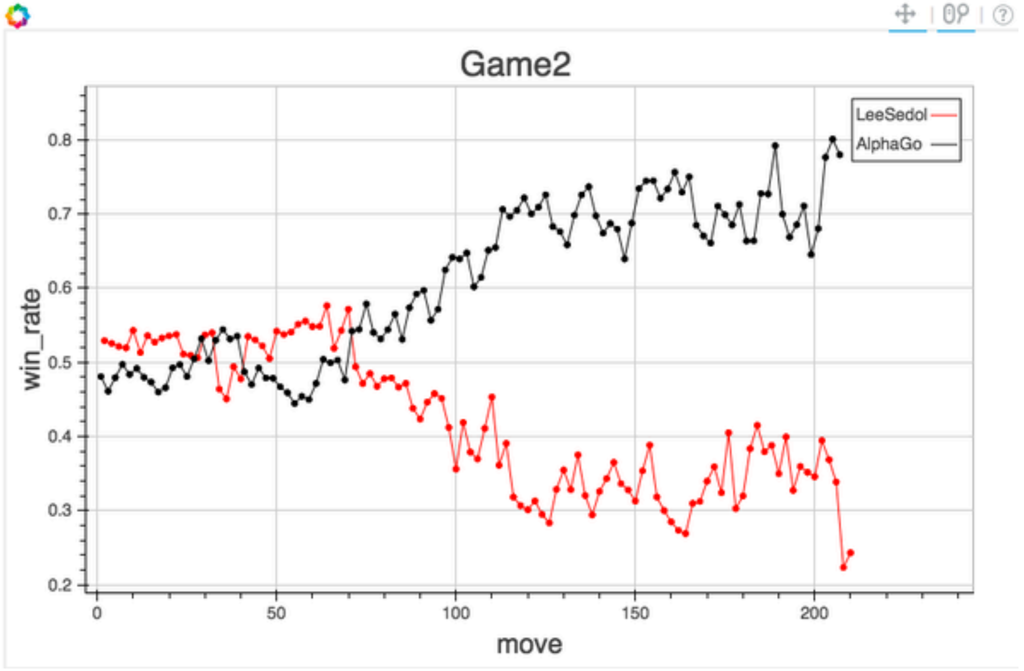
- DCNN+MCTS
  - Use top3/5 moves from DCNN, 75k rollouts.
  - Stable KGS 5d. Open source. https://github.com/facebookresearch/darkforestGo
  - 3rd place on KGS January Tournaments
  - 2nd place in 9th UEC Computer Go Competition (Not this time ☺)



DarkForest versus Koichi Kobayashi (9p)

Win Rate analysis (using DarkForest) (AlphaGo versus Lee Sedol)





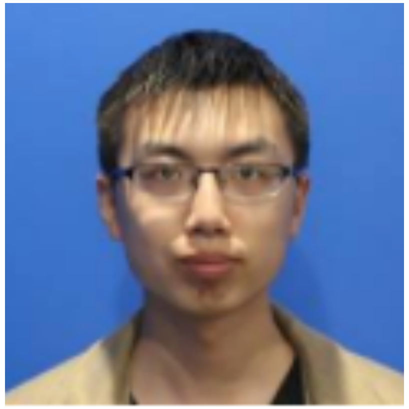
New version of DarkForest on ELF platform

https://github.com/facebookresearch/ELF/tree/master/go

First Person Shooter (FPS) Game

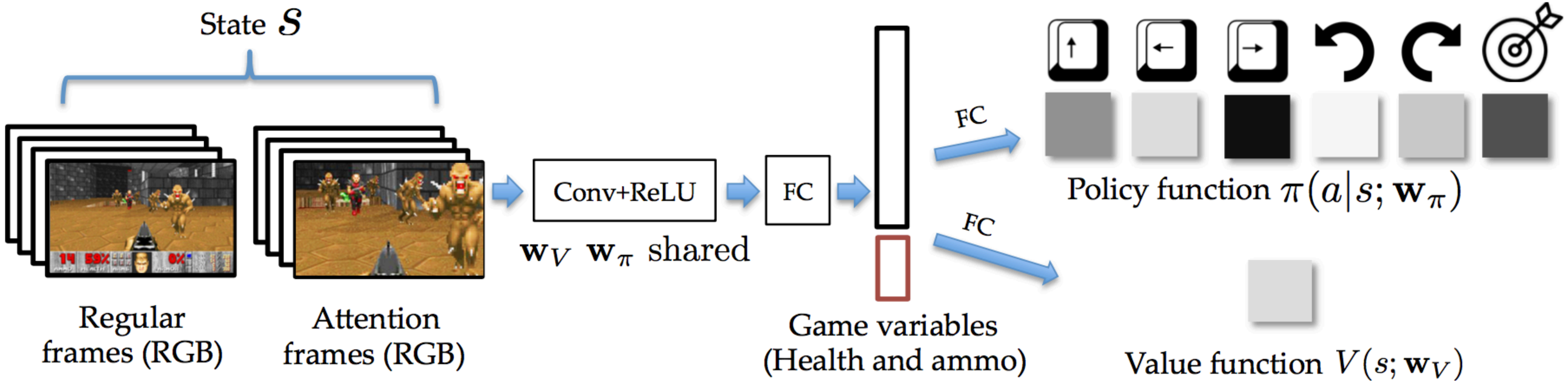
Yuxin Wu, Yuandong Tian, ICLR 2017

Play the game from the raw image!



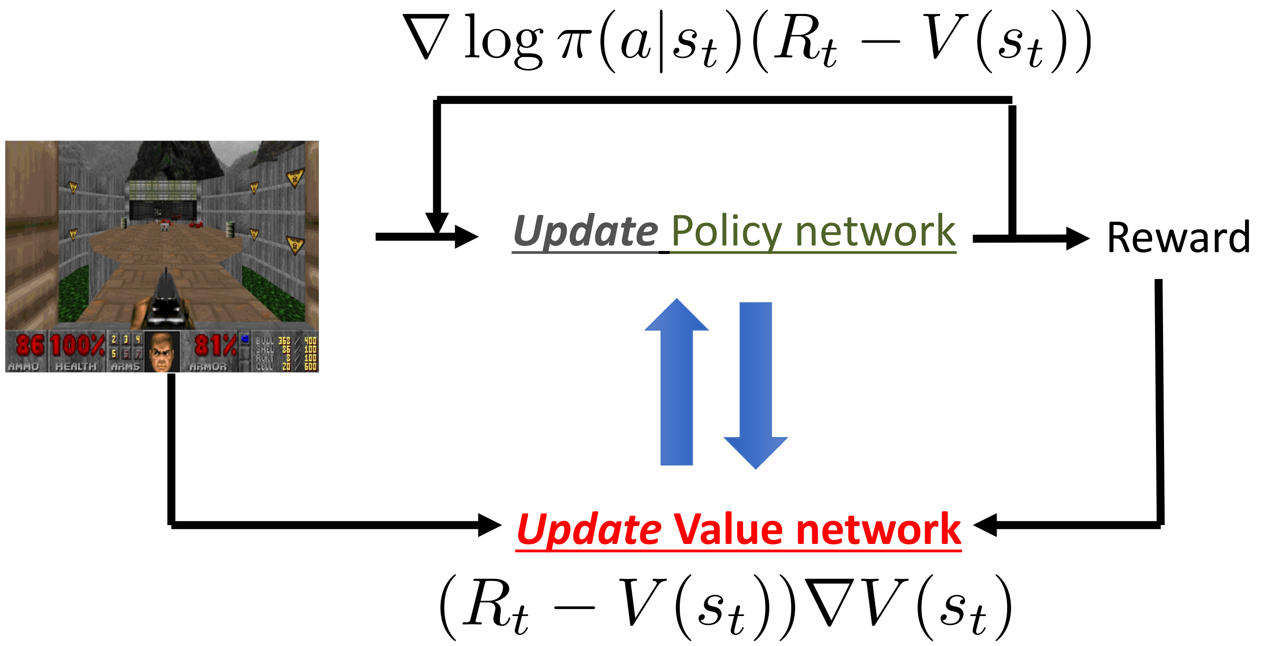
Yuxin Wu

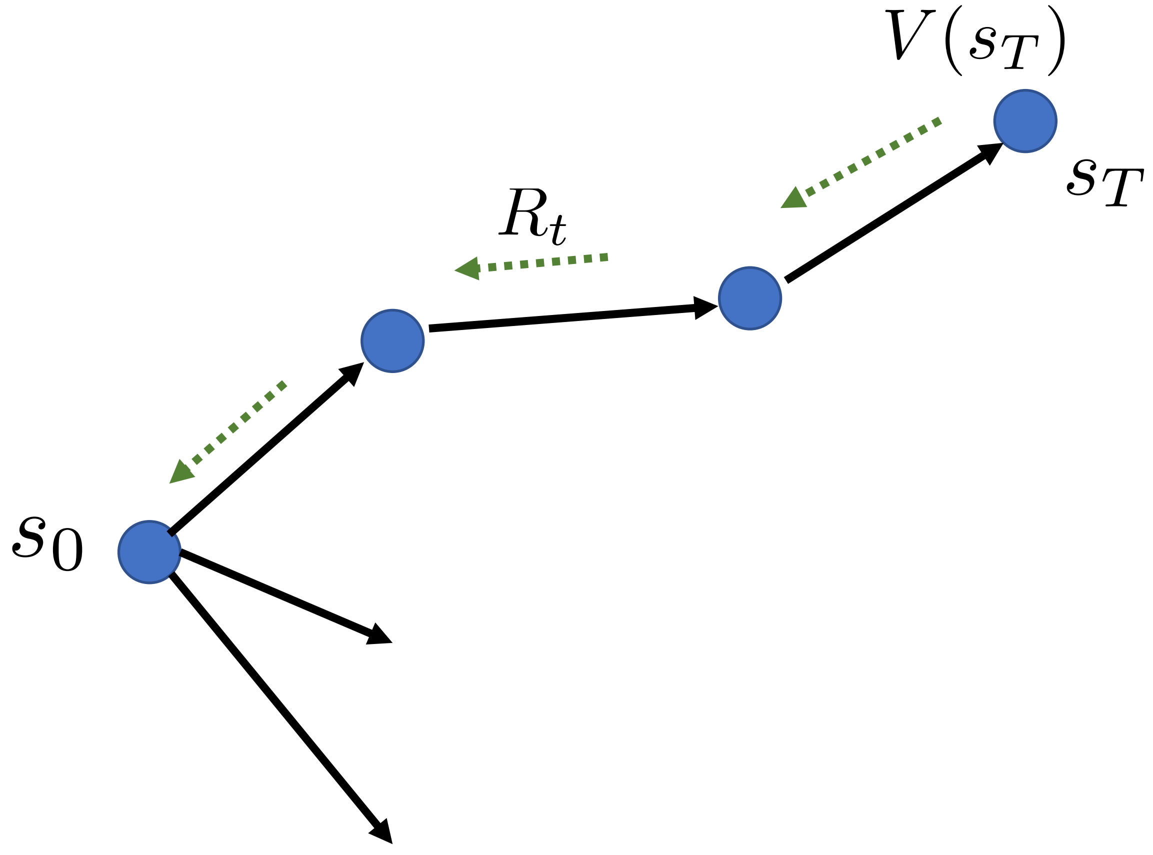
Network Structure



Simple Frame Stacking is very useful (rather than Using LSTM)

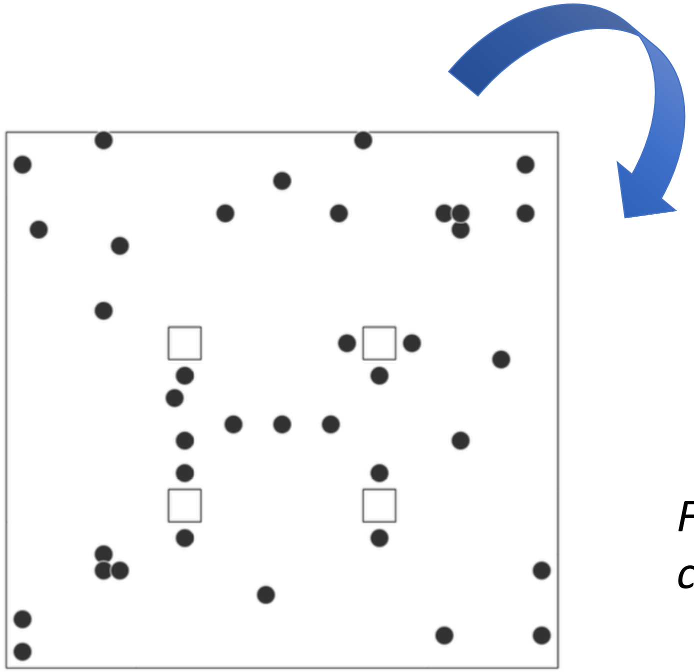
Actor-Critic Models





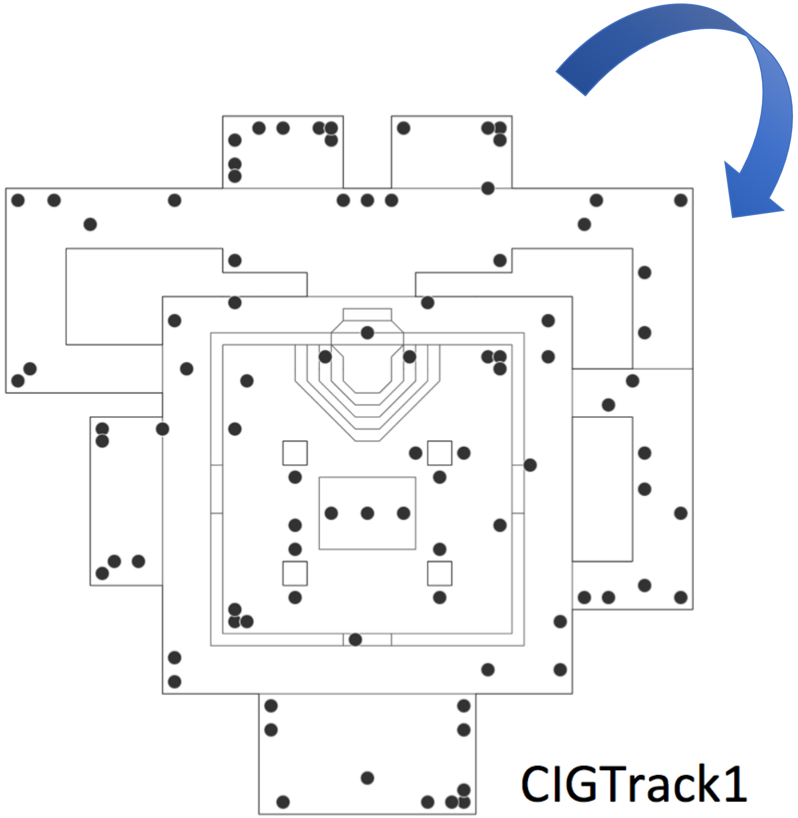
Encourage actions leading to states with high-than-expected value.
Encourage value function to converge to the true cumulative rewards.
Keep the diversity of actions

Curriculum Training



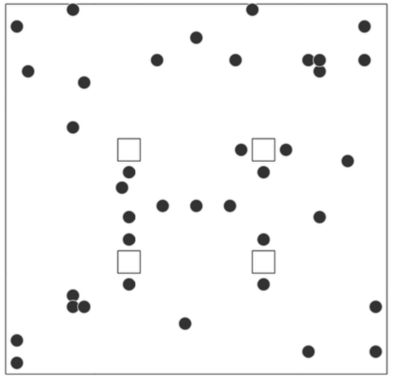
FlatMap

From simple to complicated



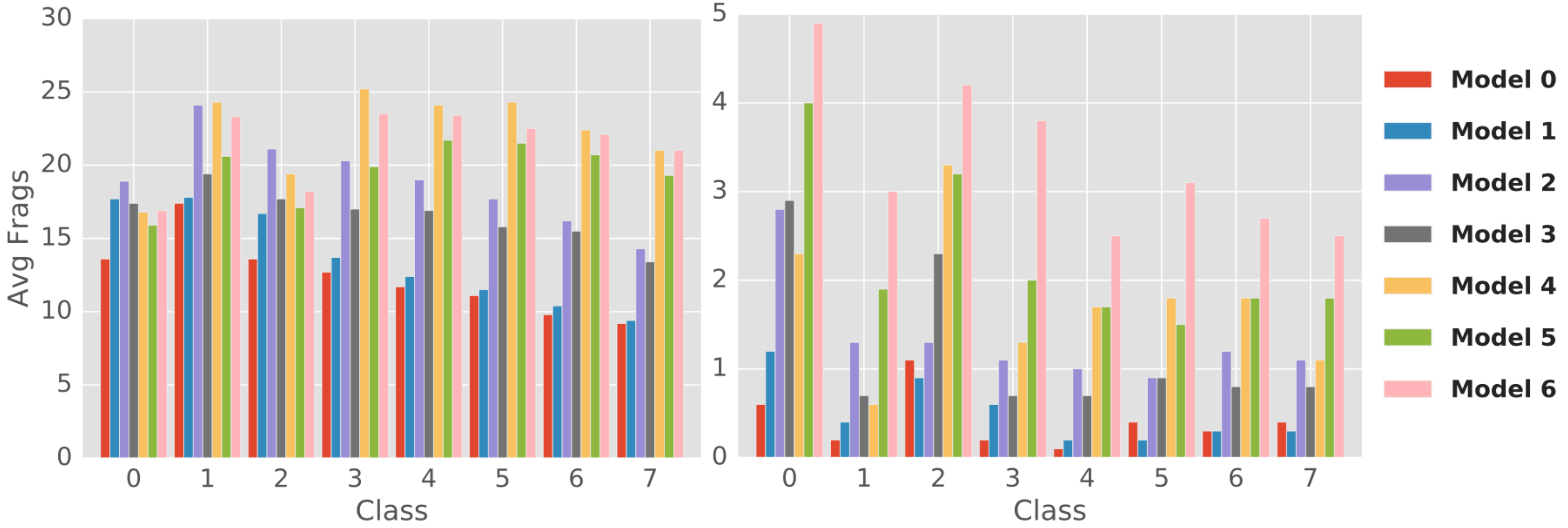
CIGTrack1

Curriculum Training



FlatMap

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
Speed	0.2	0.2	0.4	0.4	0.6	0.8	0.8	1.0
Health	40	40	40	60	60	60	80	100



VizDoom AI Competition 2016 (Track1)

We won the first place!

Rank	Bot	1	2	3	4	5	6	7	8	9	10	11	Total frags
1	F1	**56**	**62**	n/a	**54**	**47**	43	**47**	**55**	**50**	**48**	**50**	**559**
2	Arnold	36	34	**42**	36	36	**45**	36	39	n/a	33	36	413
3	CLYDE	37	n/a	38	32	37	30	46	42	33	24	44	393

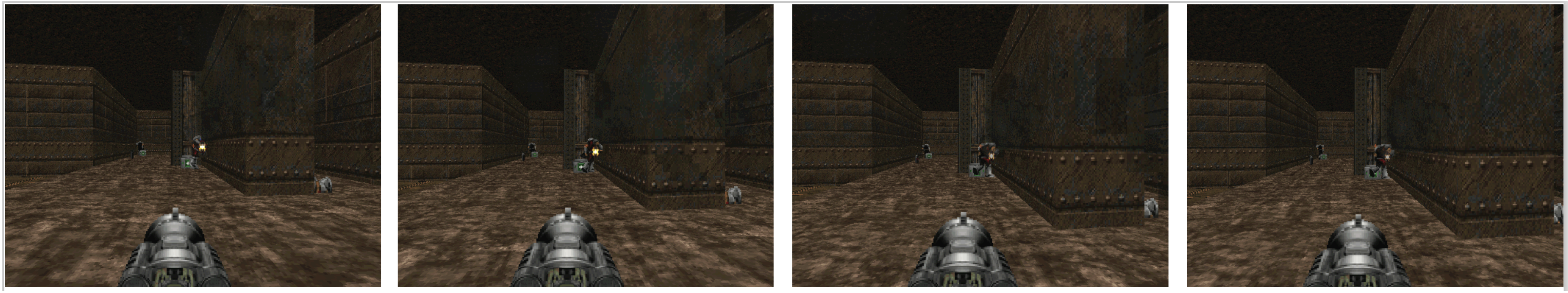
Videos:

https://www.youtube.com/watch?v=94EPSjQH38Y
https://www.youtube.com/watch?v=Qv4esGWOg7w&t=394s

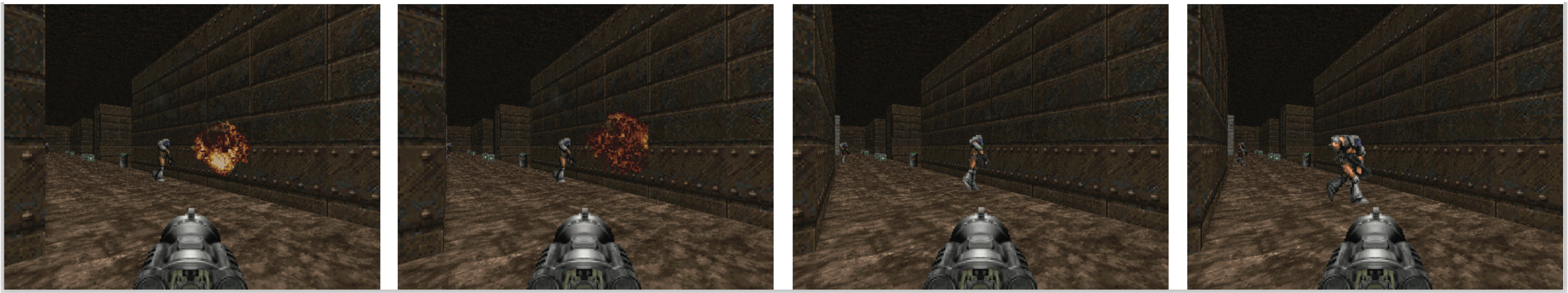

14
AMMO
93%
HEALTH
1
FRAGS
0%
ARMOR
0 / 200
0 / 50
14 / 50
0 / 300

Visualization of Value functions

Best 4 frames (agent is about to shoot the enemy)



Worst 4 frames (agent missed the shoot and is out of ammo)



Thanks!