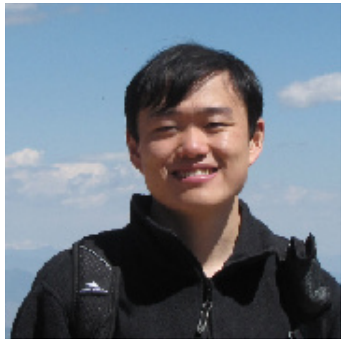
Reproducing AlphaZero with ELF: What we learned

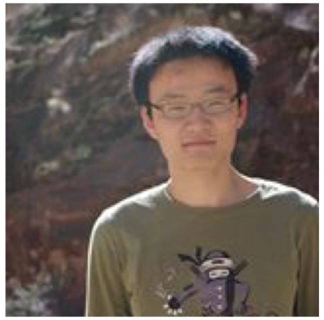
Yuandong Tian

Facebook AI Research



Yuandong Tian

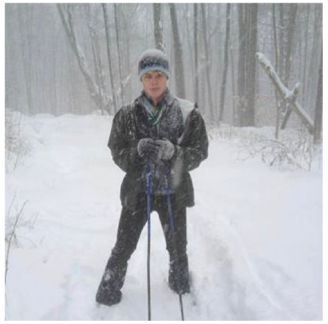
Jerry Ma*



Qucheng Gong*

Shubho Sengupta*

Zhuoyuan Chen



James Pinkerton

Larry Zitnick

AlphaGo Series

AlphaGo Lee
(Mar. 2016)

AlphaGo Master
(May. 2017)

AlphaGo Zero
(Oct. 2017)

AlphaGo Series

AlphaGo Lee
(Mar. 2016)

AlphaGo Master
(May. 2017)

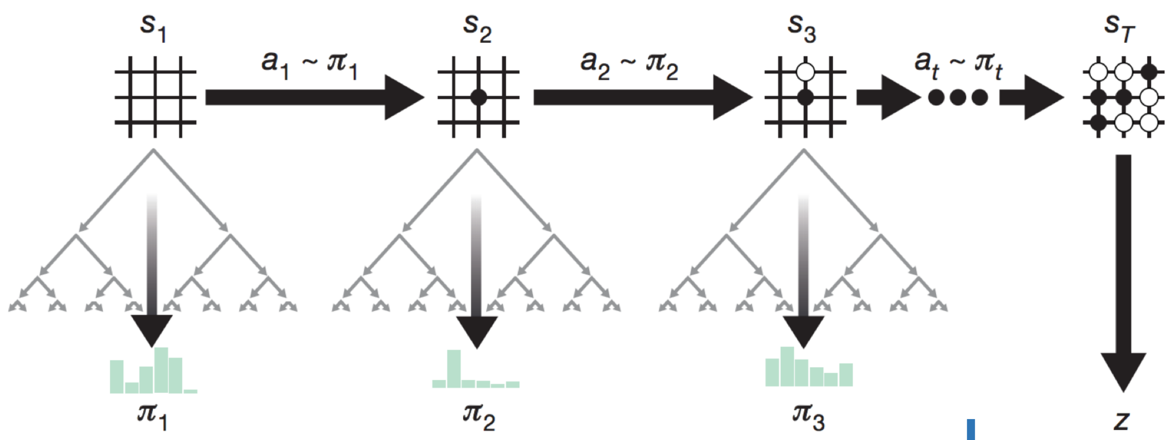
AlphaGo Zero
(Oct. 2017)

Impressive Results, No code, No model

Demystifying AlphaGoZero/AlphaZero

- Hard to reproduce
  - Details are missing in the paper
  - Huge computational cost (15.5 years to generate 4.9M selfplays with 1 GPU)
  - Sophisticated (distributed) systems.
- Lack of ablation analysis
  - What factor is critical for the performance?
  - Is the algorithm robust to random initialization and changes of hyper parameters?
  - How the ladder issue is solved?
- Lots of mysteries
  - Is the proposed algorithm really universal?
  - Is the bot almighty? Is there any weakness in the trained bot?

Reimplementation of AlphaGoZero / AlphaZero



Generate Training data

Zero-human knowledge

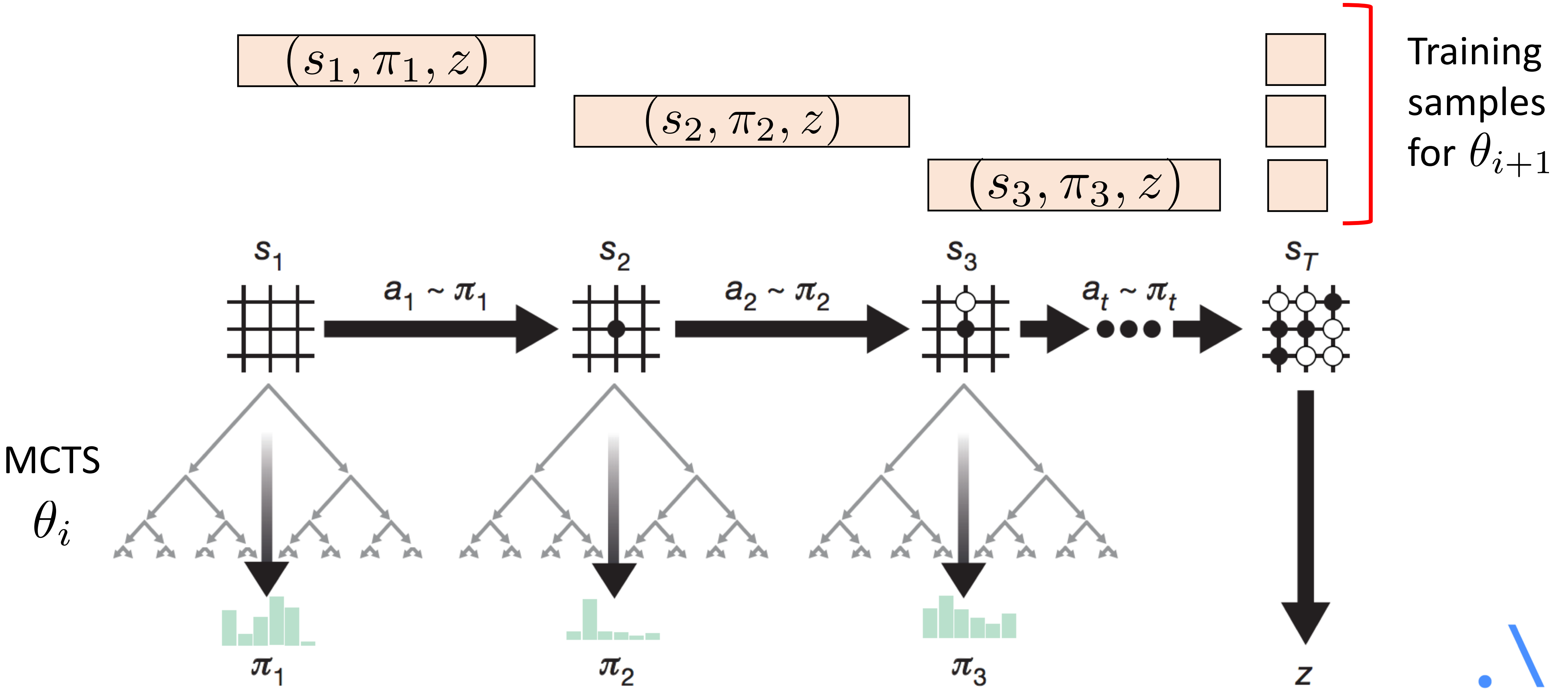
θ_i

Update Models

Self-Replays

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

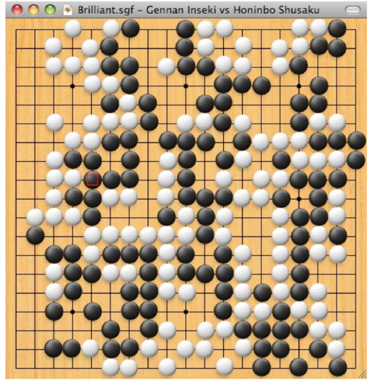
AlphaGo Zero



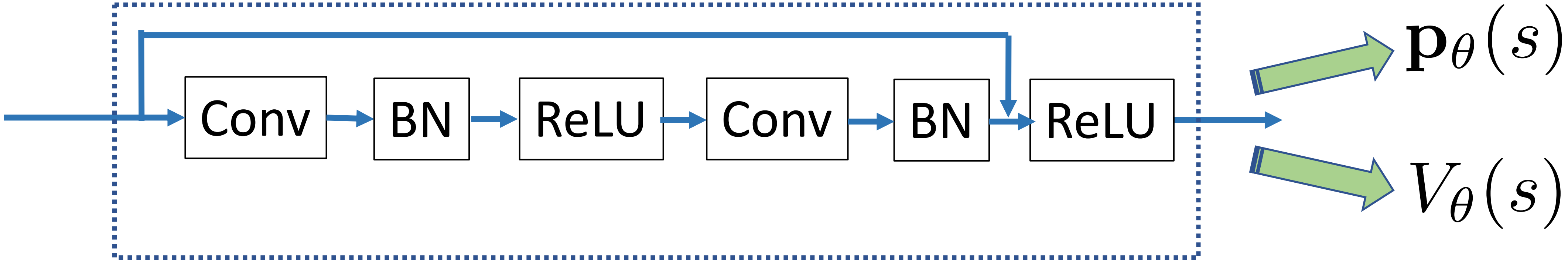
AlphaGo Zero

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

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



s



Player situation at time 0

Opponent situation at time 0

Player situation at t=-7

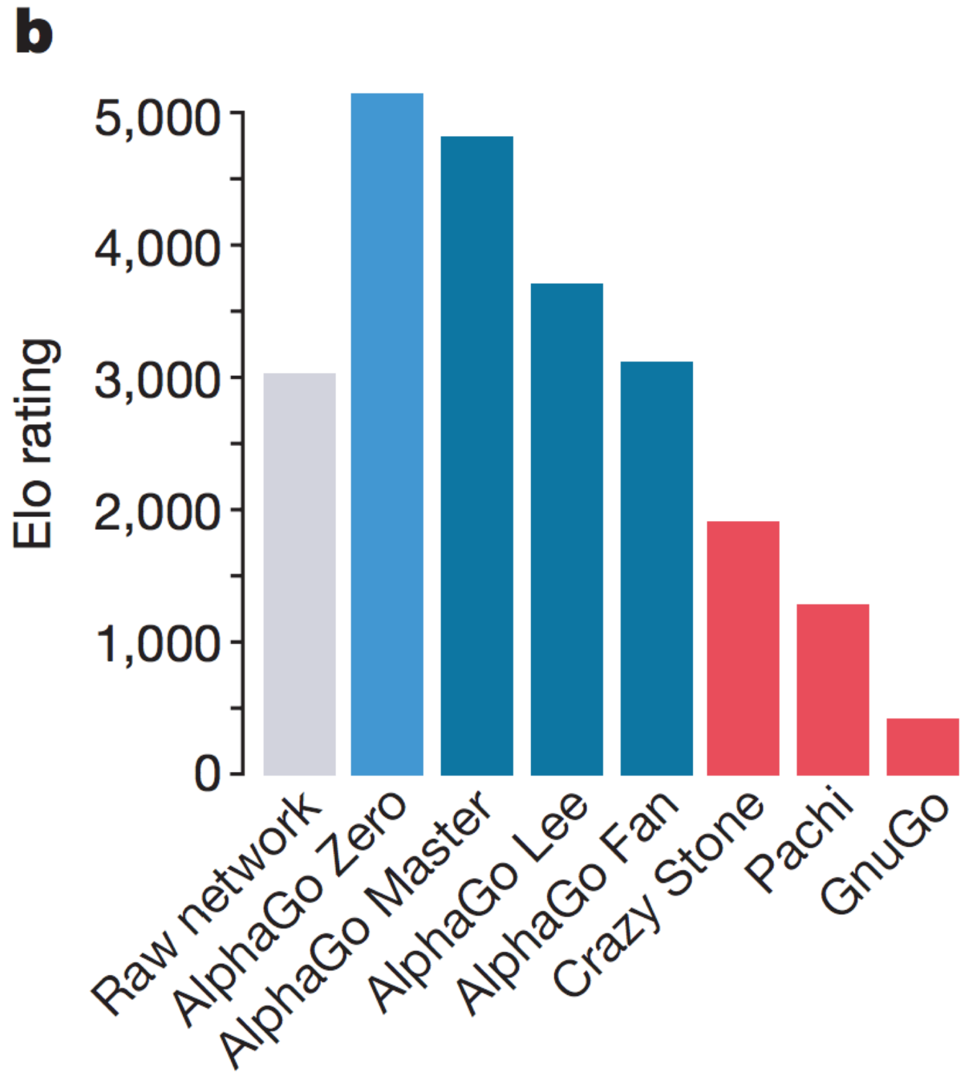
Color to play

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

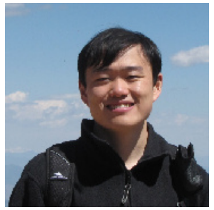
AlphaGo Zero Strength

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

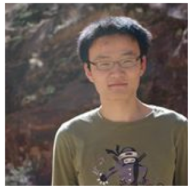
- 40 days version
  - 29M Games, 1600 rollouts/move
  - 40 blocks ResNet.
  - Defeat AlphaGo Master by 89:11



ELF OpenGo

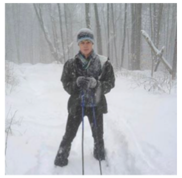

Yuandong Tian


Jerry Ma


Qucheng Gong

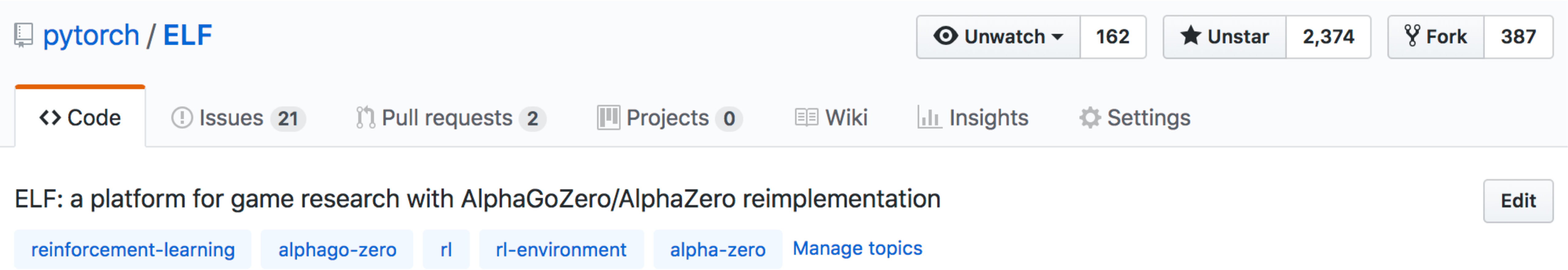

Shubho Sengupta

Zhuoyuan Chen


James Pinkerton

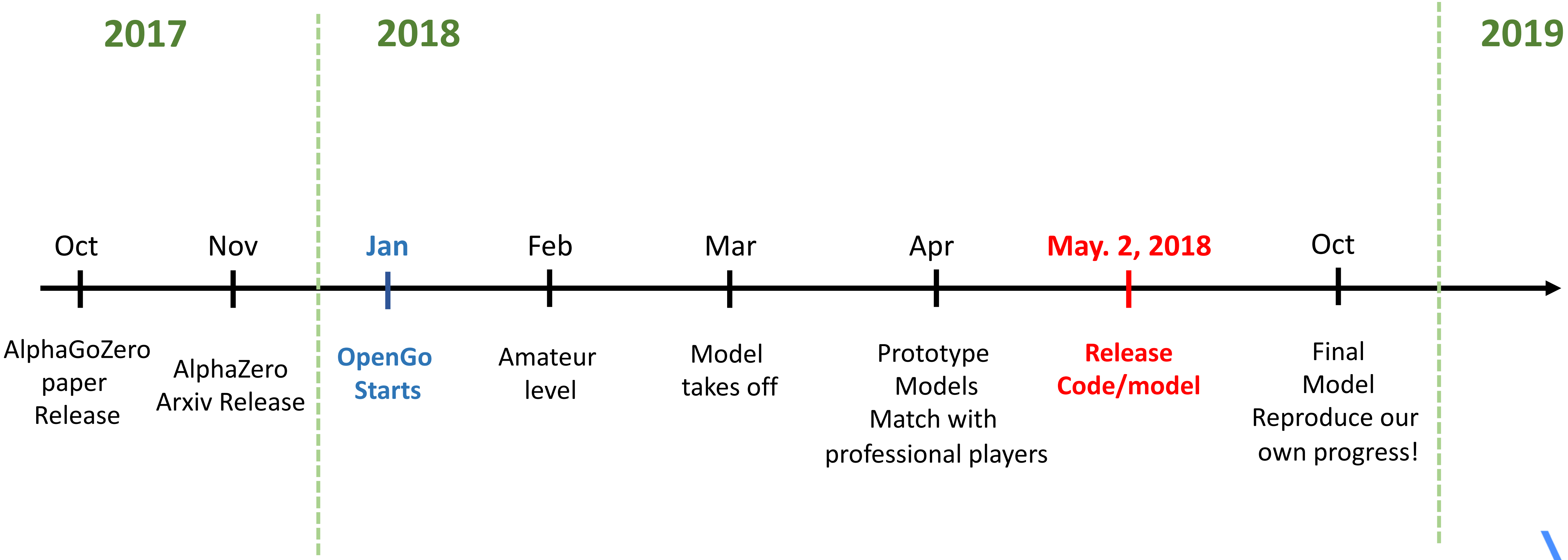

Larry Zitnick

- System can be trained with 2000 GPUs in 2 weeks (20 block version)
- Superhuman performance against professional players and strong bots.
- Abundant ablation analysis
- Decoupled design, code reusable for other games.



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

ELF OpenGo Timeline

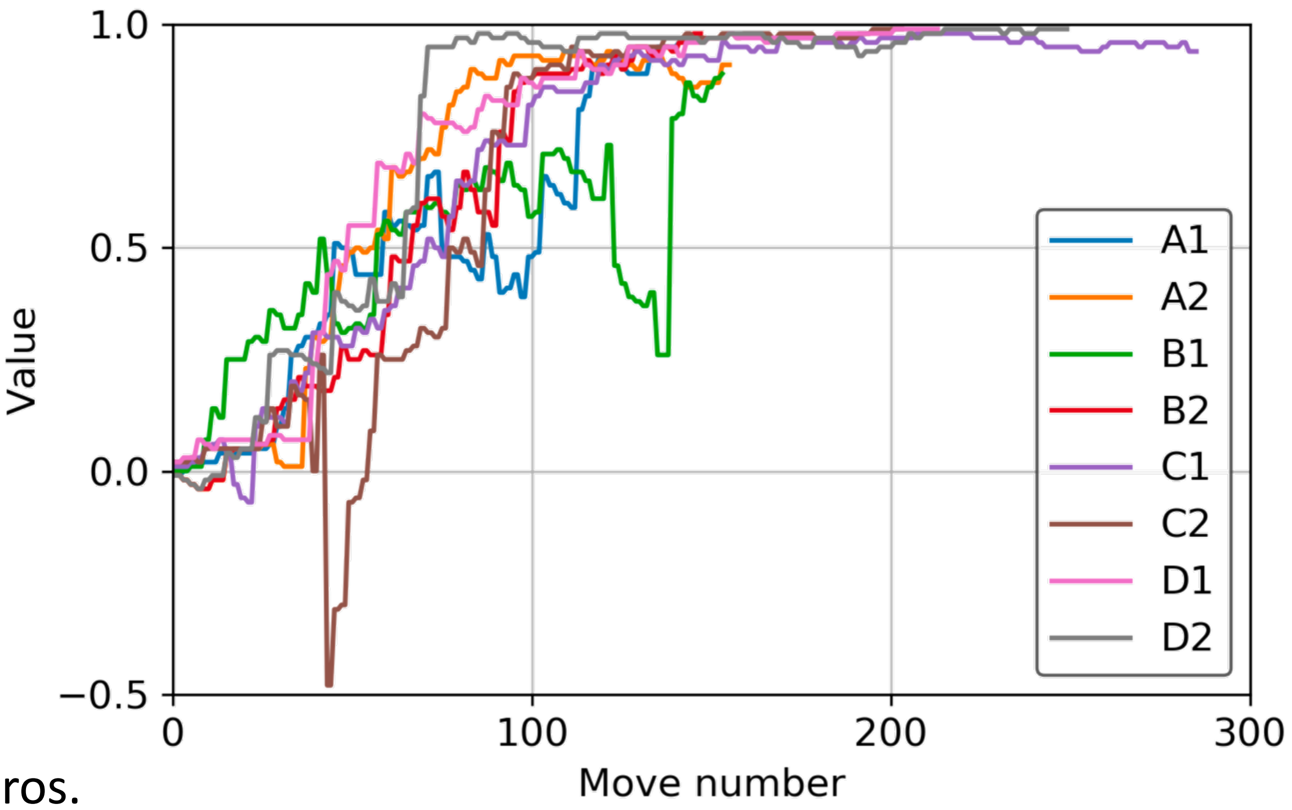


ELF OpenGo Performance

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



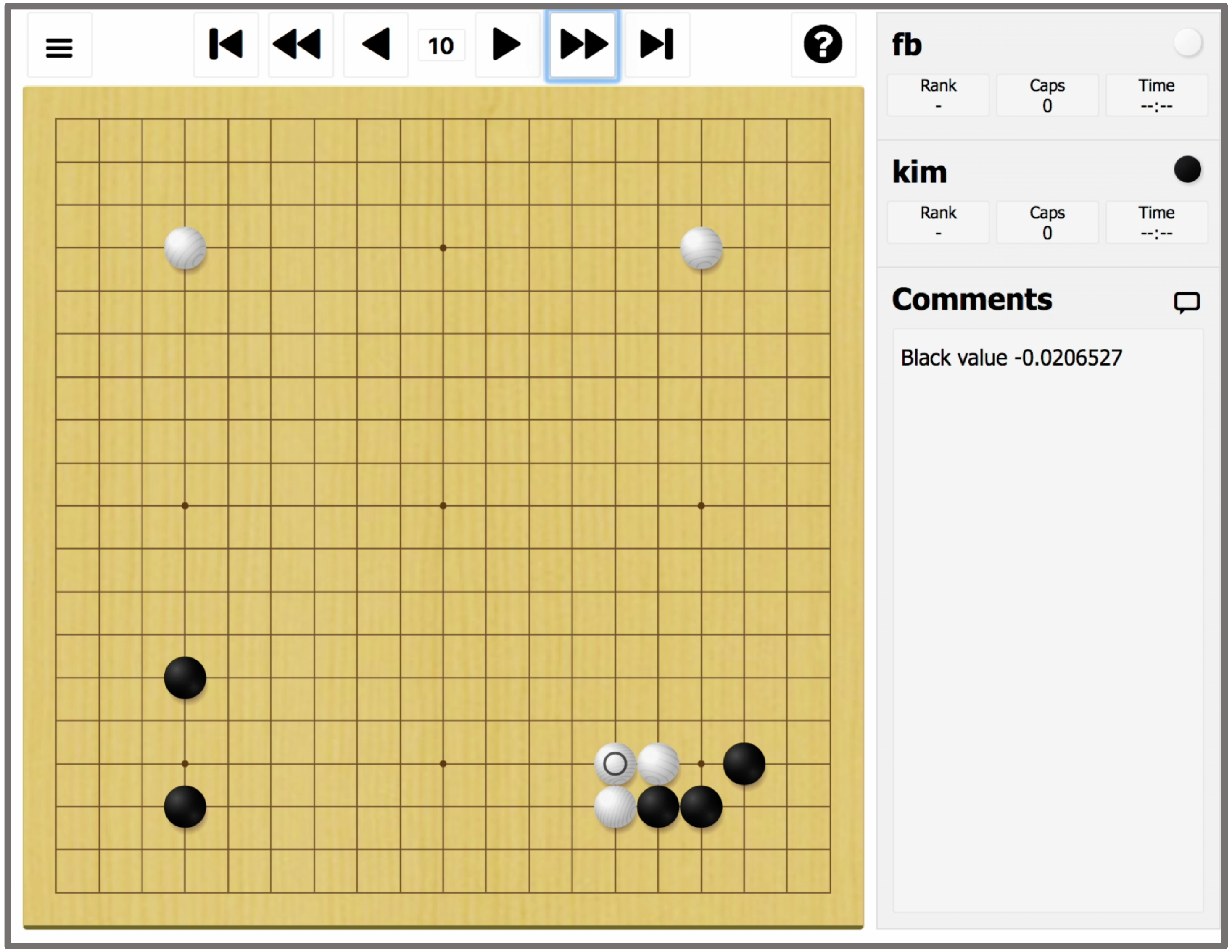
Vs professional players

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

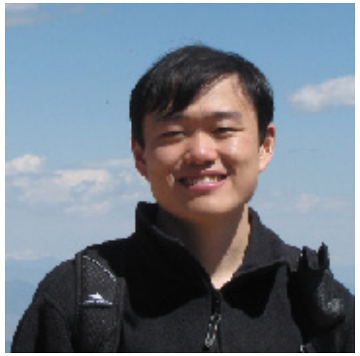
Vs strong bot (LeelaZero)

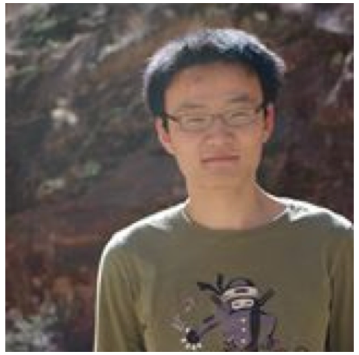
[158603eb, 192x15, Apr. 25, 2018]: 980 wins, 18 losses (98.2%)

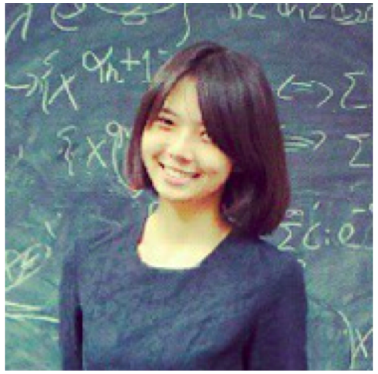
ELF OpenGo Sample Game

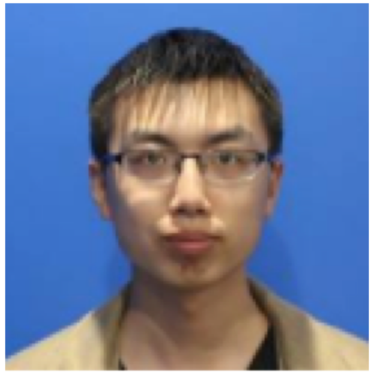


ELF: Extensive, Lightweight and Flexible Framework for Game Research


Yuandong Tian


Qucheng Gong

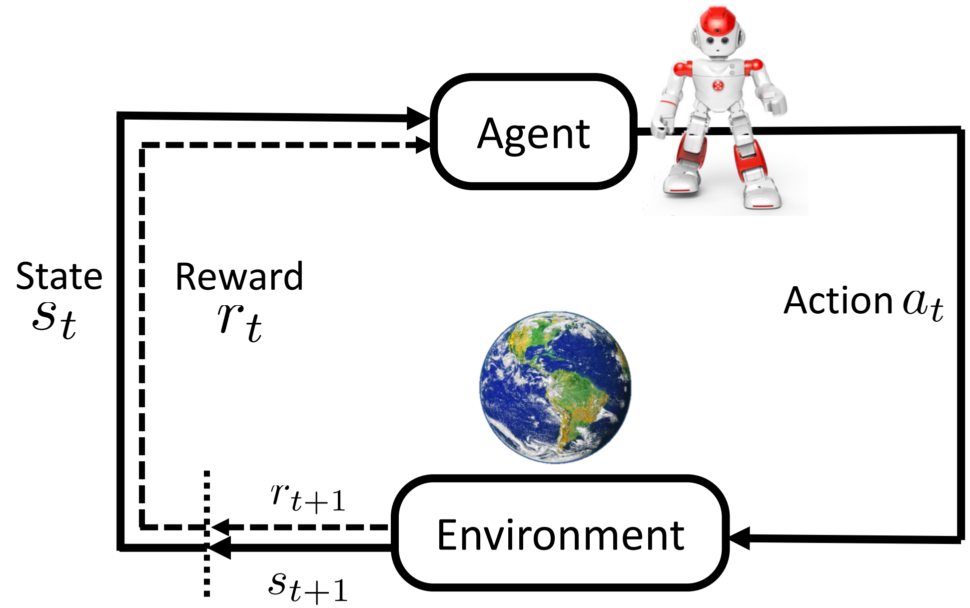

Wenling Shang

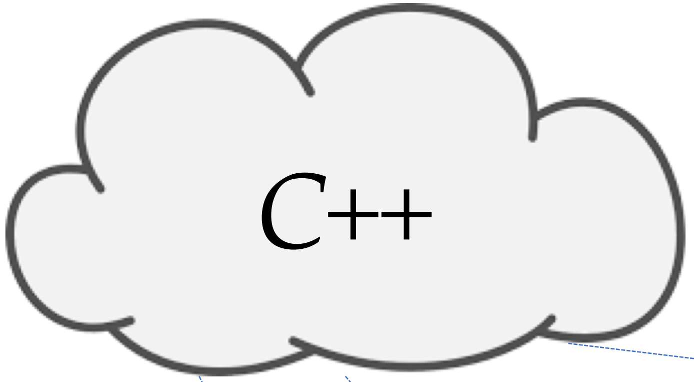

Yuxin Wu


Larry Zitnick

[Y. Tian et al, ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games, NIPS 2017]

ELF: A simple for-loop

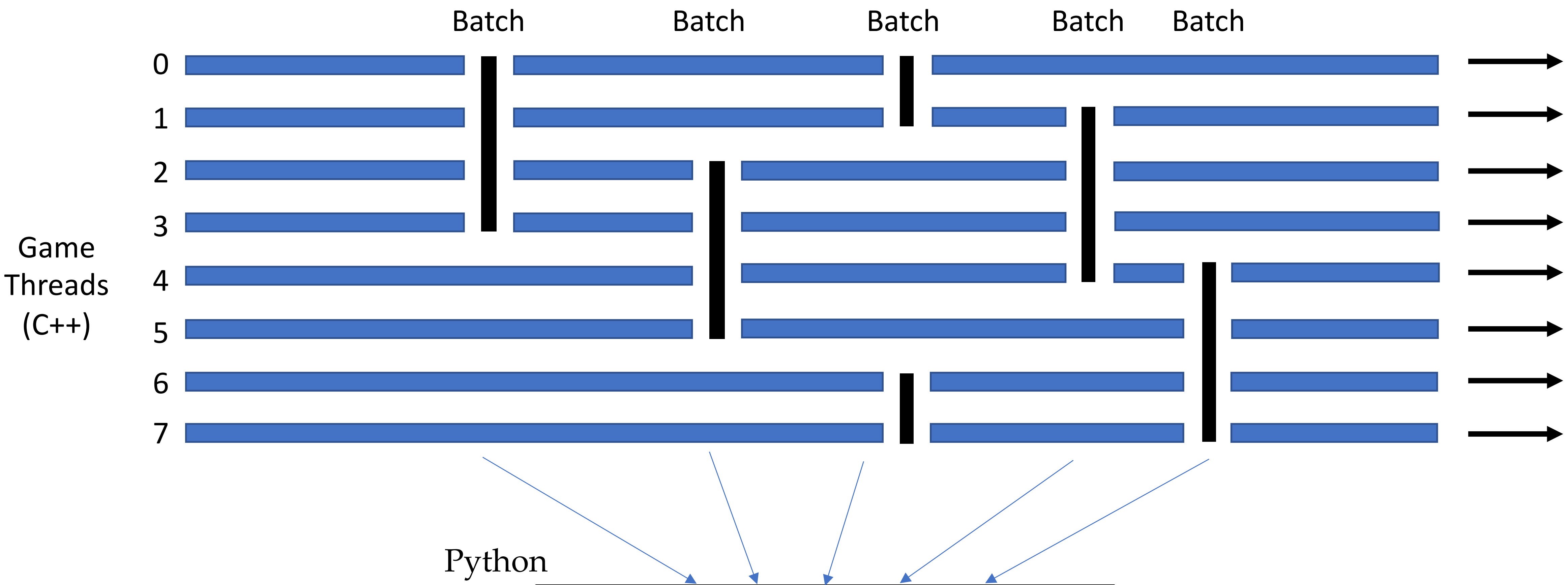




Python

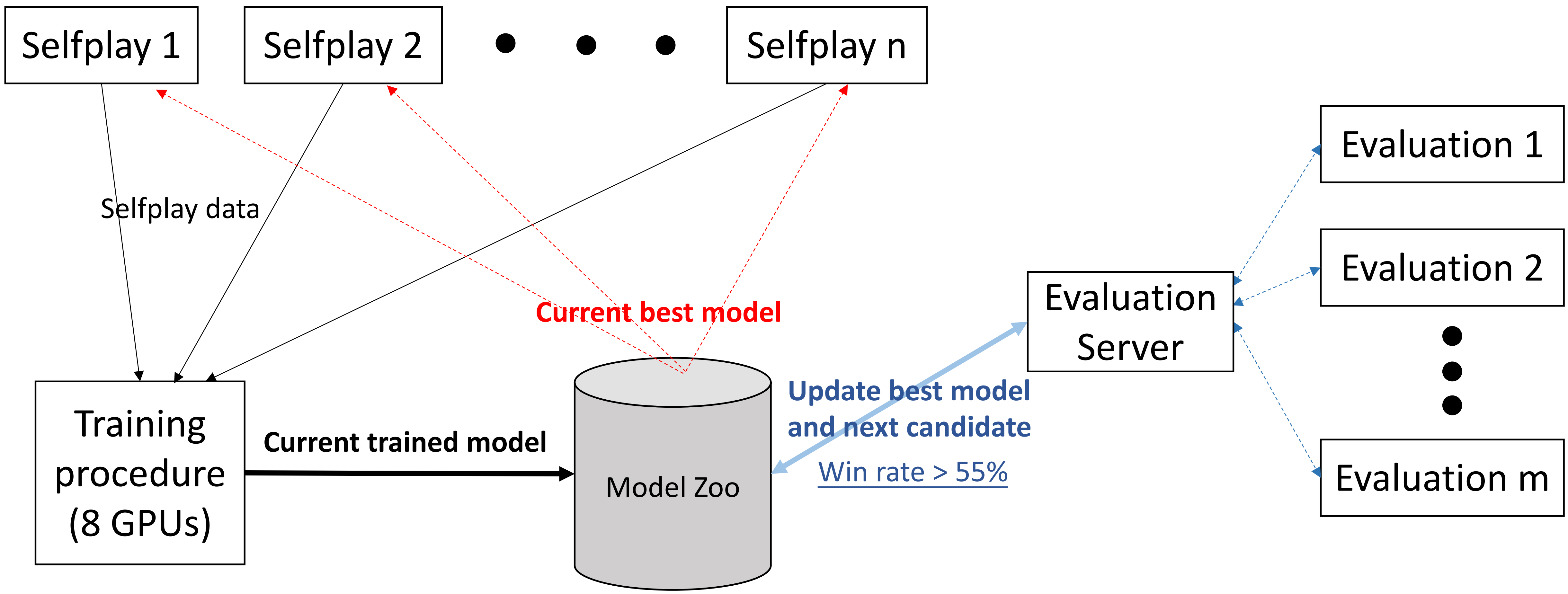
```python
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```

How ELF works

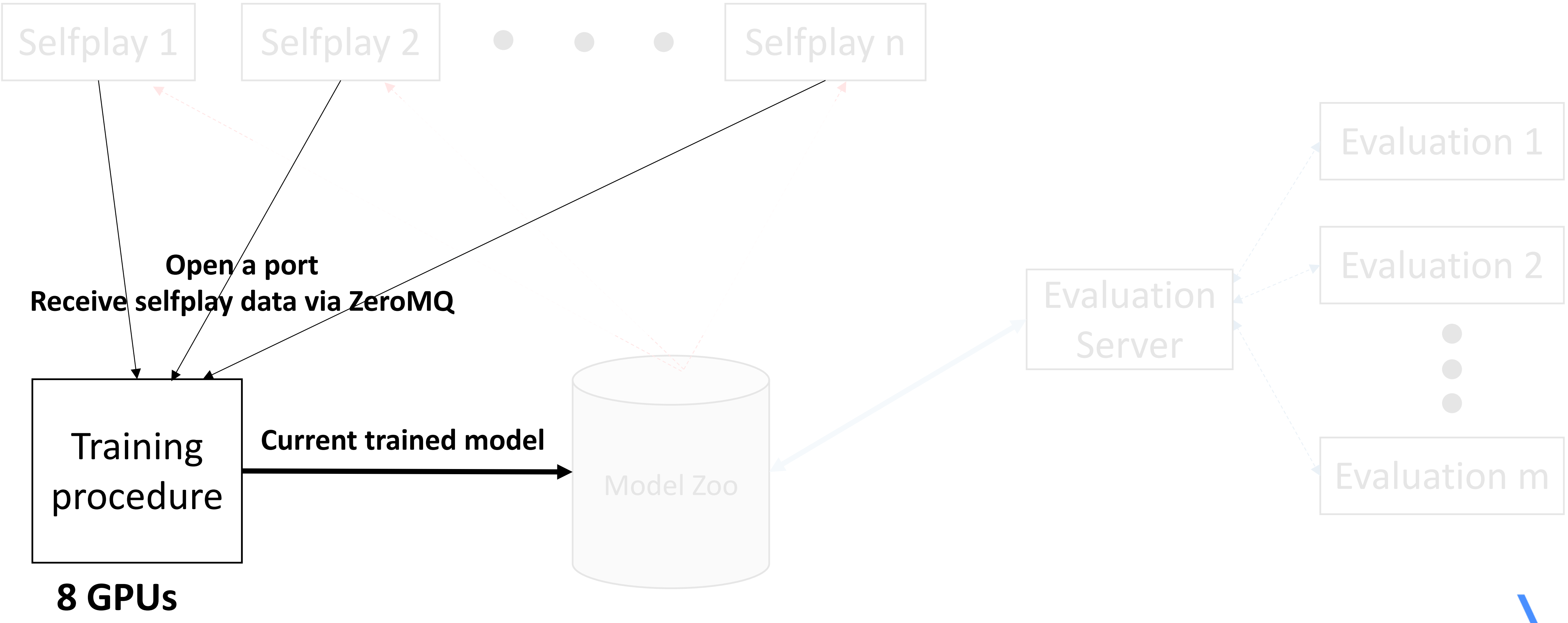



```python
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```

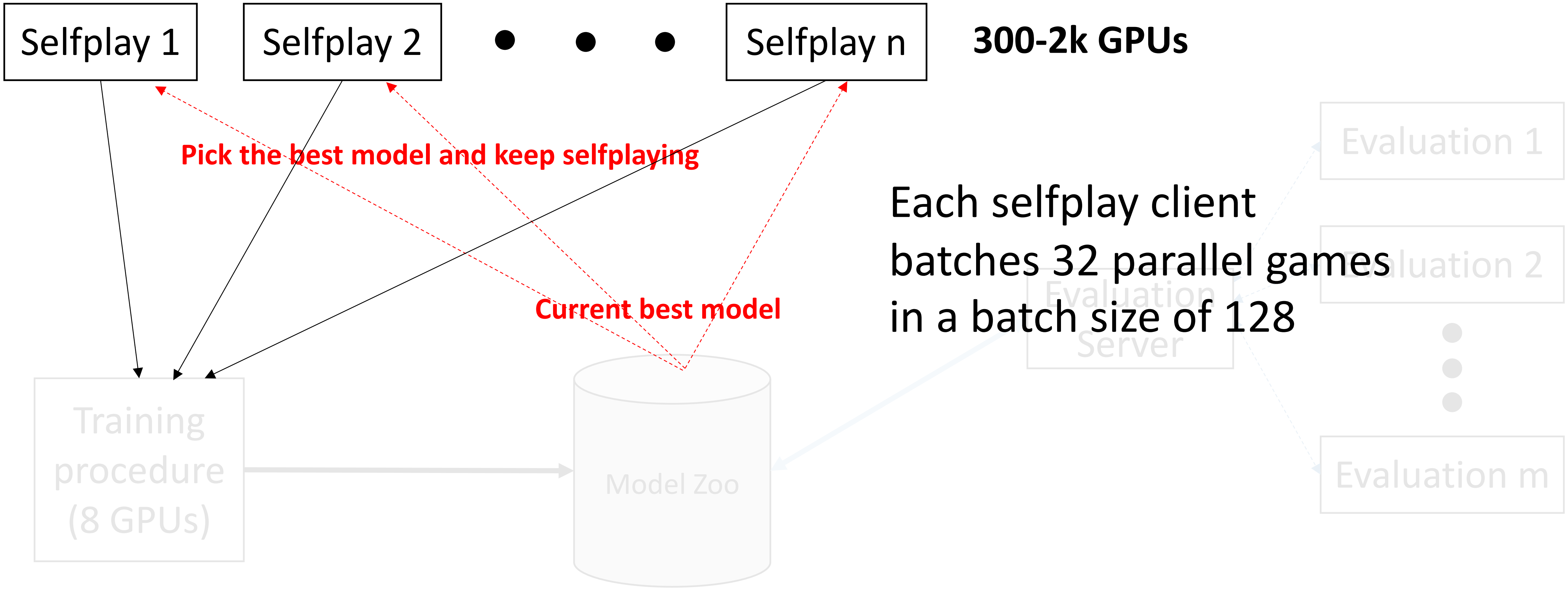
Distributed ELF (version 1)



Distributed System (version 1)

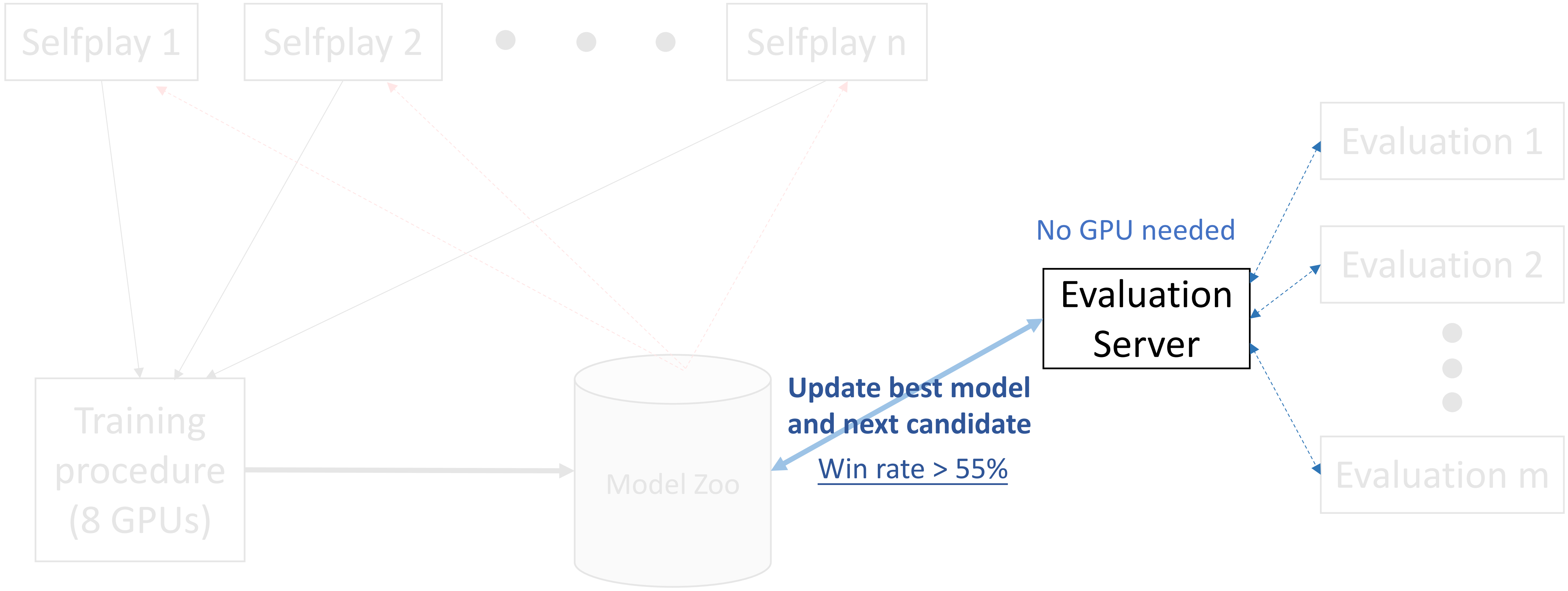


Distributed System (version 1)

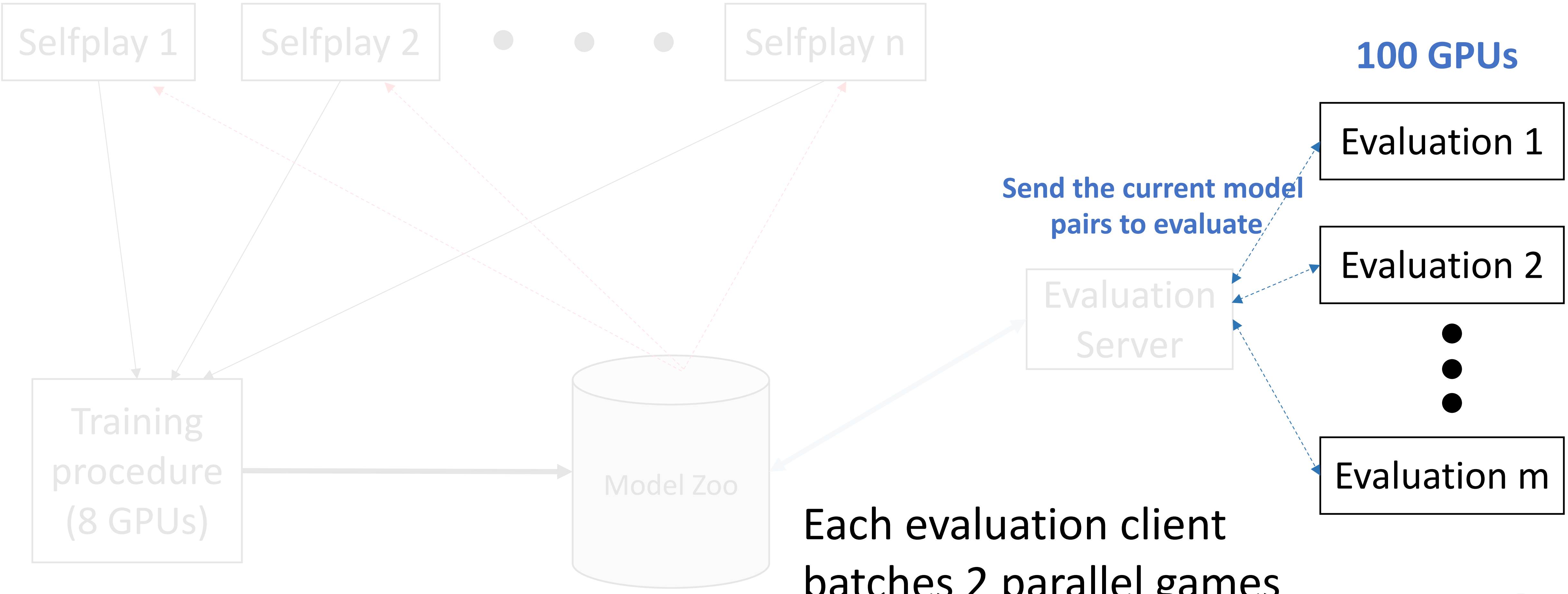


Each selfplay client batches 32 parallel games in a batch size of 128

Distributed System (version 1)

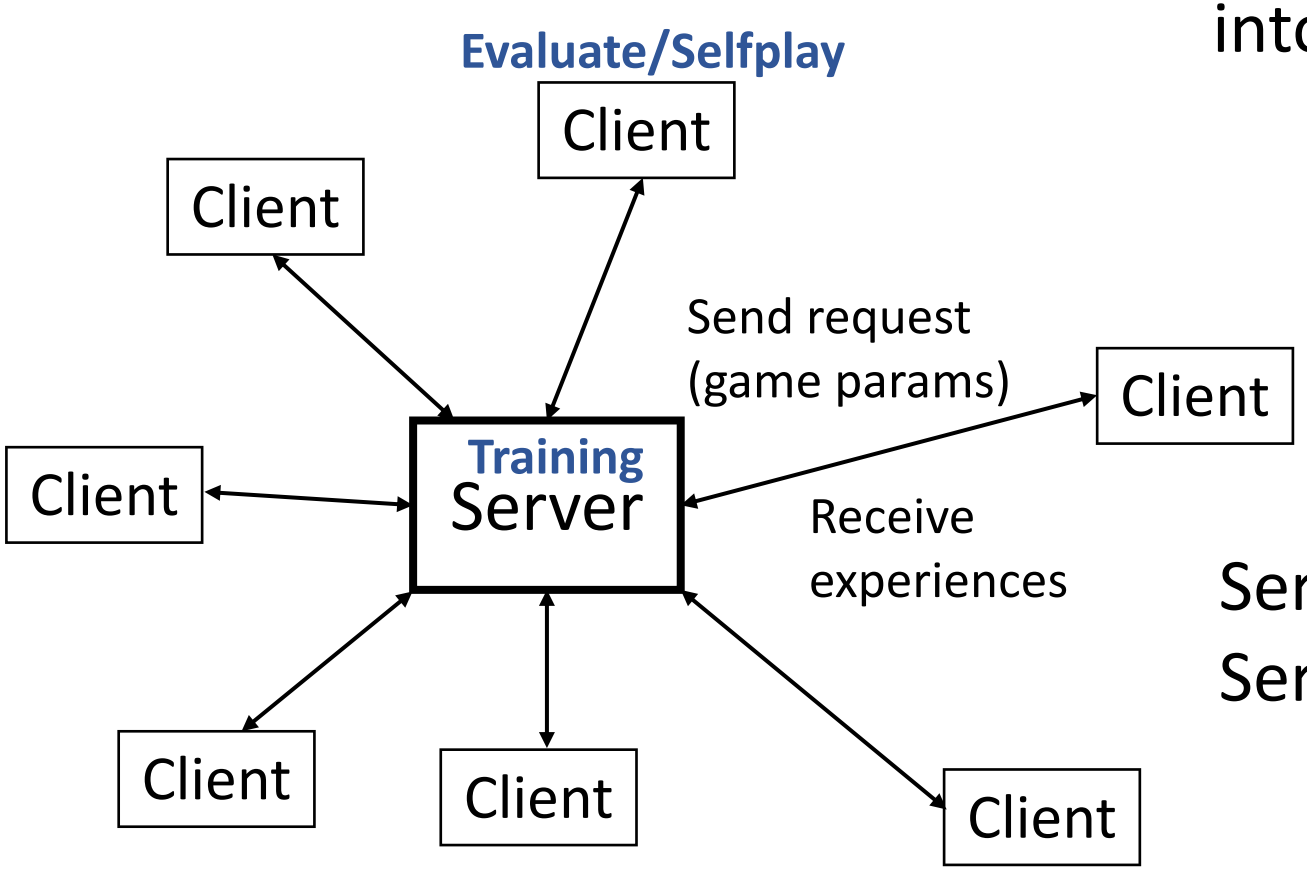


Distributed System (version 1)



Each evaluation client
batches 2 parallel games

Distributed ELF (v2)

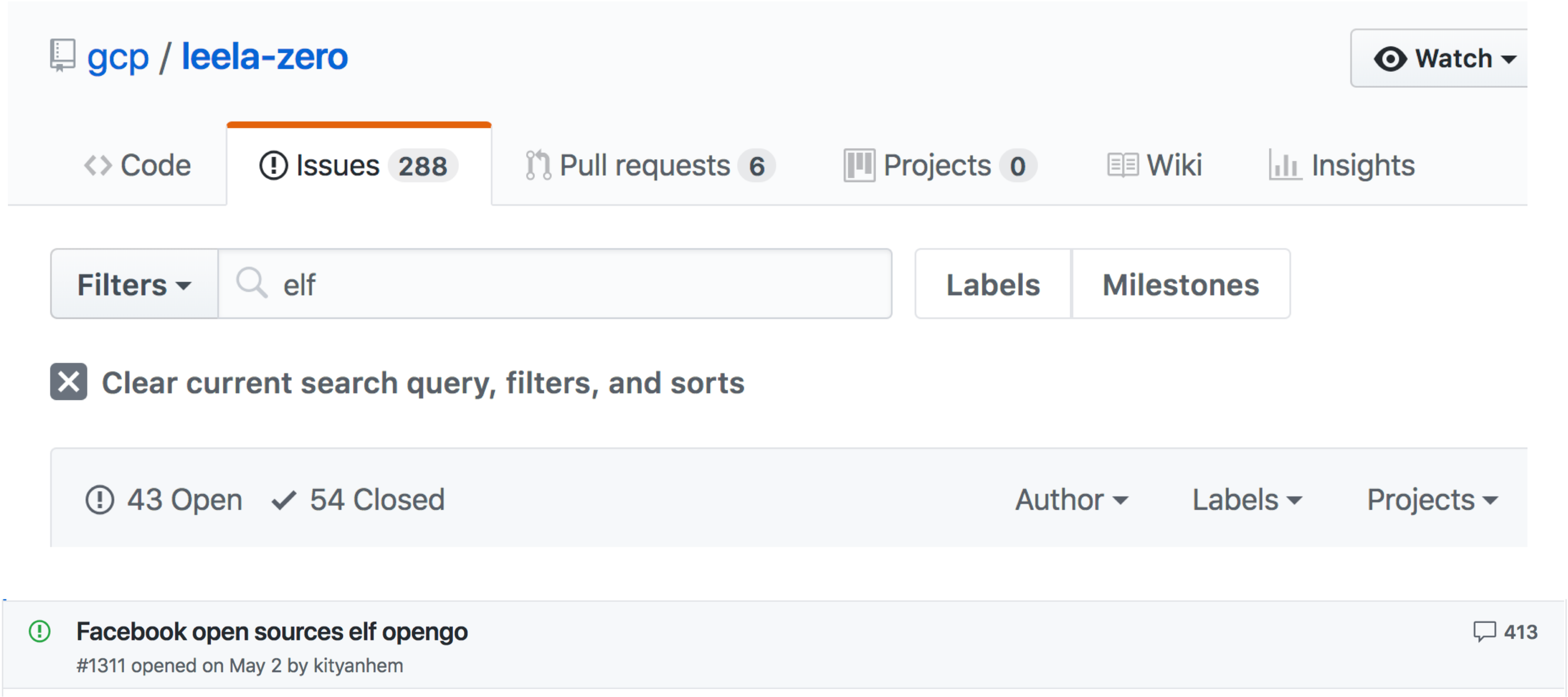


Putting AlphaGoZero and AlphaZero into the same framework

AlphaGoZero (more synchronization)
AlphaZero (less synchronization)

Server controls synchronization
Server also does training.

Adaptation


gcp / leela-zero
Watch
Code
Issues 288
Pull requests 6
Projects 0
Wiki
Insights
Filters
elf
Labels
Milestones
Clear current search query, filters, and sorts
43 Open
54 Closed
Author
Labels
Projects
Facebook open sources elf opengo
#1311 opened on May 2 by kityanhem
413

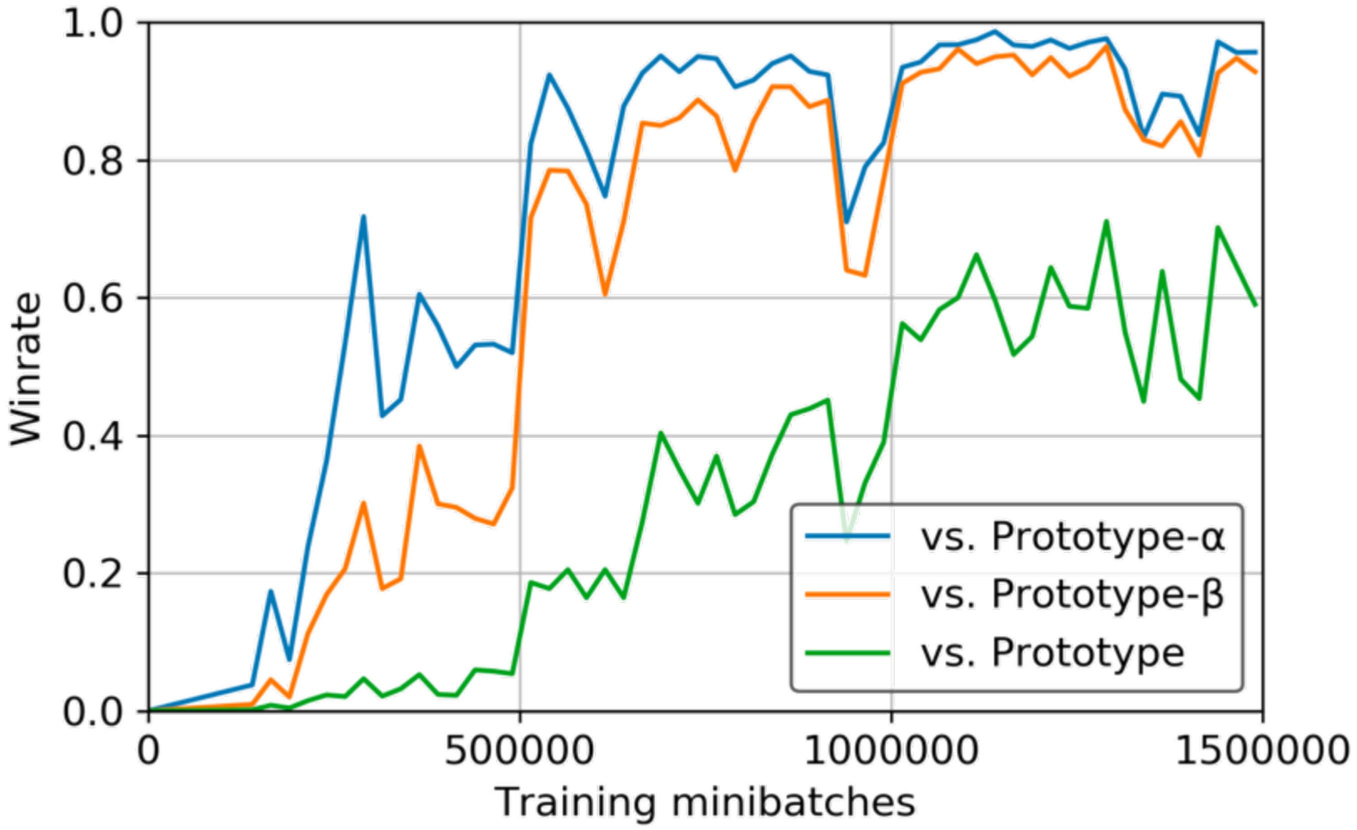
We put our bot on Fox server

野狐围棋

排名	用户名	段位	胜	负	月排行积分
1	骊龙	10段	165	0	7,573,934
1	ELFOpenGo	10段	90	10	4,681,775
1	金毛测试	10段	189	4	3,281,788
1	愿我能(孟泰龄六段)	9段	18	12	1,290,230
2	stealer	9段	11	6	1,214,060
3	印城之霸(辜梓豪九段)	9段	9	5	838,926

What we learned?

Training Stage of Final Model



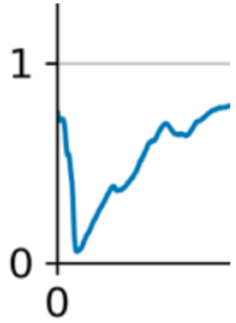
Prototype-α = strong amateur level

Prototype-β = professional level

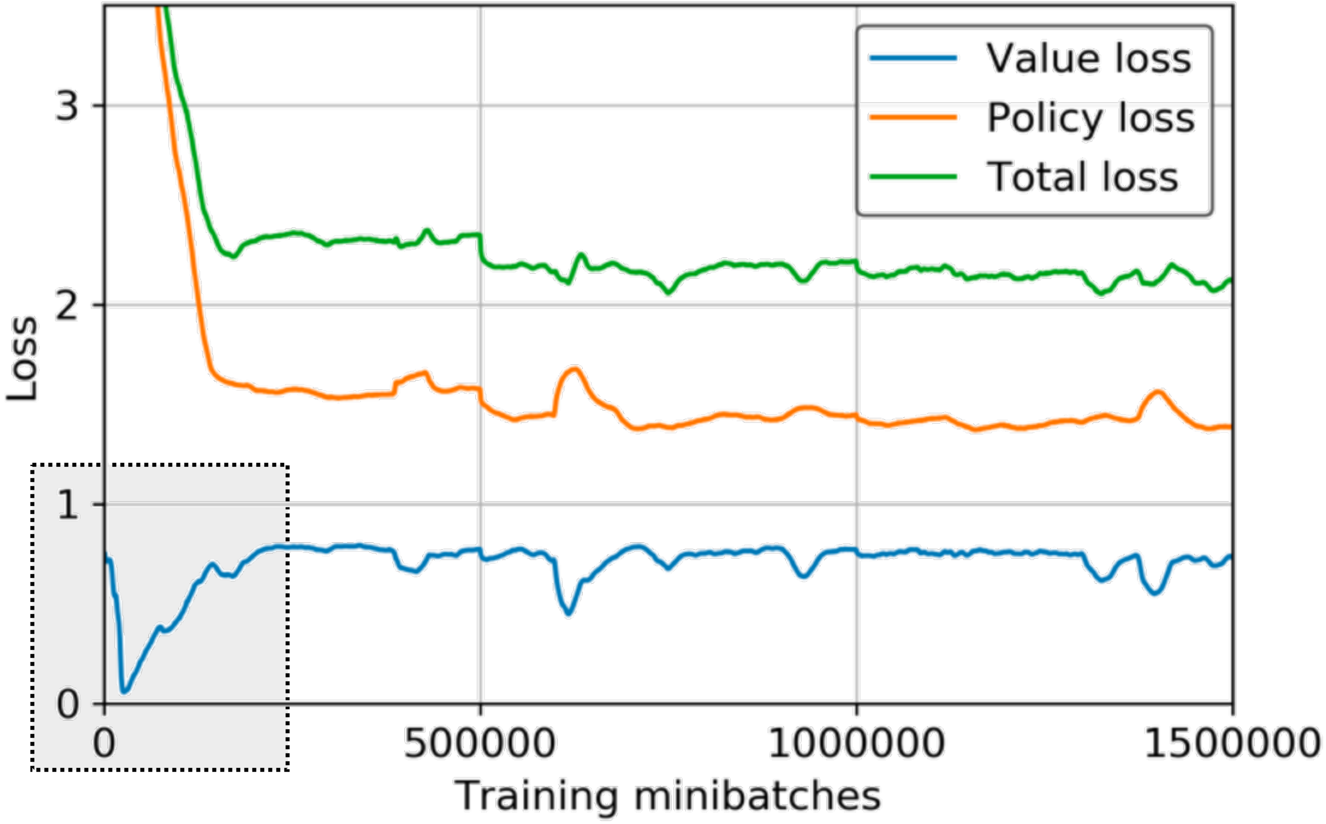
Prototype = superhuman level
(model against professional players)

A lot of zig-zag in the training process

Overfitting issues



Dip of the value function



Overestimate white winrate

Black resigns prematurally

Black loses many games

Imbalanced replay buffer

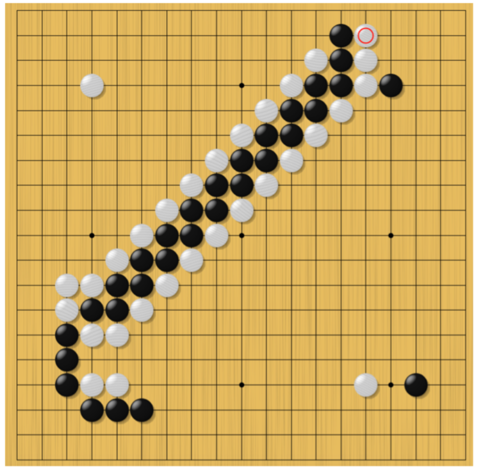
Large replay buffer is the key

Adaptive resign threshold has delays

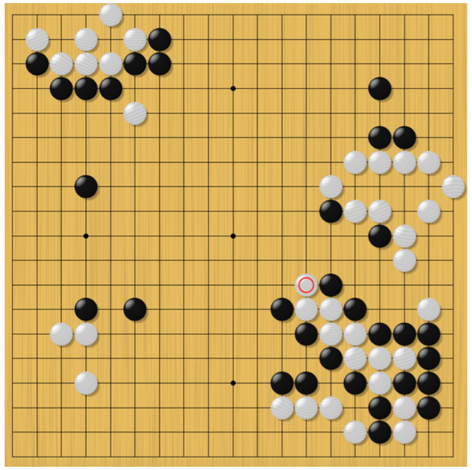
However, it is quite stable.

- Without policy head, it can still achieve ~2d level.
- With strong correlation in batch, it still train 1/3 of the time.
- With batchnorm with shifted mean/std, it still works to some extend.

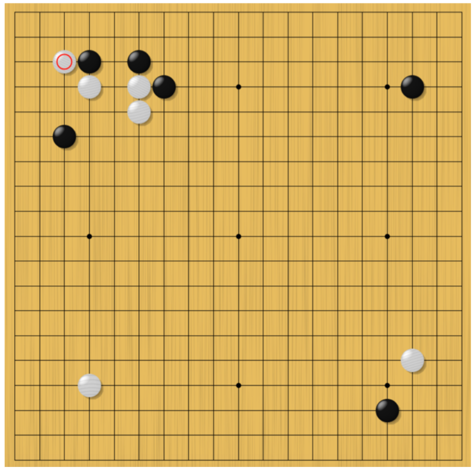
Ladder Issues



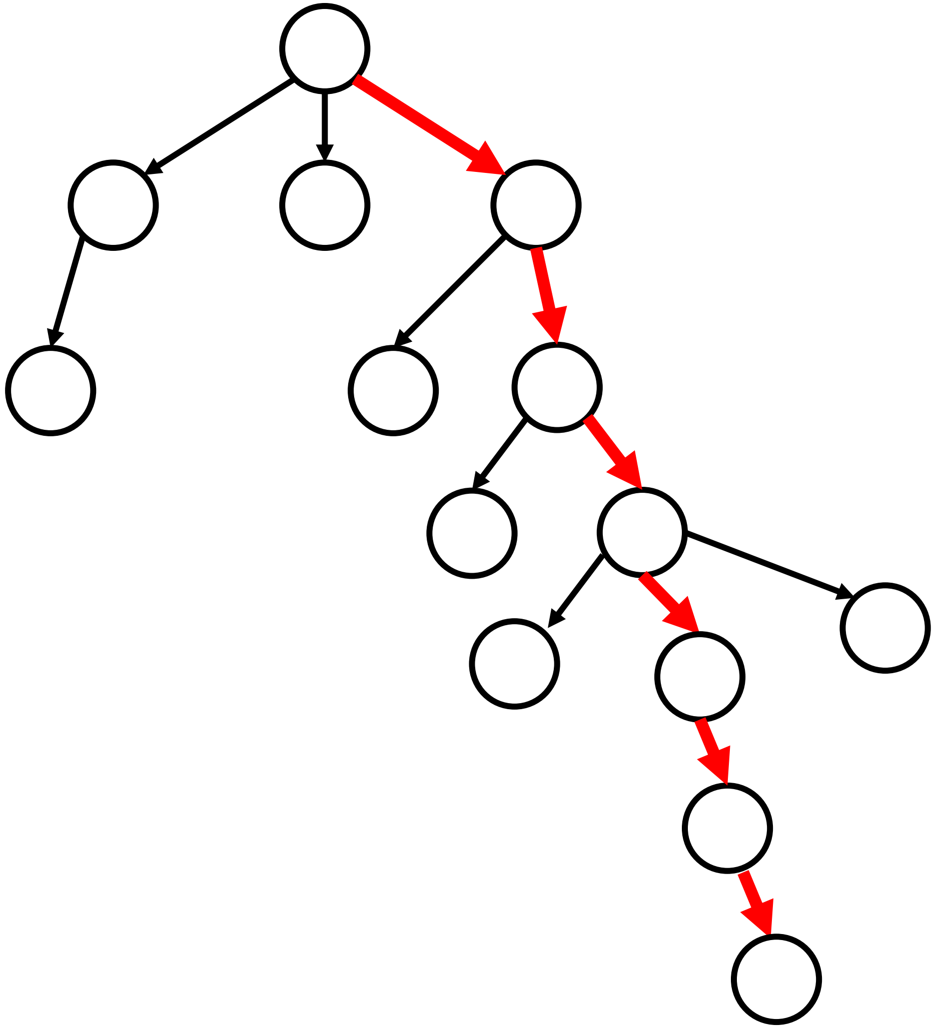
Run a ladder and lost



Run shorter ladder and lost



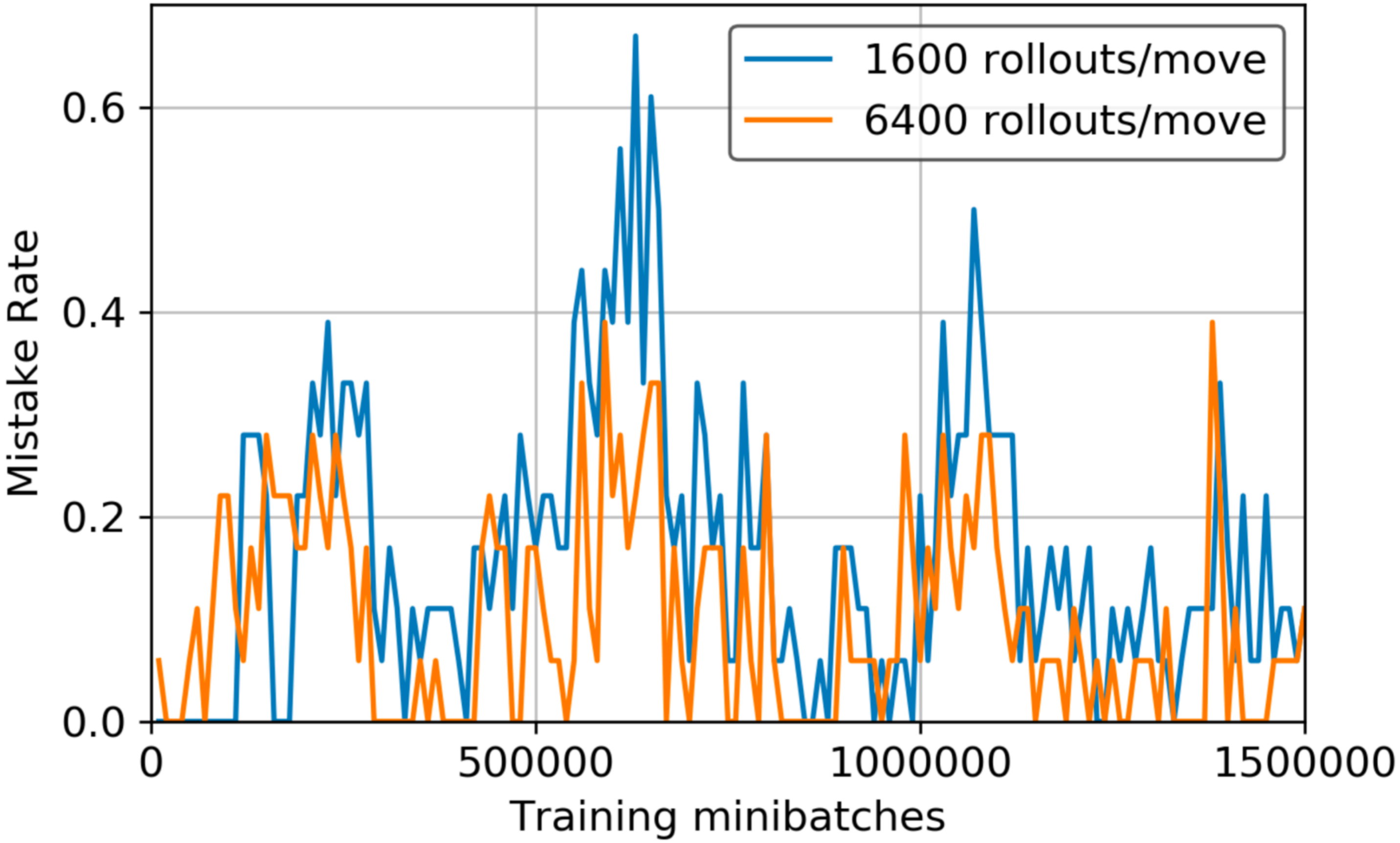
Doesn’t run ladder



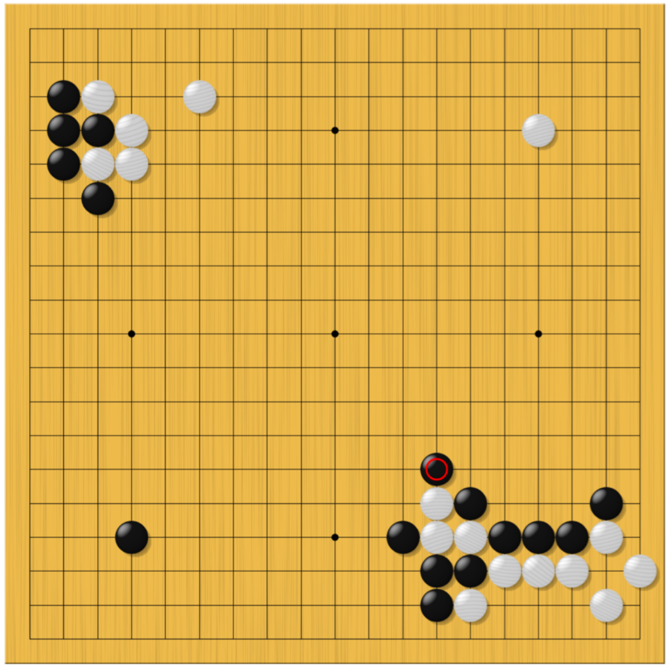
There is only one long path that is correct

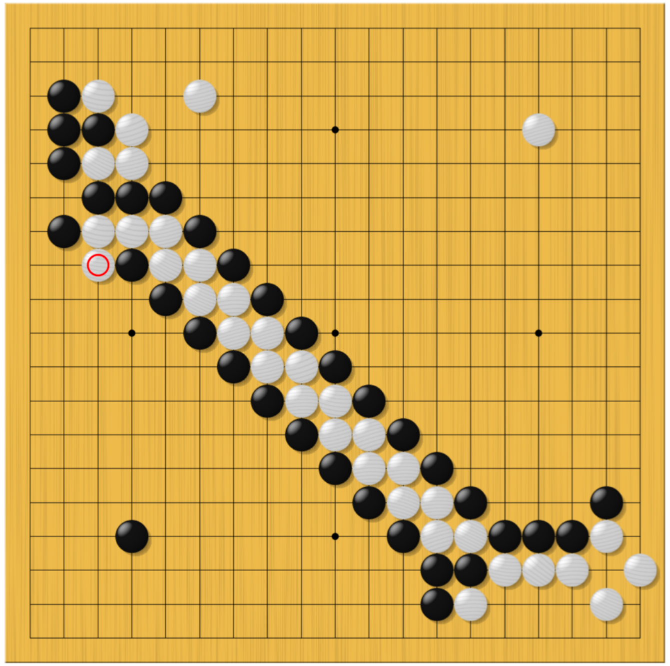
Value propagation is really slow.

Did we solve ladder?



No





Why is the model still strong? → It plays alternative moves to avoid these situations.

AlphaZero versus AlphaGoZero

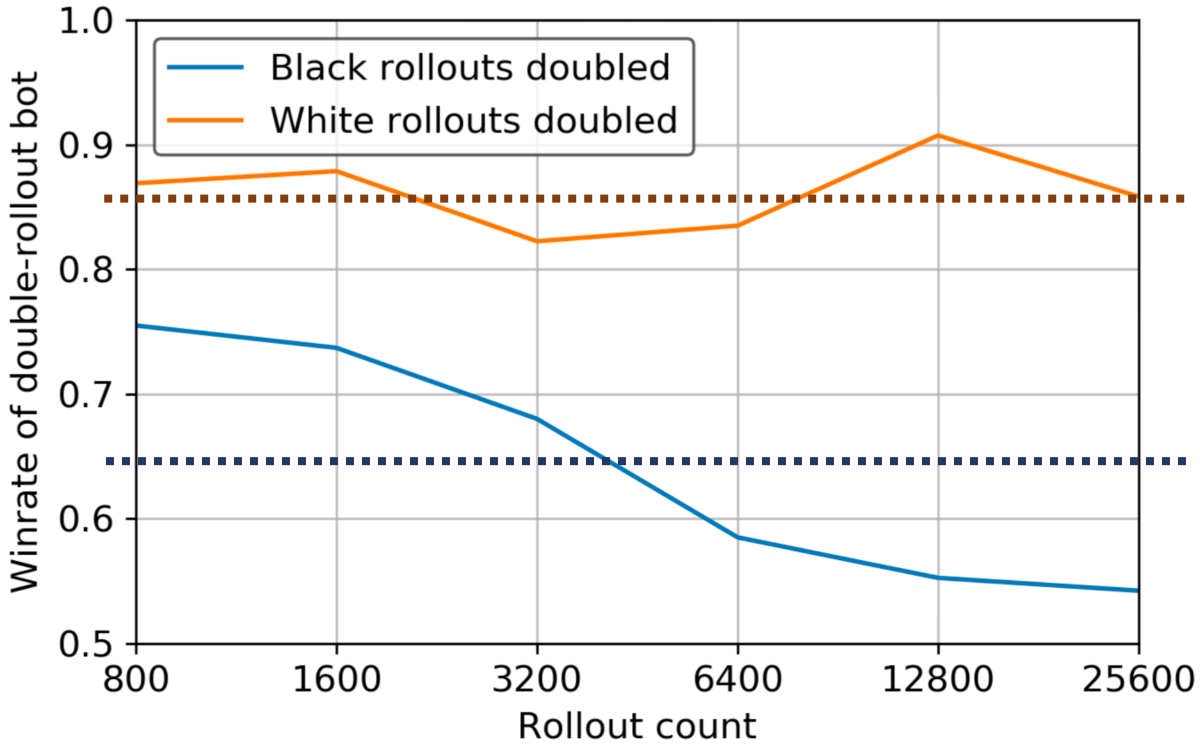
- AlphaZero is much faster than AlphaGoZero
  - No synchronization locks
  - After a day’s training, model trained with AZ won 100:0 against model trained with AGZ

- Essentially a value/policy iteration with function approximation.
  - No evaluation needed.

- Zig-zag slight overfitting which leads to improvement

Why MCTS is so important?

Look-ahead is how new knowledge is created.



On Final Model

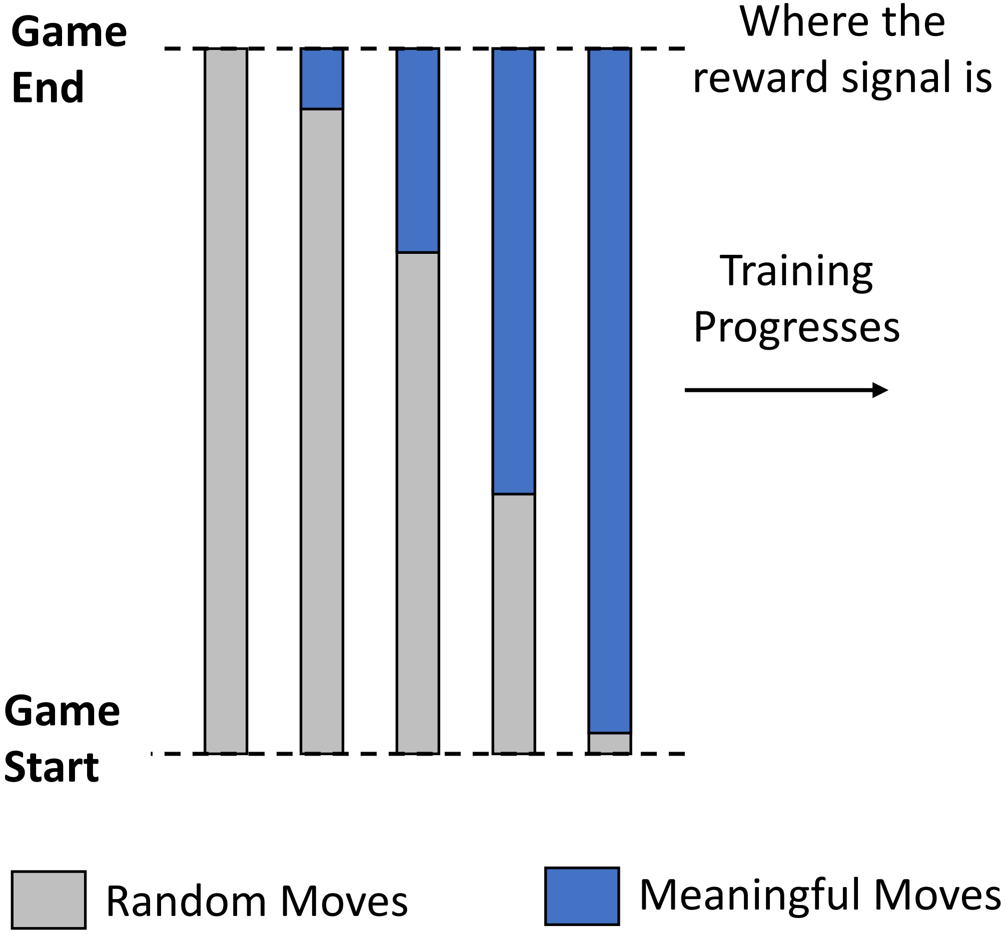
White rollouts 2x → ~85% winrate

Black rollouts 2x → ~65% winrate

Training is almost always constrained by model capacity (why 40b > 20b)

How sensible moves are learned?

Hypothetically

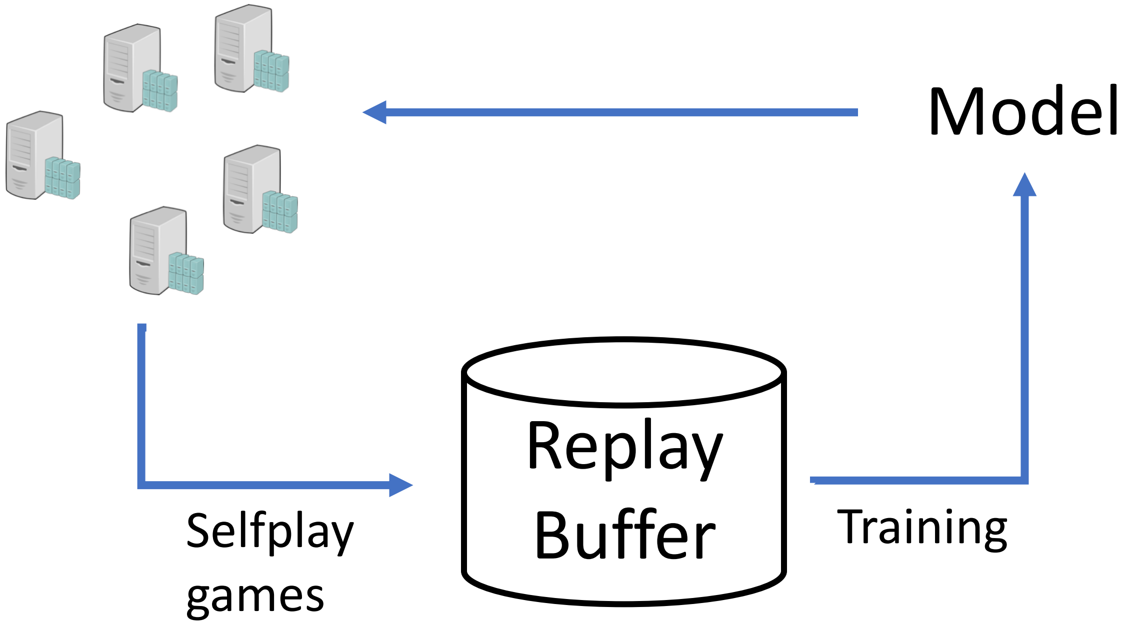


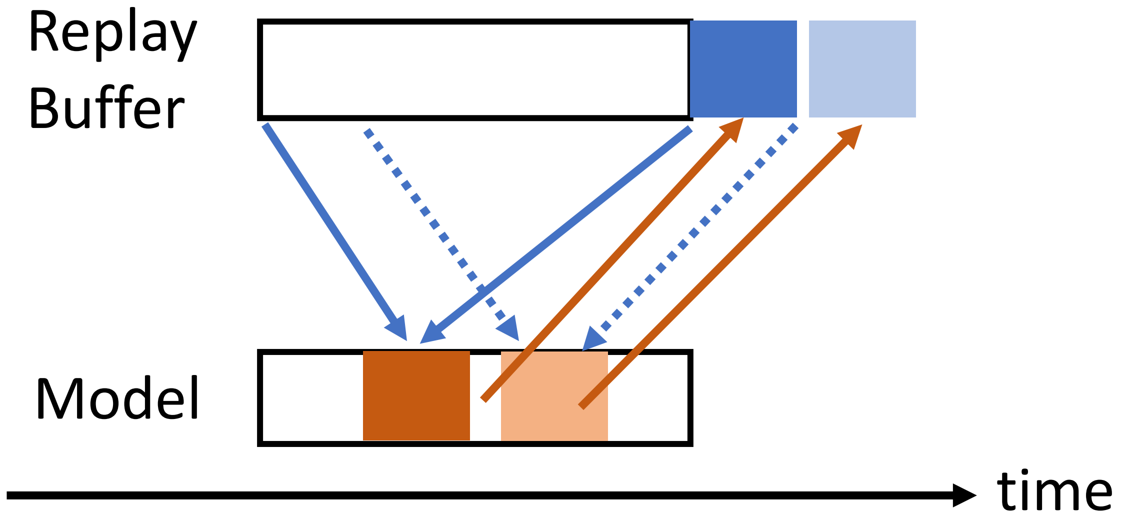
Practically

Match rate of each move against the *prototype* model.

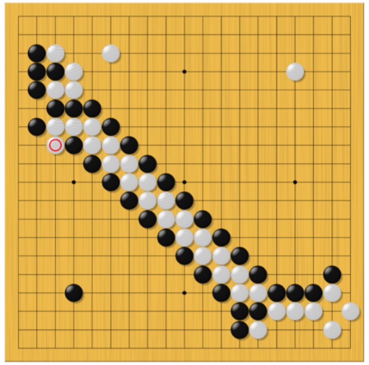
Further train with learning rate 10^{-5}...

- Surprisingly, it is not stable any more.
- Once at capacity, new models becomes similar to each other.
- Replay buffer becomes uniform and models start to overfit.





Conclusion



- The algorithm has pros and cons
  - Inductive bias
  - Planning is the key

- A lot of mysteries remain.
  - Why the method still works even with zig-zag and high-variance?
  - How to build a theoretical framework?
  - Maybe population-based approach is more stable?
  - More research to do

Challenge in Reproducibility

- How to reproduce a distributed ML/RL system like AlphaZero?
  - On-policy RL system does not have fixed dataset.
  - Distributed system poses more challenges.
- Practice
  - Fix the **random seeds**.
  - Record the script, the command argument and **git commit number**
    - Put the commit number into C++ library compilation.
  - Save the raw logs (stdout / stderr) and the script from raw logs to figures

Thanks!