



A Software Framework for AlphaZero-Like Applications

Thesis Defense

Presenter: Wei Li

Advisor: I-Chen Wu

Outline

- Introduction
- Background
- Design and Implementation
- Implementation Issues
- Experiments
- Conclusions and Future Work

Introduction

History of Computer Games AI

History of AlphaGo

Motivation and Goal

History of Computer Games AI

- Alpha-Beta Search:
 - Combine with other heuristics
 - Need lots of domain knowleage
 - E.g., Stockfish (chess engine), Elmo (shogi engine)
- Monte Carlo Tree Search (MCTS):
 - Works well for Go
 - Combine with Machine Learning techniques
 - Supervised Learning
 - Reinforcement Learning
 - E.g., Crazy Stone, MoGo, Zen
- MCTS + Deep (Reinforcement) Learning:
 - Achieve superhuman performance
 - E.g., AlphaGo [DeepMind], FineArt [Tencent], CGI [CGI Lab], ELF OpenGo [FAIR]

History of AlphaGo - AlphaGo

- **AlphaGo^[1]** is the first Go program beating Go champions without handicaps
 - Against Human Players
 - In 2016, defeats Lee Sedol in five matches (4:1)
 - In 2017, defeats Ke Jie in three matches (3:0)
 - Main Methods
 - Uses three deep convolutional neural networks (DCNNs)
 - **Supervised Learning (SL) Policy Network**: predict experts' moves
 - **Reinforcement Learning (RL) Policy Network**: improving playing strength of SL policy network
 - Uses **Policy Gradient^[2]** algorithm via self-play game positions
 - **Value Network**: estimate a value for given game position
 - Combines the DCNNs with MCTS
 - SL policy used for move expansion
 - Mixed **fast rollout policy** and **value network** for evaluation

[1] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *nature* 529.7587 (2016): 484.

[2] Sutton, et al. "Policy gradient methods for reinforcement learning with function approximation."

History of AlphaGo - AlphaGo Zero

- **AlphaGo Zero**^[1] is an improved version of AlphaGo
 - Main Methods
 - Uses one deep convolutional neural networks with two heads
 - The output of the tower is passed into two separate **heads**
 - **Policy Head**: predict experts' moves
 - **Value Head**: estimate a value for any given game position
 - Combines the two heads network with MCTS
 - Policy Head used for move generation
 - Value Head for evaluation
 - No rollouts are used
 - Without using any data from human games, stronger than any previous versions

[1] Silver, David, et al. "Mastering the game of Go without human knowledge." Nature 550.7676 (2017): 354.

History of AlphaGo - AlphaZero

- **AlphaZero (AZ)** [1][2] is a more generalized variant of the AlphaGo Zero (**AGZ**) algorithm
 - AZ is able to play shogi and chess as well as Go

[1] Silver, David, et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." arXiv:1712.01815 (2017).

[2] Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018) 7/60

History of AlphaGo - AlphaZero

- **AlphaZero (AZ)** [1][2] is a more generalized variant of the AlphaGo Zero (**AGZ**) algorithm
 - AZ is able to play shogi and chess as well as Go
 - Comparison with specialized programs
 - Chess: AZ vs. Stockfish
 - Only 6 lose of 1000 games
 - Shogi: AZ vs. Elmo
 - Winning 91.2% of games
 - Go: AZ vs. AGZ
 - Winning 61% of games

[1] Silver, David, et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." arXiv:1712.01815 (2017).

[2] Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018) 8/60

History of AlphaGo - AlphaZero

- **AlphaZero (AZ)** [1][2] is a more generalized variant of the AlphaGo Zero (**AGZ**) algorithm
 - AZ is able to play shogi and chess as well as Go
 - AZ algorithm is also available for all **two-player perfect information games**
 - **Two-player perfect information games**
 - Number of Players: 2
 - Result: win, loss or draw
 - Zero-sum: one player's loss is exactly the other player's gain, vice versa
 - Different Games:
 - Go
 - Chess
 - Chinese chess
 - Shogi (Japanese chess)
 - Gomoku

[1] Silver, David, et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." arXiv:1712.01815 (2017).

[2] Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018) 9/60

Motivation and Goal

- **Motivation**
 - Many researchers applied the AGZ/AZ algorithm to different games, but these projects are often developed independently
 - The limitations of specific games
 - Difficult to extend or transfer to other games
 - Many of code in AGZ/AZ algorithm can be shared
- **Goal**
 - Develop a software framework:
 - Easy to develop new games based on AGZ/AZ algorithm with a few of efforts
 - Allowed to be extended to a more general framework for other variations, such as
 - Different algorithms (e.g., alpha-beta), different networks (e.g., n-tuple networks)
- **Current Results**
 - We developed a framework named **CZF (CGI Zero Framework)**
 - Support both AGZ and AZ algorithms
 - Support any traditional two-player perfect information games
 - E.g., NoGo, Othello, Gomoku
 - Our NoGo program (**WaHaHaNoGo**) beat state-of-the-art program (**HaHaNoGo**)
 - Won the **championship** in TAAI 2018 tournament

Background

Deep Convolutional Neural Network (DCNN)

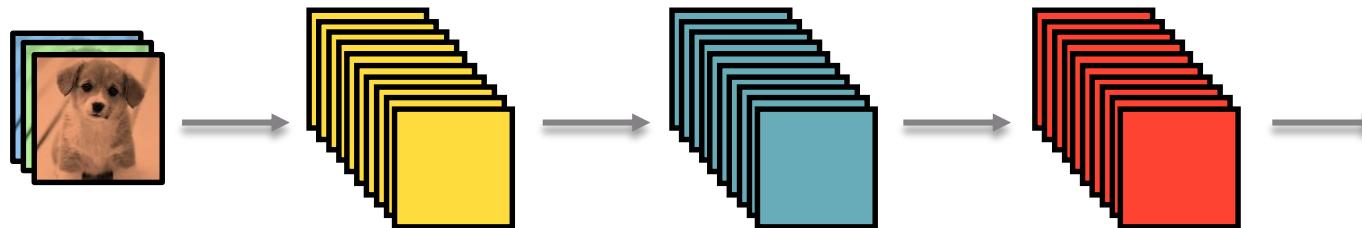
Monte Carlo Tree Search (MCTS)

Training Workflow of AlphaZero

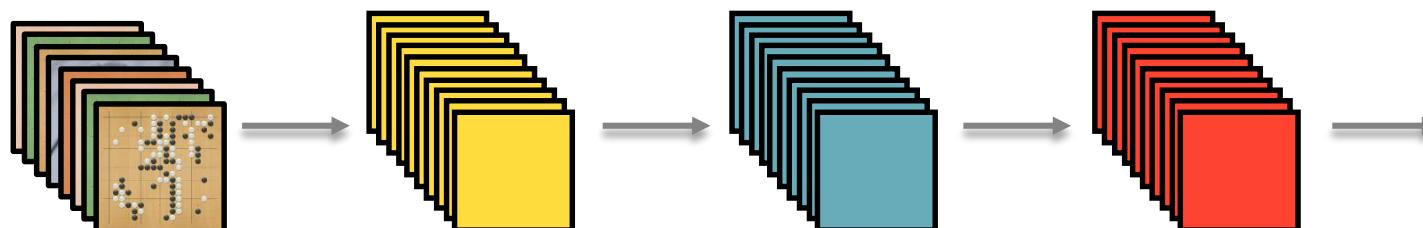
Related Work

Deep Convolutional Neural Network (DCNN)

- **Convolution Neural Network (CNN)** is a type of feed-forward artificial neural network
 - Yann LeCun introduced the concept of convolutional neural networks^[1]
 - CNNs are currently the **best-performing** method for many **classification** problems



- **CNN for Go:**
 - Input: **board features**
 - Output: **moves probabilities** (or **values of board**)

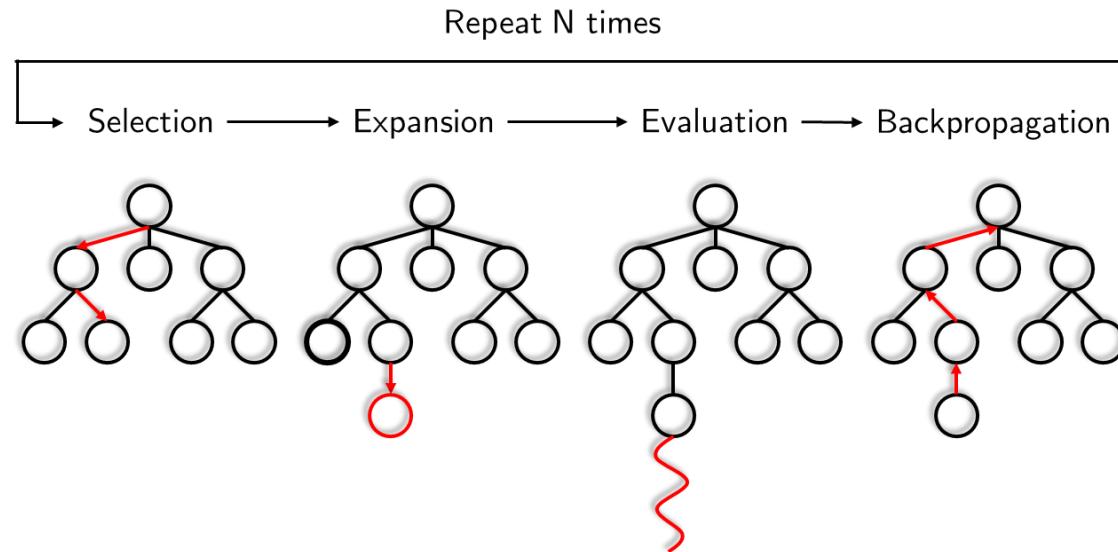


[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324

Monte Carlo Tree Search (MCTS)

- MCTS^[1] is a heuristic search algorithm for decision making
 - MCTS will repeat multi-times of simulations
 - The more simulations we perform, the more convincing the decision is
 - For each simulation, there are four stages:

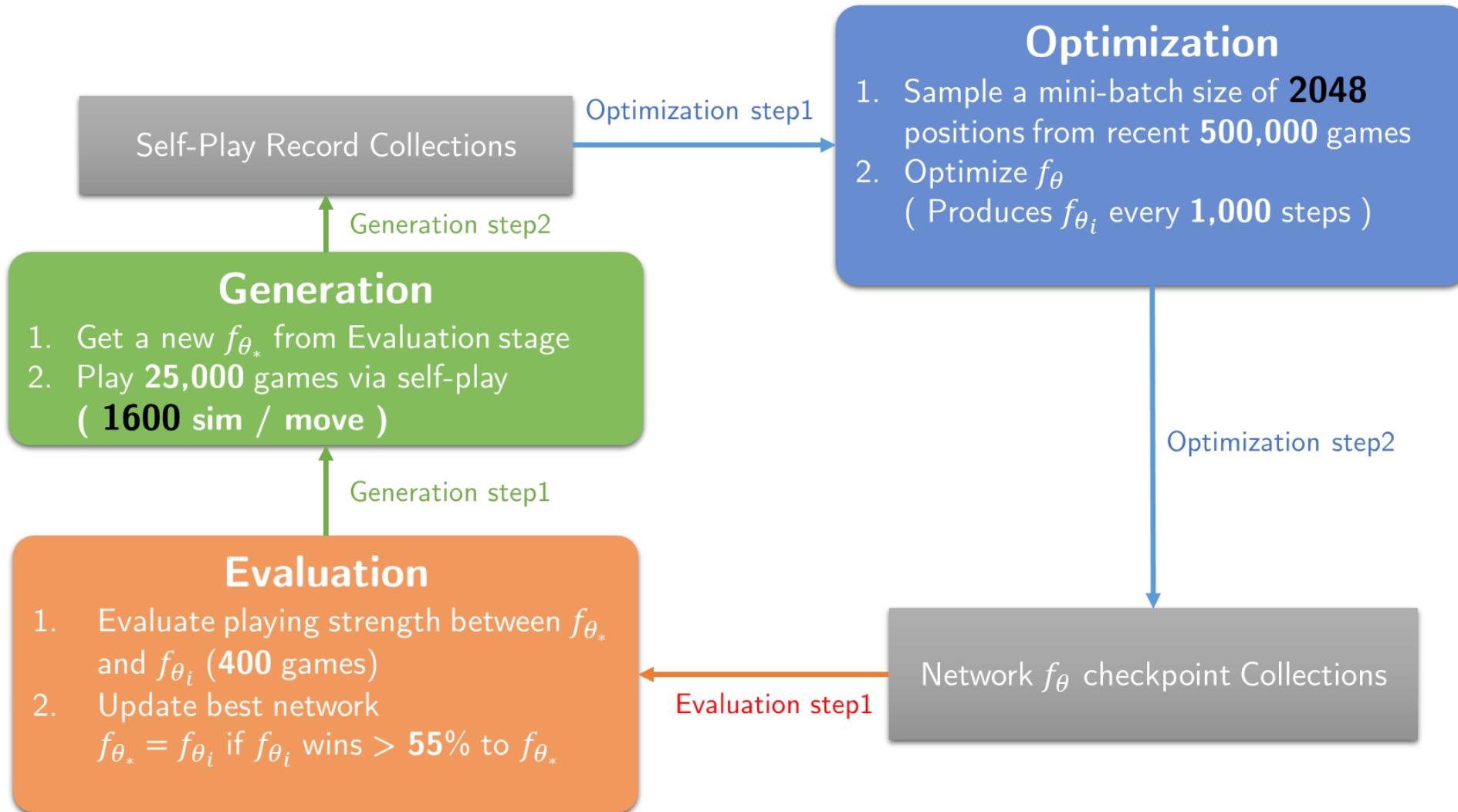
- Selection
- Expansion
- Evaluation
- Backpropagation



[1] Coulom, Rémi. "Efficient selectivity and backup operators in Monte-Carlo tree search." International conference on computers and games. 2006. 13/60

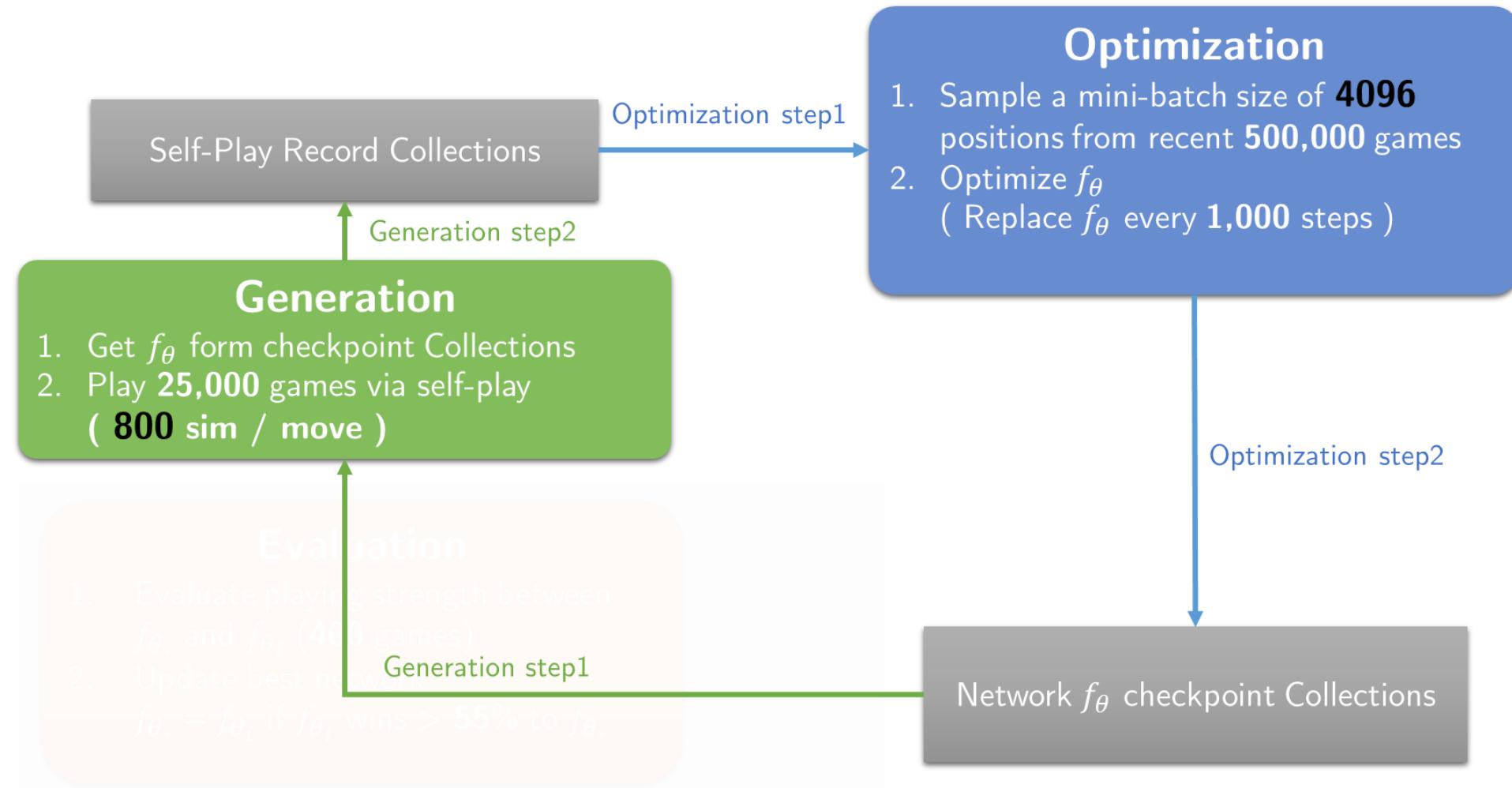
Training Workflow of AlphaZero

- **AlphaGo Zero**
 - Generation
 - Optimization
 - Evaluation



Training Workflow of AlphaZero

- **AlphaGo Zero**
 - Generation
 - Optimization
 - Evaluation
- **AlphaZero**
 - Generation
 - Optimization



Training Workflow of AlphaZero

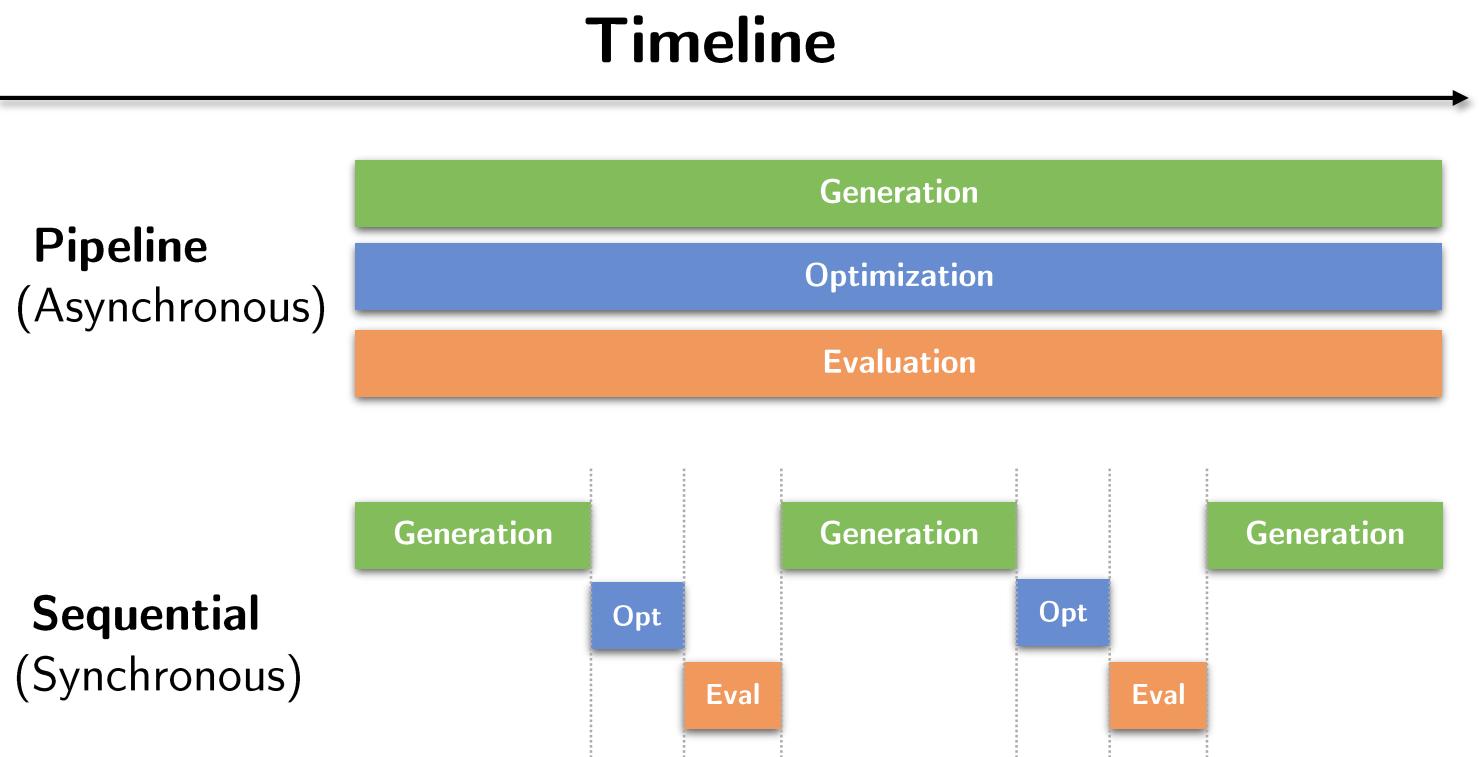
- **AlphaGo Zero**

- Generation
- Optimization
- Evaluation

- **AlphaZero**

- Generation
- Optimization

- **Pipeline vs. Sequential**



Related Work

- LeelaZero
 - Open source
 - Volunteer computing
- ELF OpenGo
 - Based on ELF^[1]
 - Facebook AI Research (FAIR)
 - Open source
 - Focus on (Deep) Reinforcement Learning problems
- CrazyZero
 - Claimed to support the following two-player perfect information games
 - Go, Gomoku, Renju, Shogi
 - Works well
 - Not open source

[1] Tian, Yuandong, et al. "Elf: An extensive, lightweight and flexible research platform for real-time strategy games." NIPS. 2017.

Design and Implementation

Architecture

Training Workflow

Control Side and Inference Side

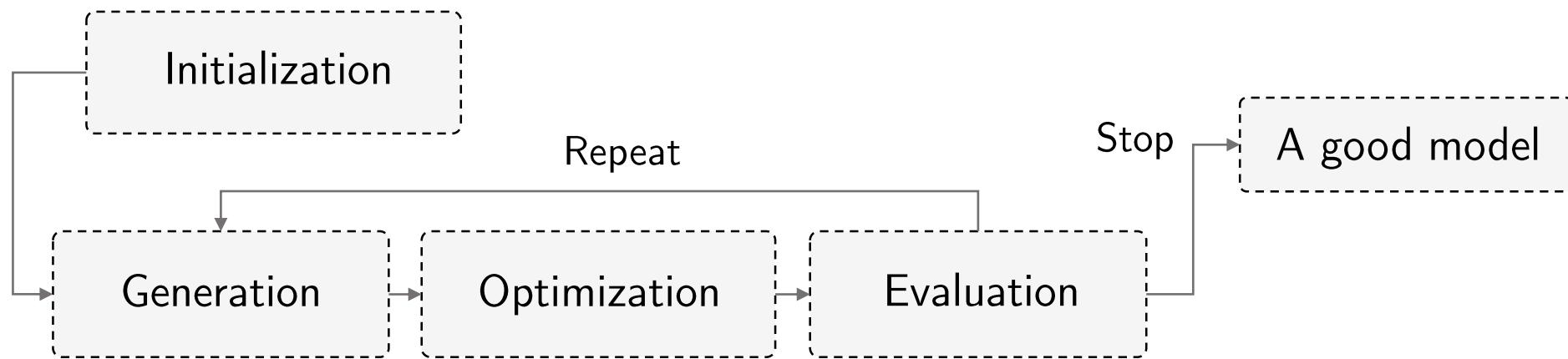
Case Study

Architecture

- Three part of the architecture
 - Master
 - Manage the entire training process
 - Handle the communication with Worker and Optimizer **via job and report**
 - Worker
 - Handle two tasks
 - (Self-Player) generate game records for training via self-play
 - (Evaluator) evaluate different agents' strength
 - Optimizer
 - Handle the update of neural network by using deep learning techniques

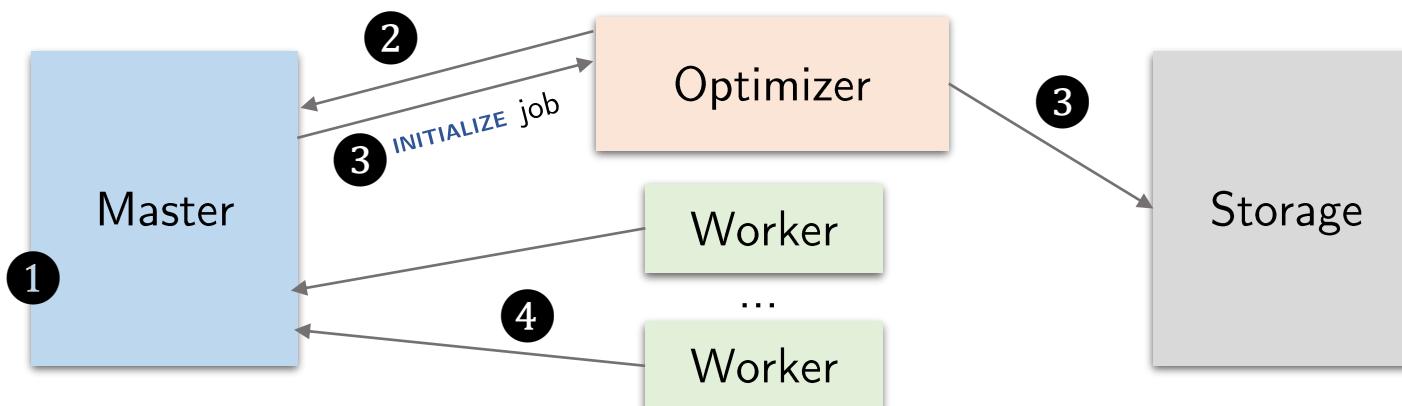
Training Workflow

- We will talk about **Sequential Training** first, and **Semi-Asynchronous Training** will be explained later
- Basic Sequential Training Workflow
 1. Initialization
 2. Generation
 3. Optimization
 4. Evaluation (optional)
 5. Repeat 2 - 4 until the network converges



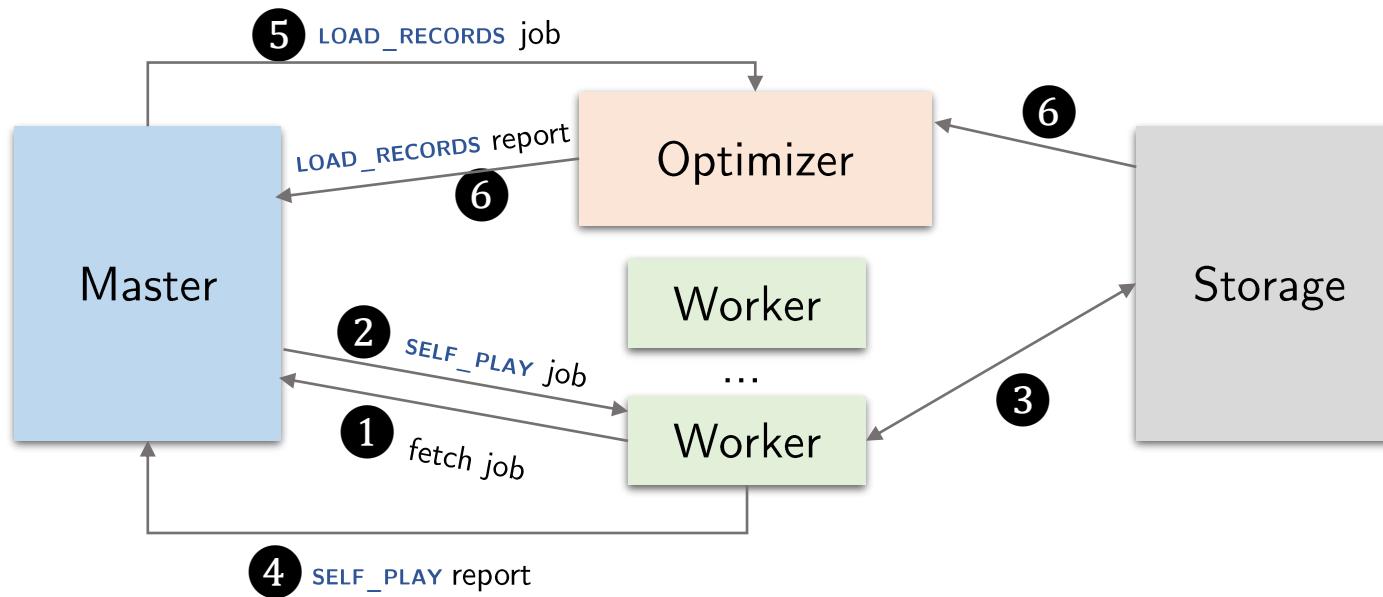
Initialization

1. Start Master
2. Start Optimizer, then it will establish connection with Master
3. Master sends the **INITIALIZE job** to Optimizer,
then Optimizer will generate a random weight and save it on Storage
4. Start Worker, then Worker will establish connection with Master



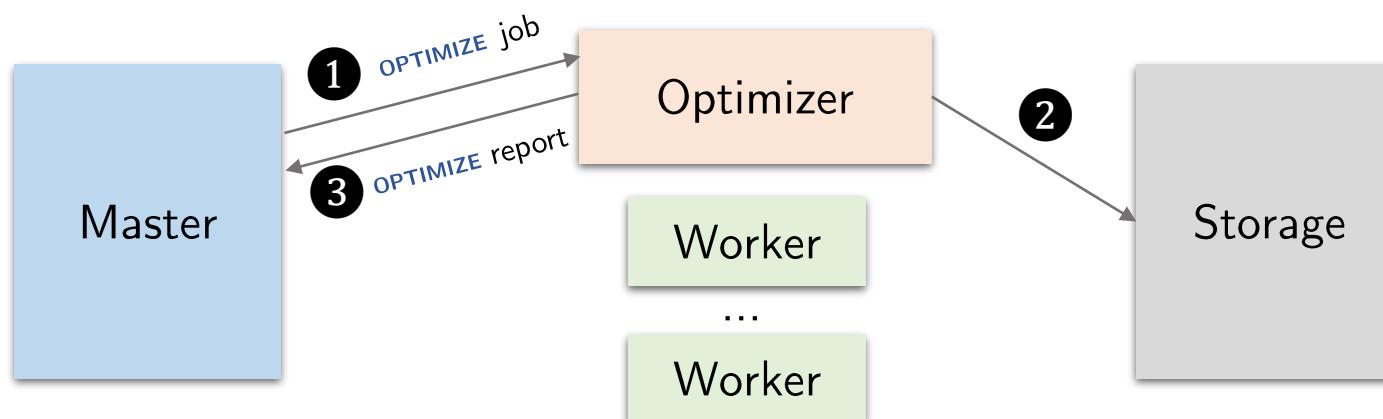
Generation

1. Worker(s) fetch job from Master
2. Master sends **SELF_PLAY job** to Worker
3. Worker will generate games via self-play, then save records on Storage
4. Worker sends **SELF_PLAY report** back to Master once the job finished
5. Master sends **LOAD_RECORDS job** to Optimizer
6. Optimizers will load records form Storage and convert them to input features, then send **LOAD_RECORDS report** back to Master once the job finished



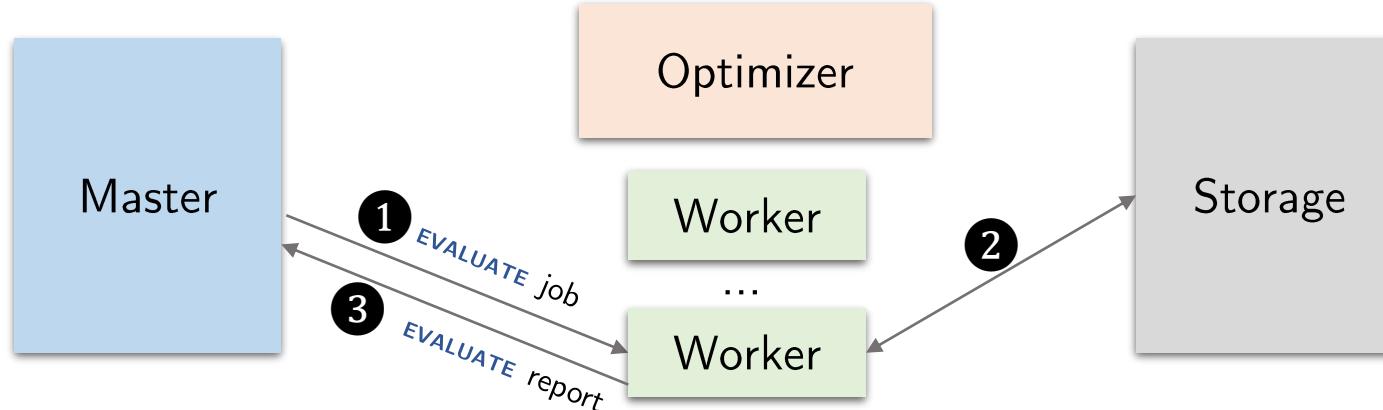
Optimization

1. Master sends **OPTIMIZE job** to Optimizer
2. Optimizer updates Neural Network via deep learning framework, then save them on Storage
3. Optimizer sends **OPTIMIZE report** back to Master once the job finished

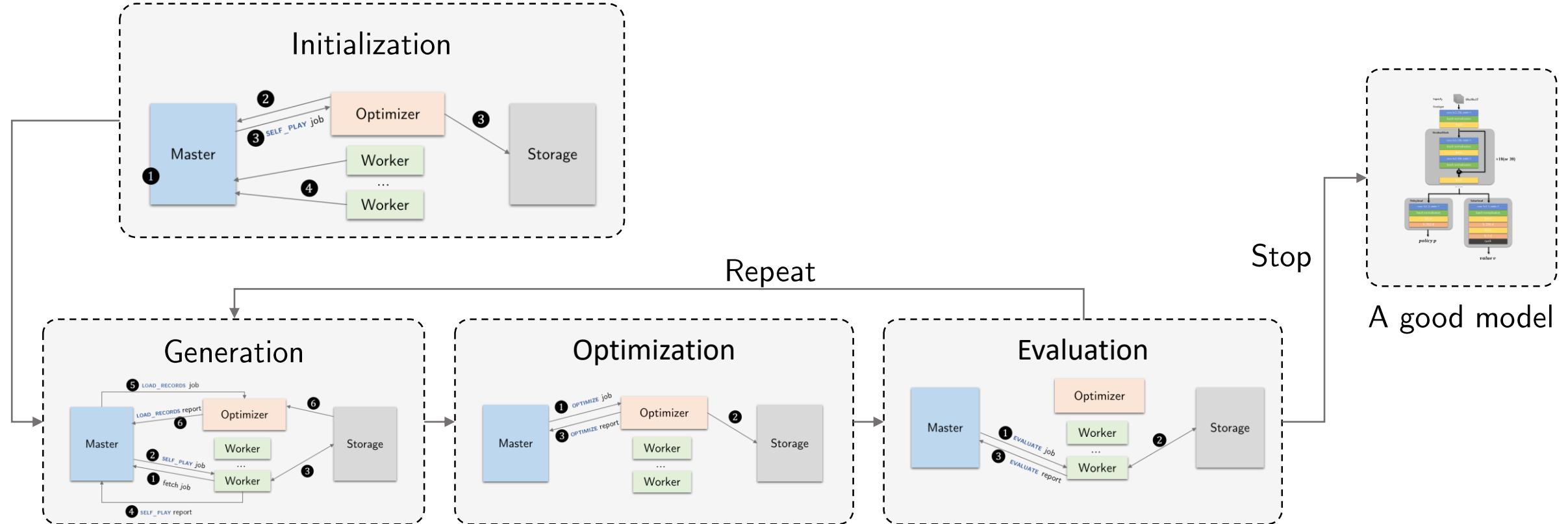


Evaluation

1. Master sends **EVALUATE job** to Worker
2. Worker evaluates weights' strength between f_{θ_*} and f_{θ_i} load from Storage
(update best network $f_{\theta_*} = f_{\theta_i}$ if f_{θ_i} wins > 55% to f_{θ_*})
3. Worker sends **EVALUATE report** back to Master once the job finished



Repeat 2 - 4 until the network converges

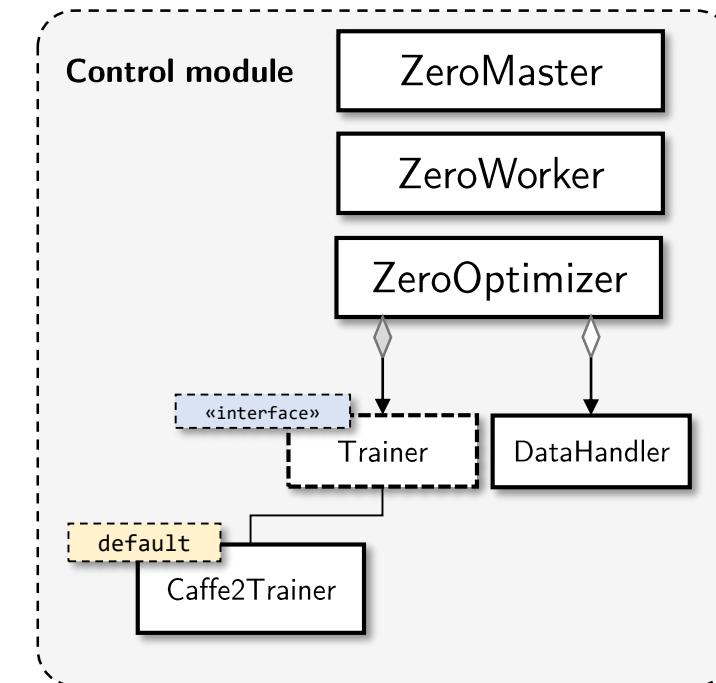


Control Module and Inference Module

- The framework is divided into two modules

- Control Module**

- Manage whole training process
- Manage deep learning training framework
- Focus on scalable
- Python implementation**

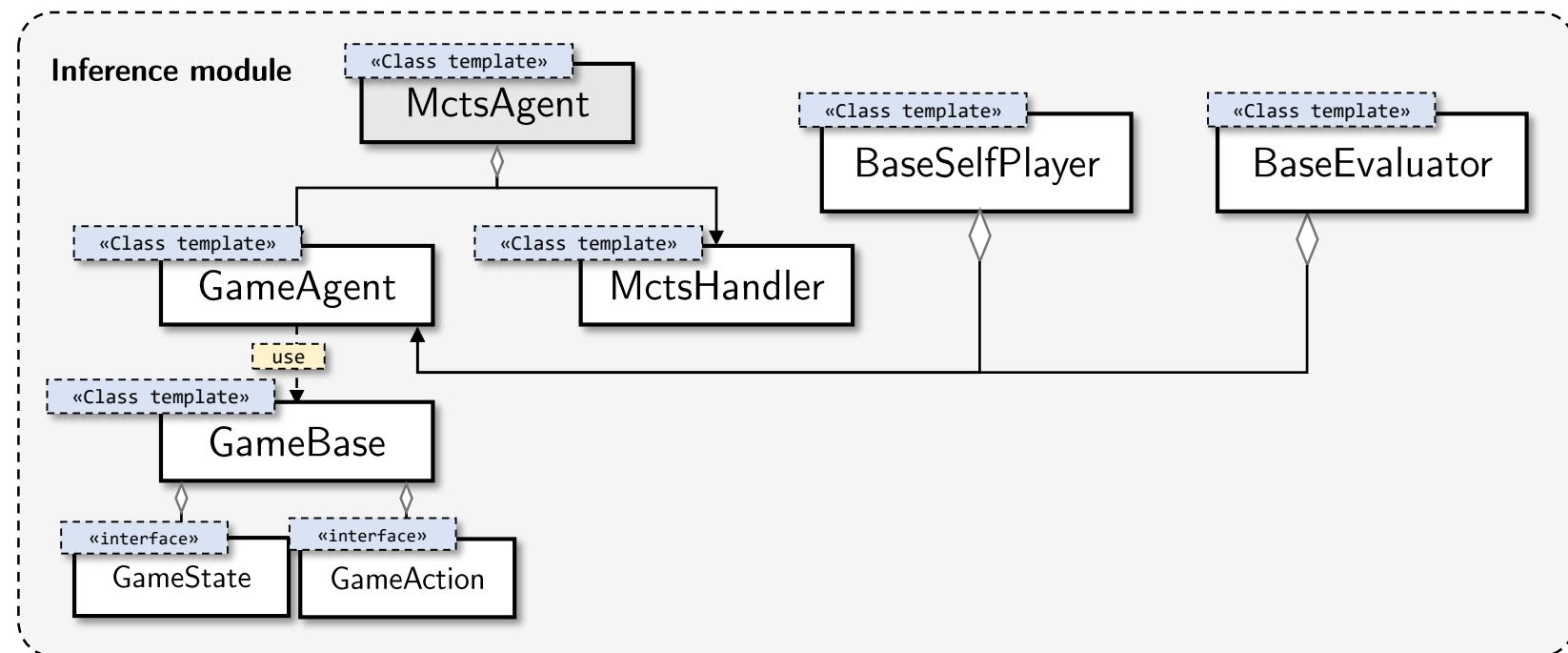


Control Module and Inference Module

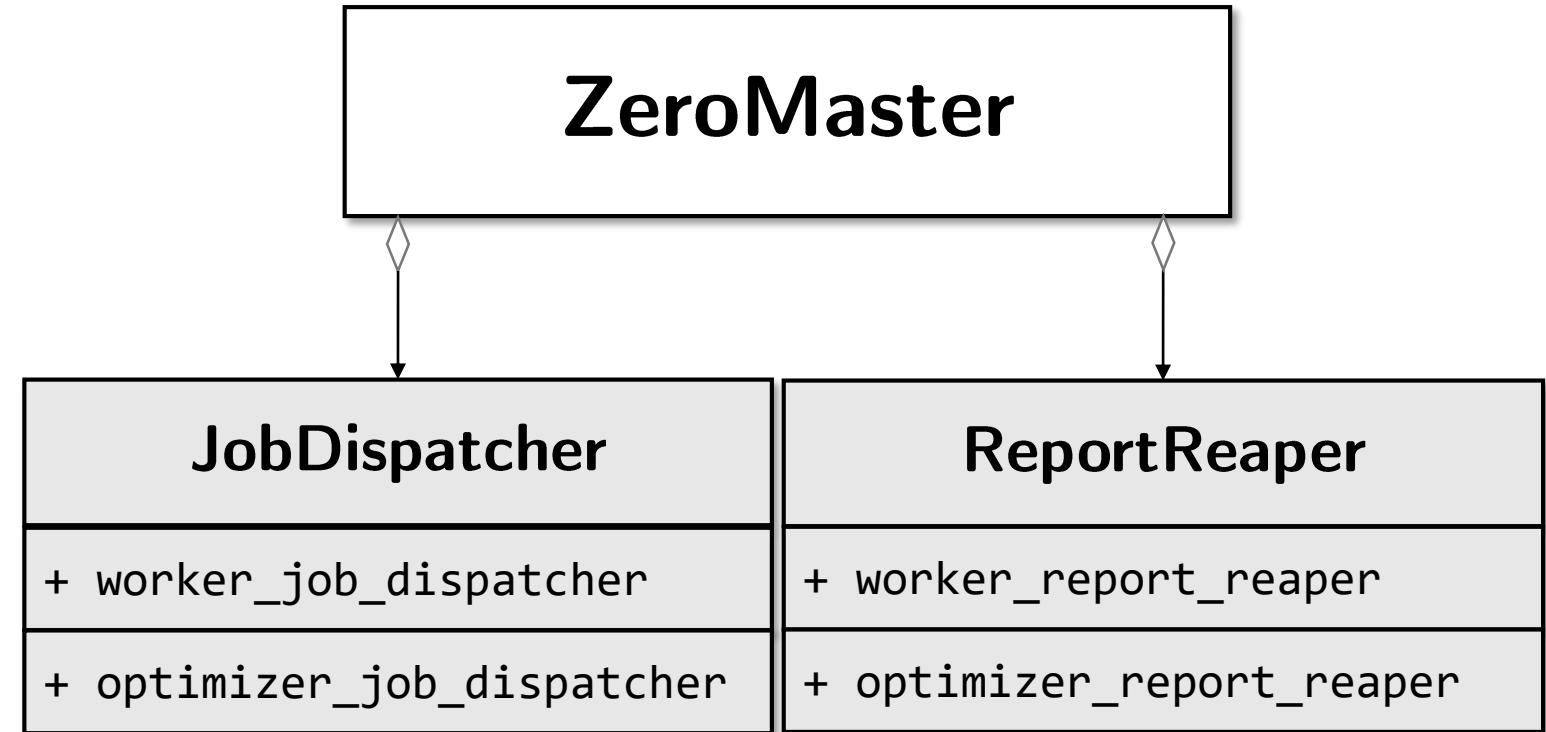
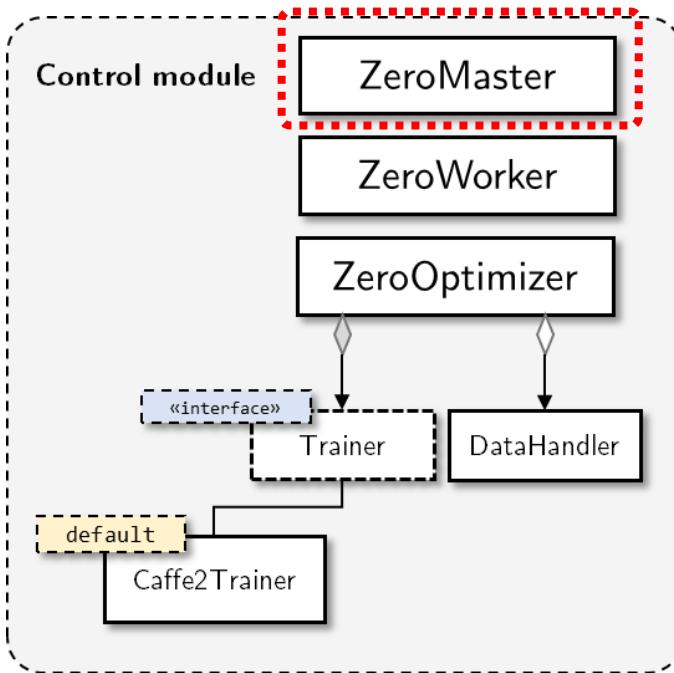
- The framework is divided into two modules

- Inference Module**

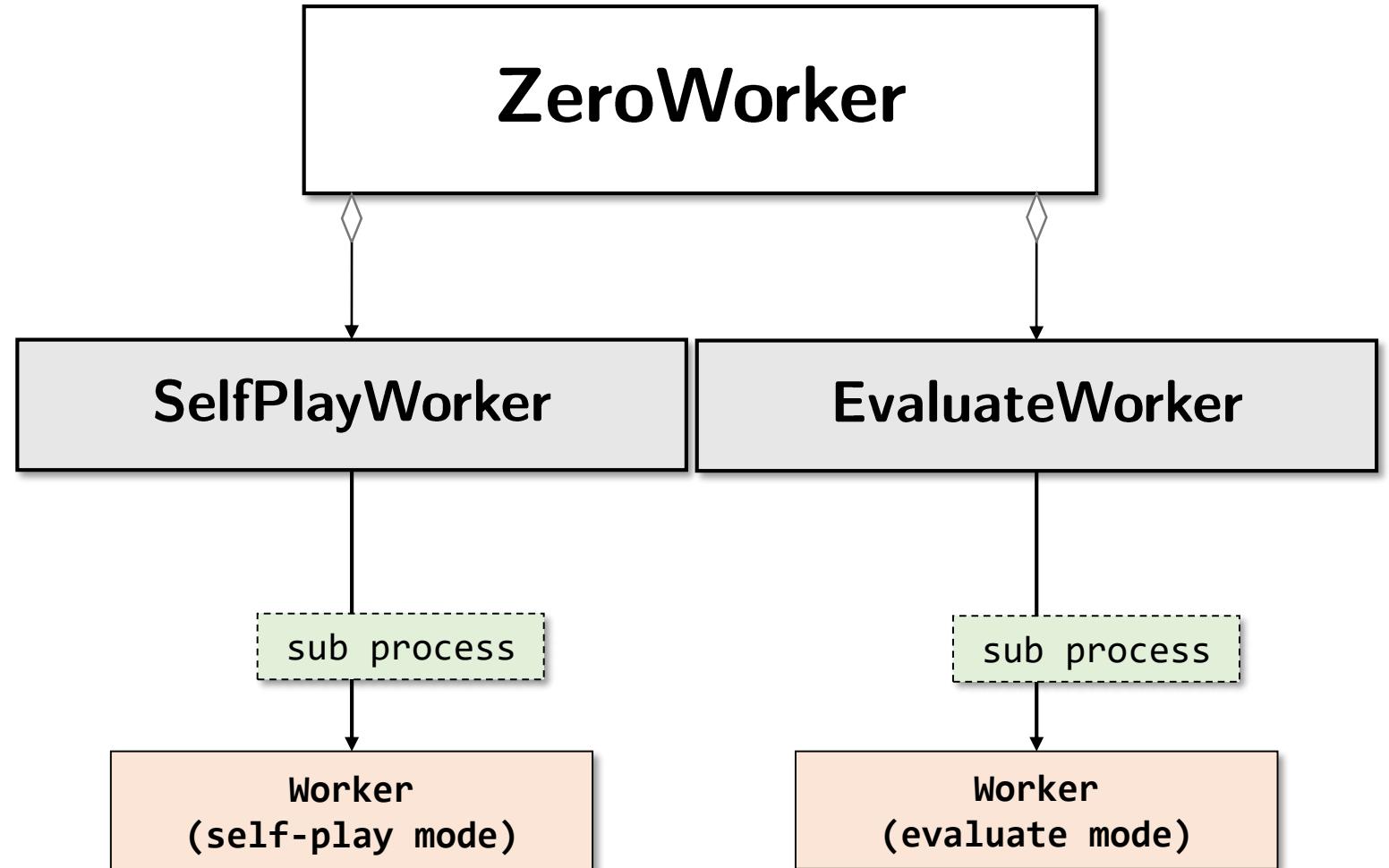
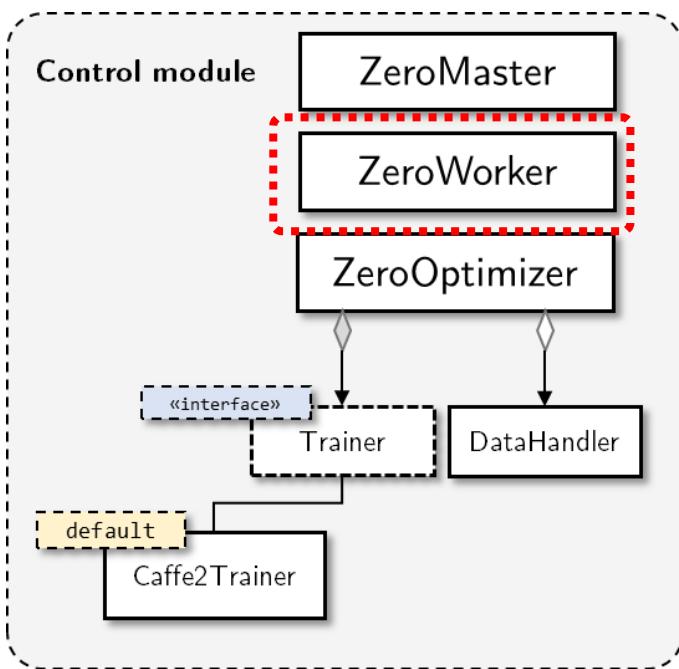
- Focus on playing
 - Self-play
 - Competition
- Focus on performance
- C++ implementation



Control Module - ZeroMaster

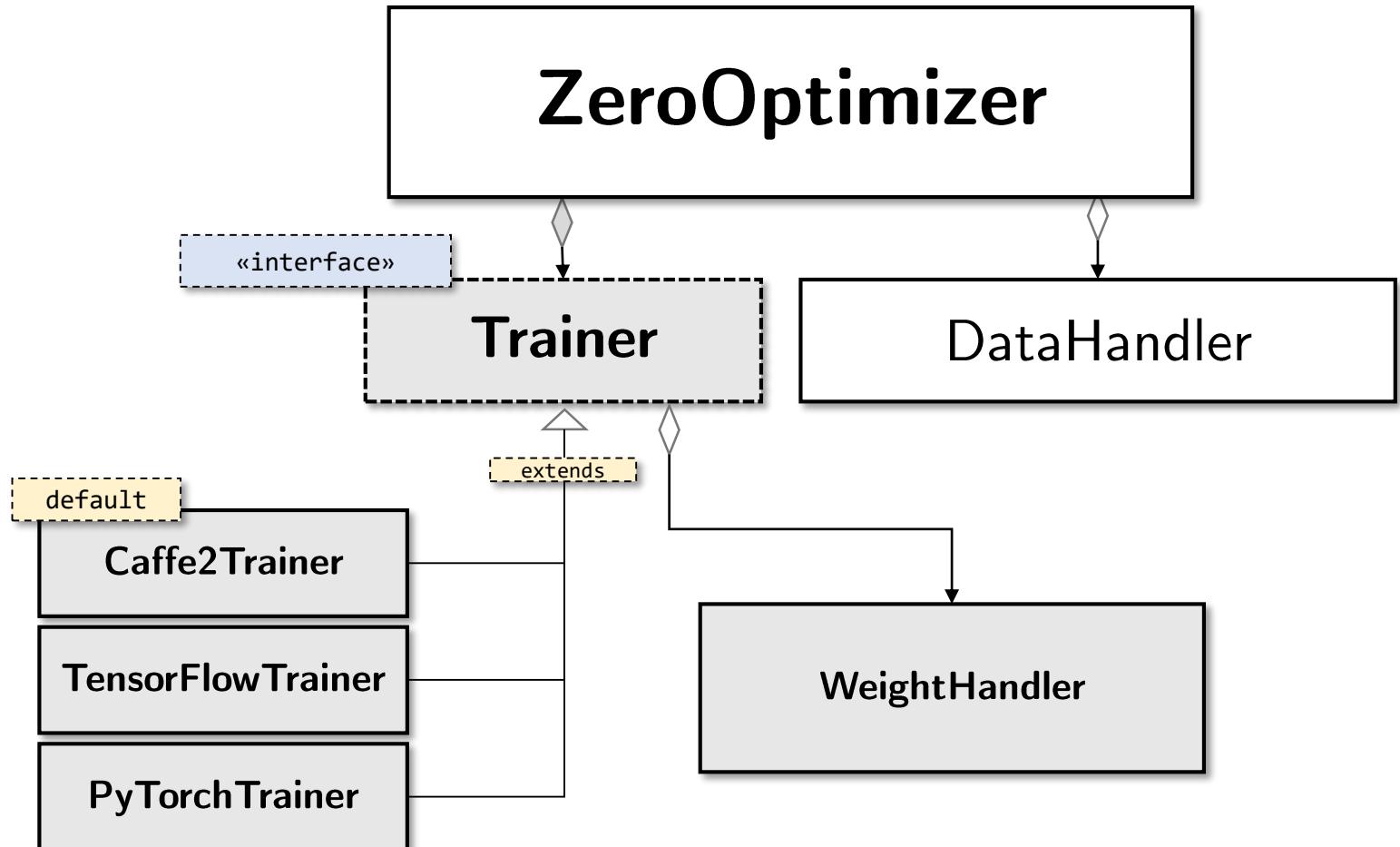
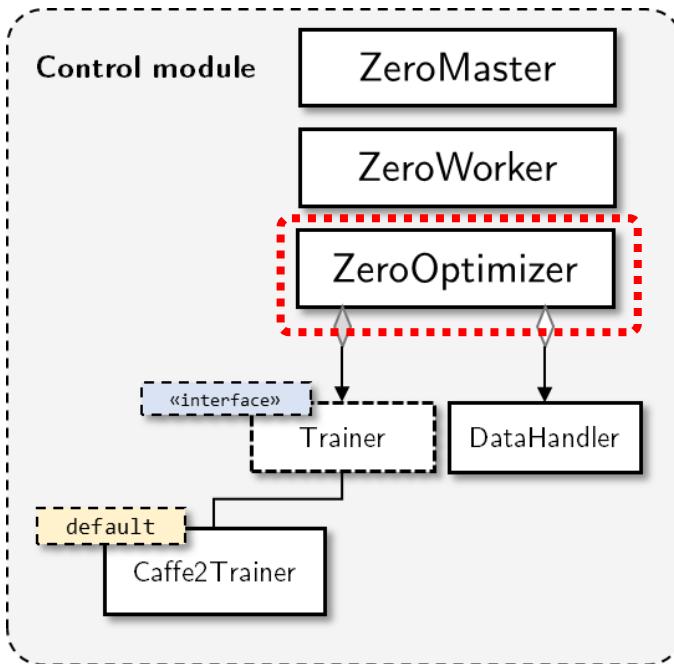


Control Module - ZeroWorker



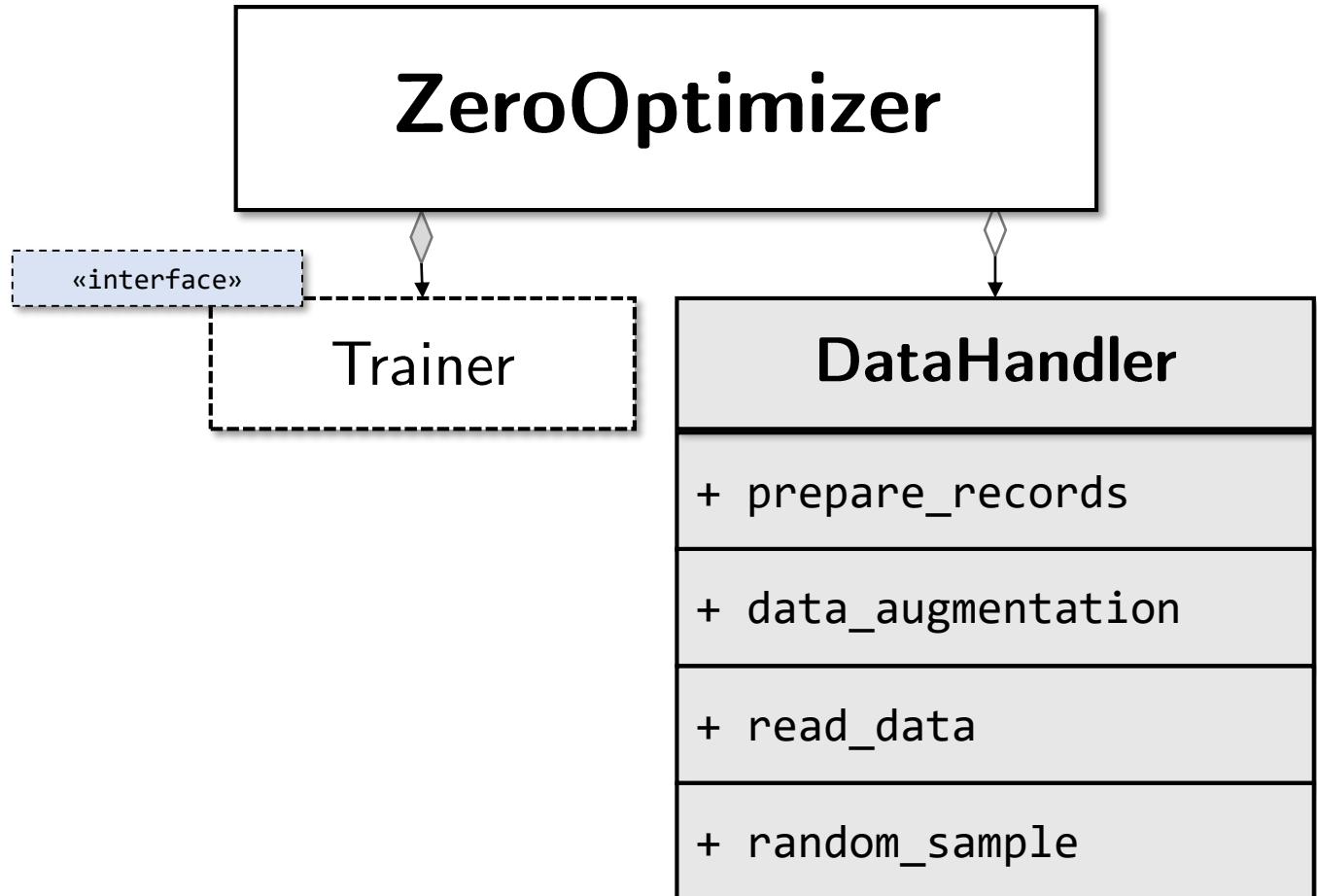
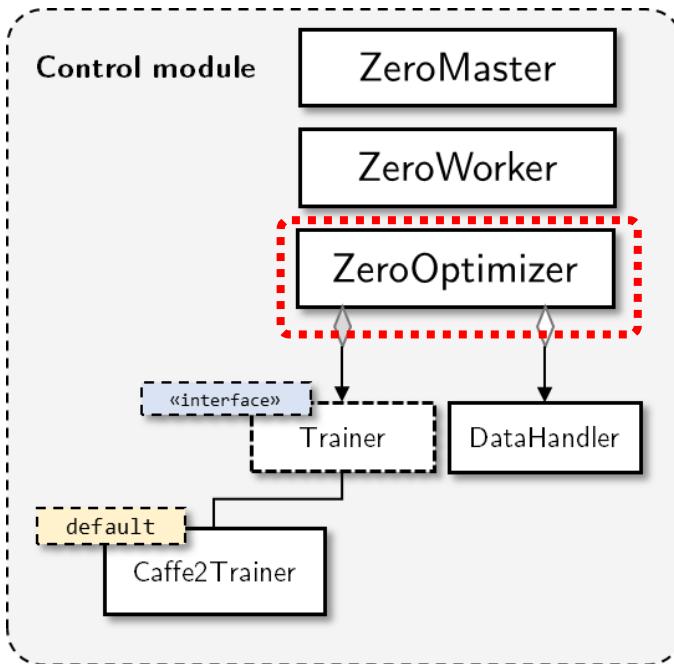
Control Module - ZeroOptimizer

- Trainer

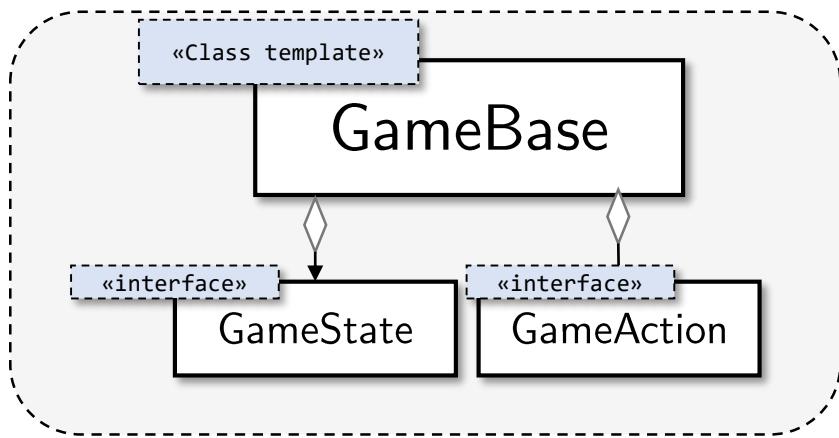


Control Module - ZeroOptimizer

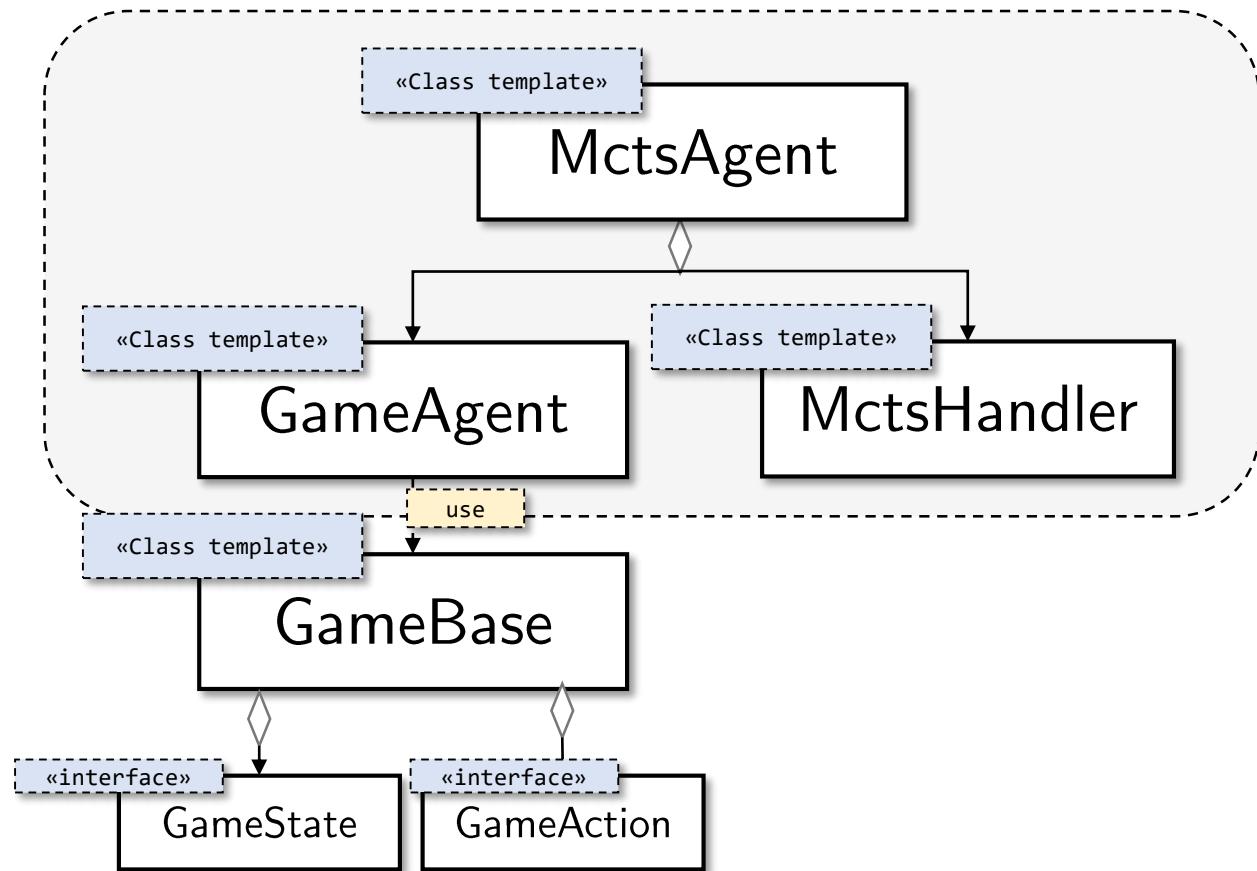
- DataHandler



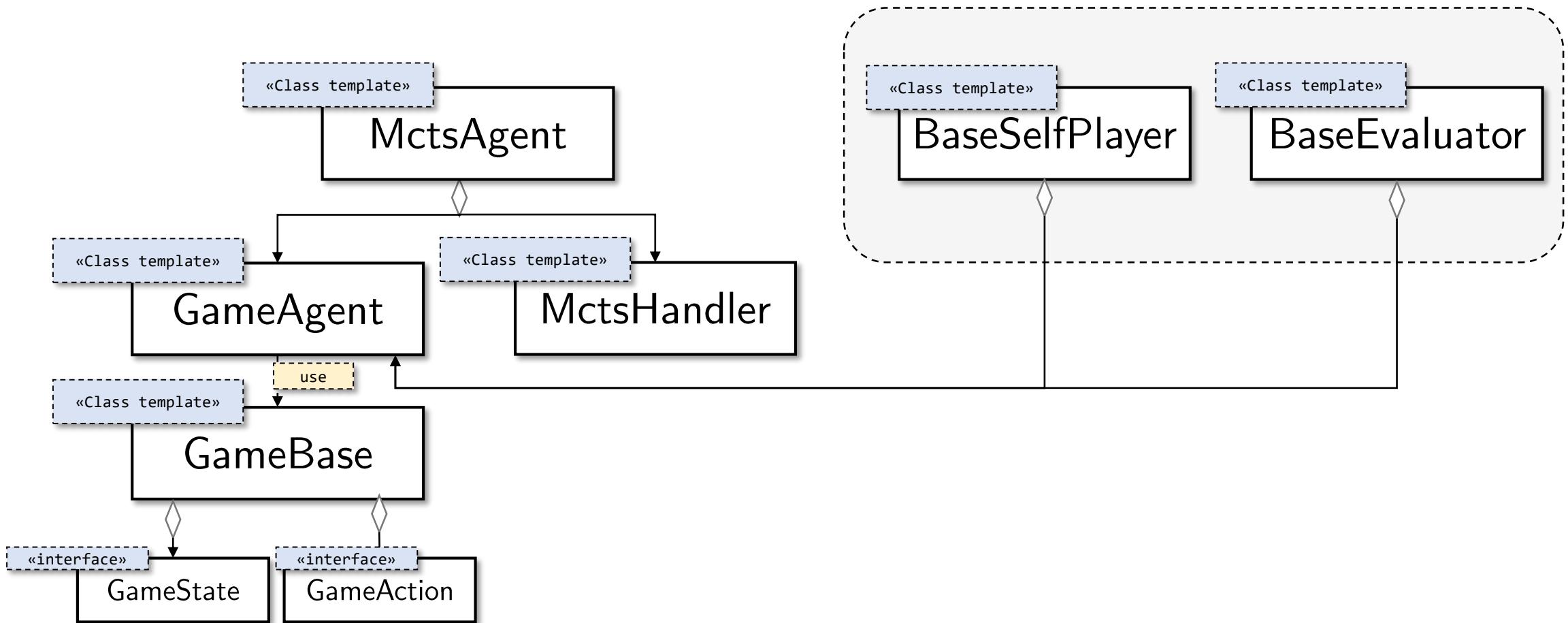
Inference Module - GameBase Part



Inference Module - GameAgent Part

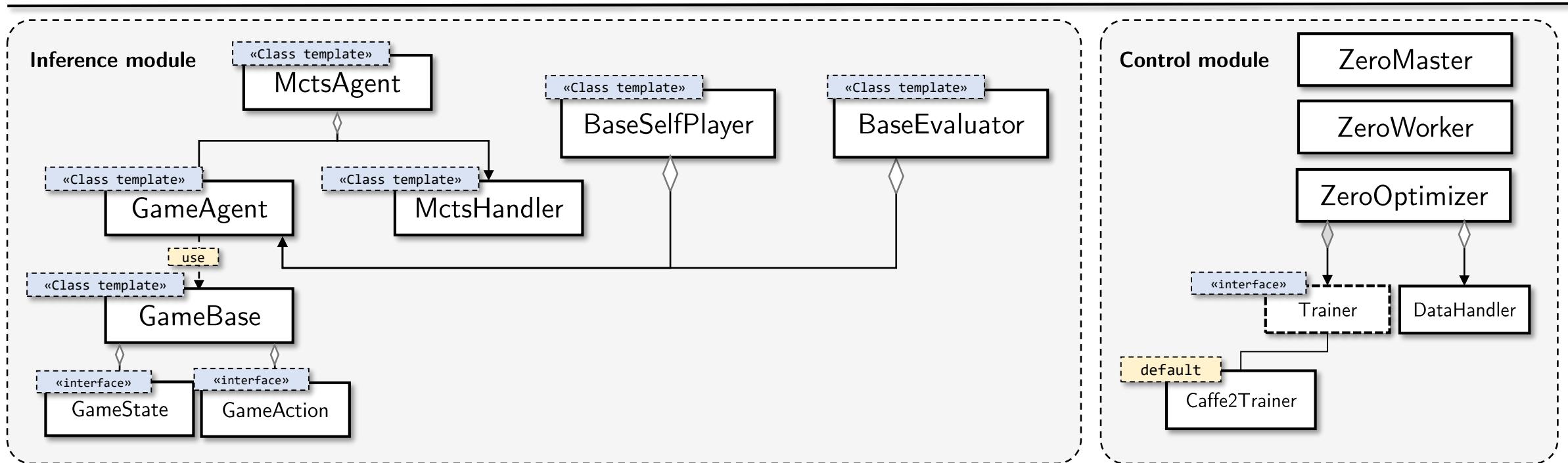


Inference Module - Self-Play/Evaluate Part

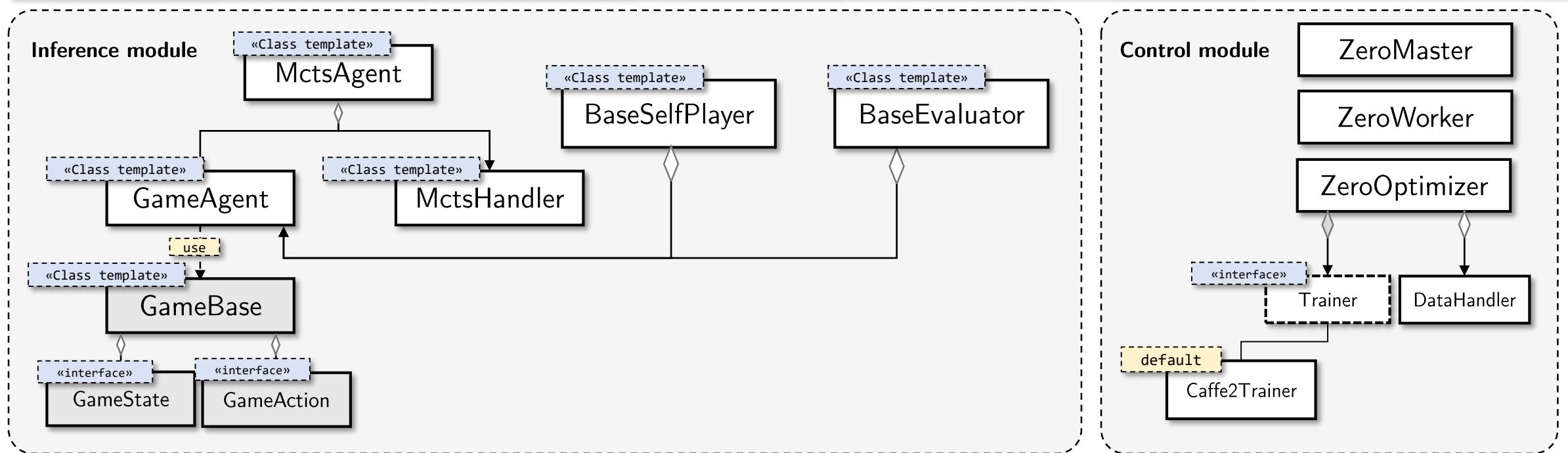


Case Study - Basic Usage

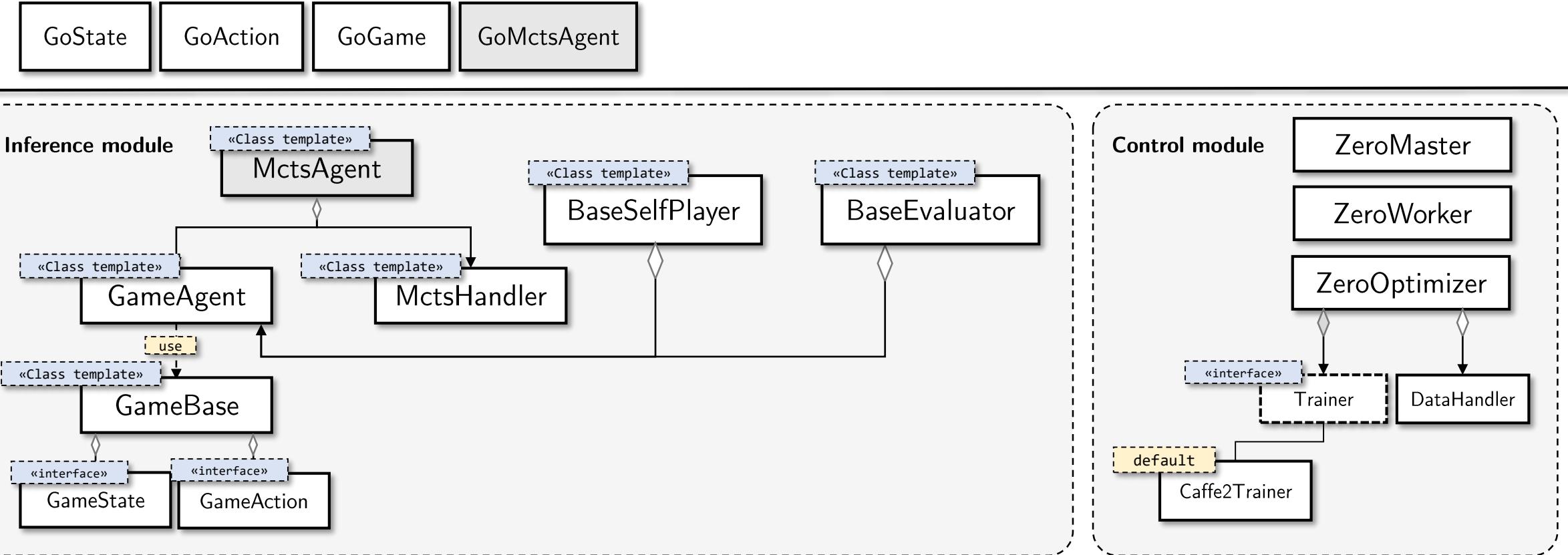
- This subsection will show you how to develop an application based on our framework
 - We use **Go** as an example



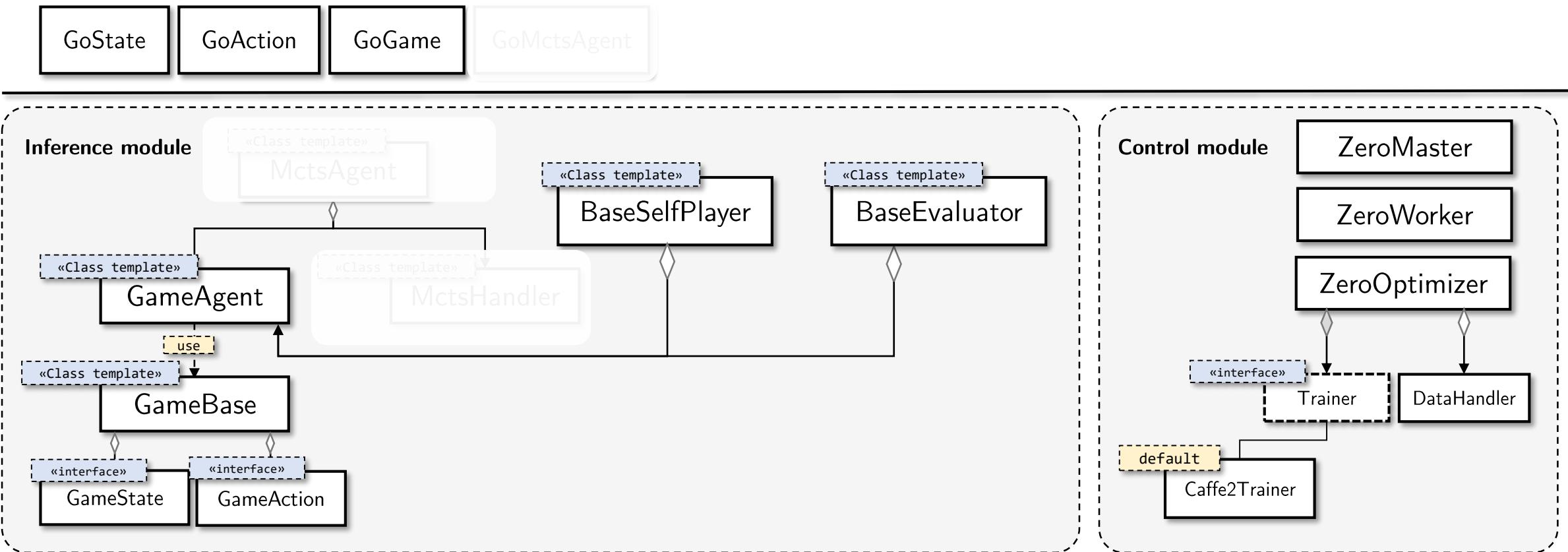
Case Study - Basic Usage



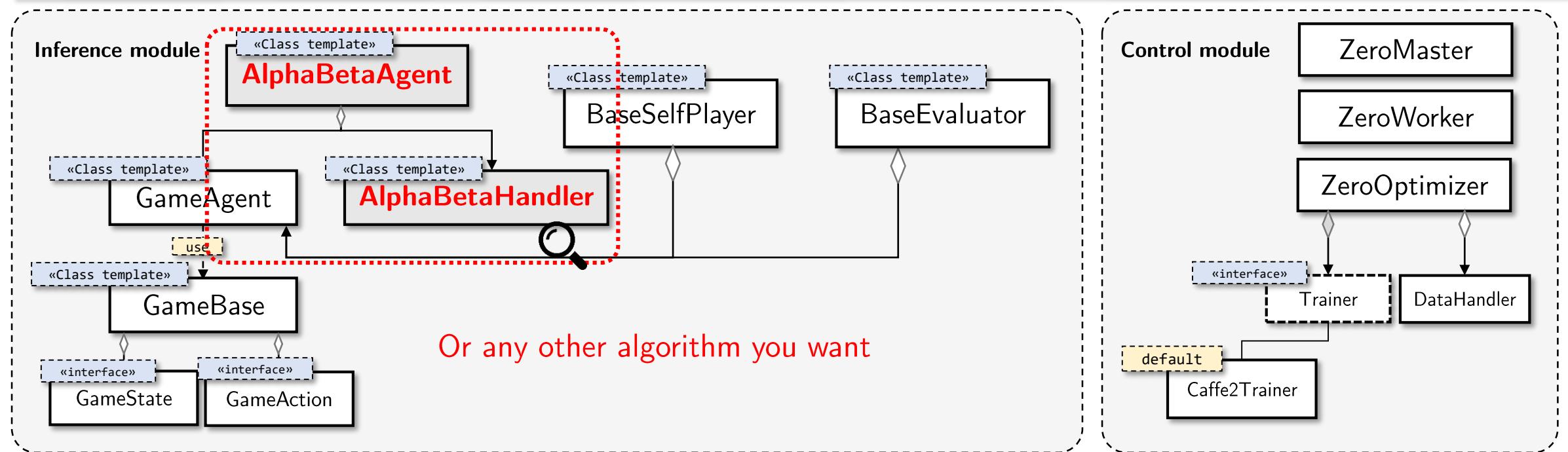
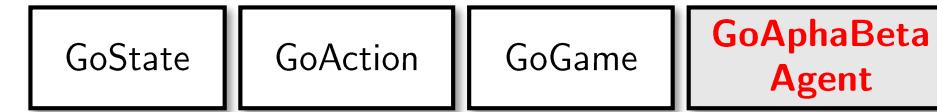
Case Study - Basic Usage



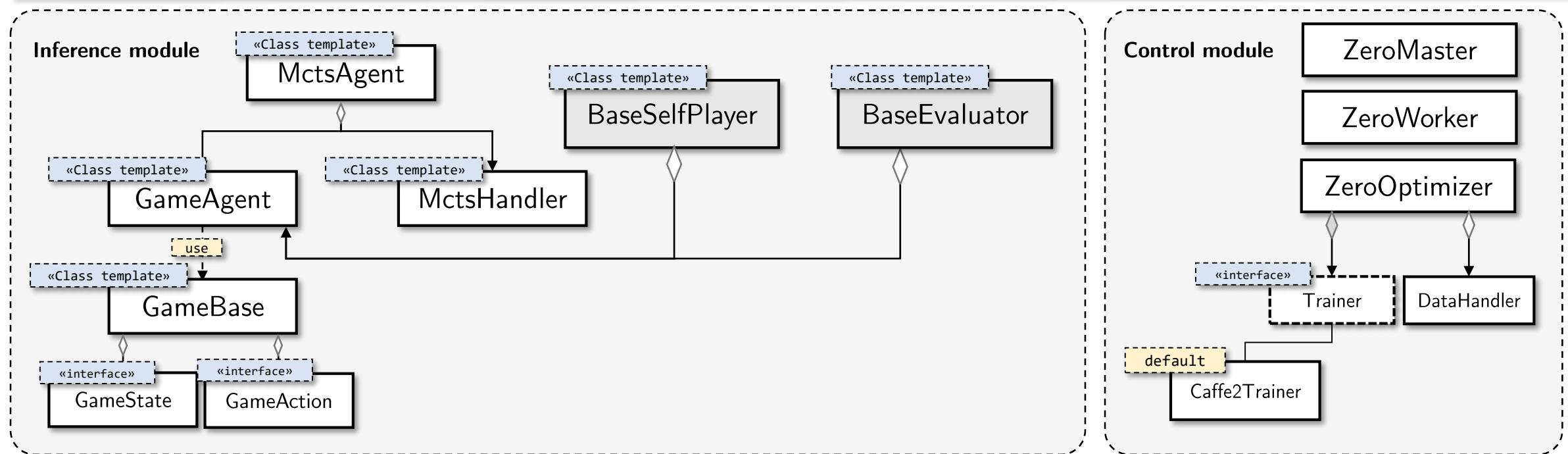
Case Study - Basic Usage



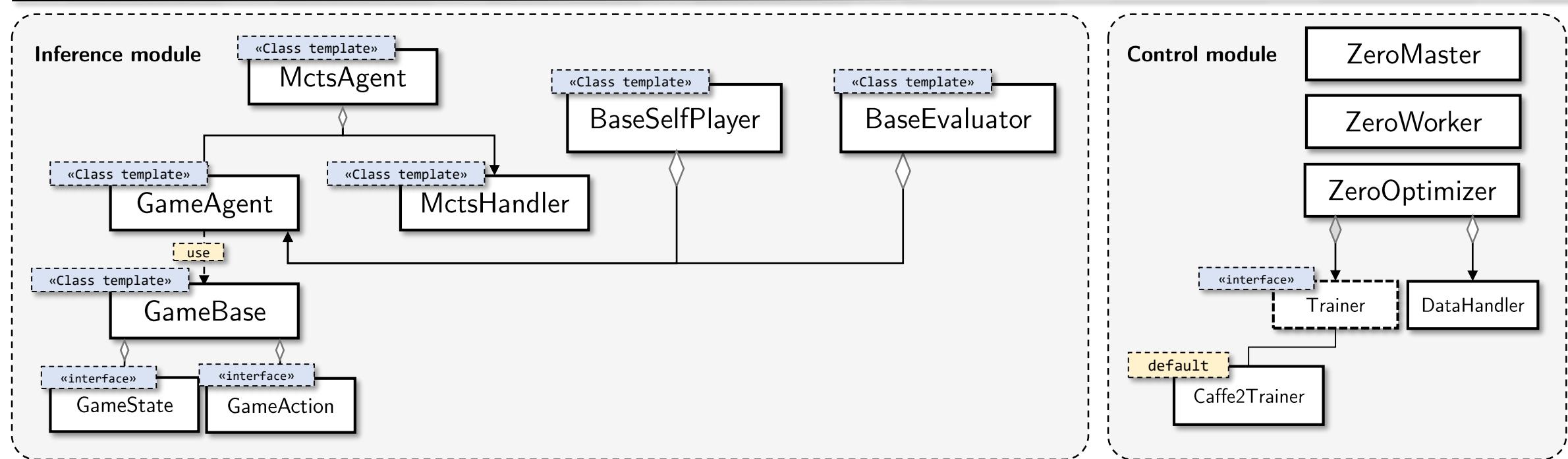
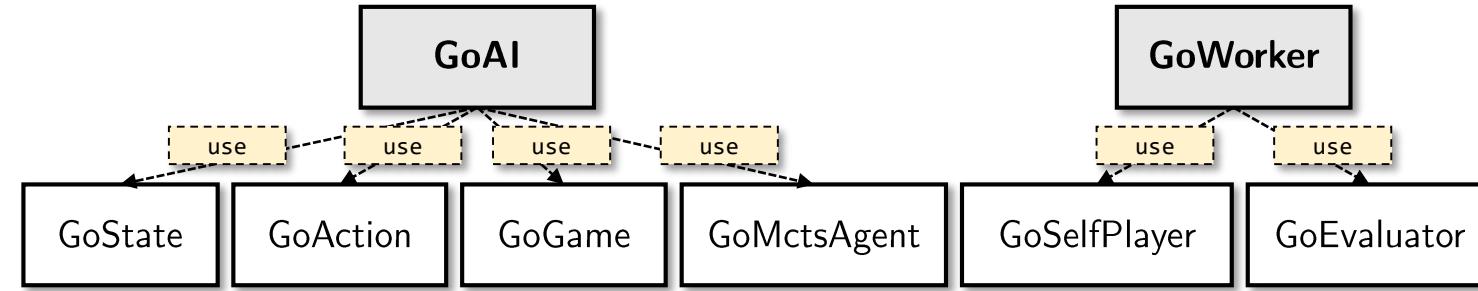
Case Study - Basic Usage



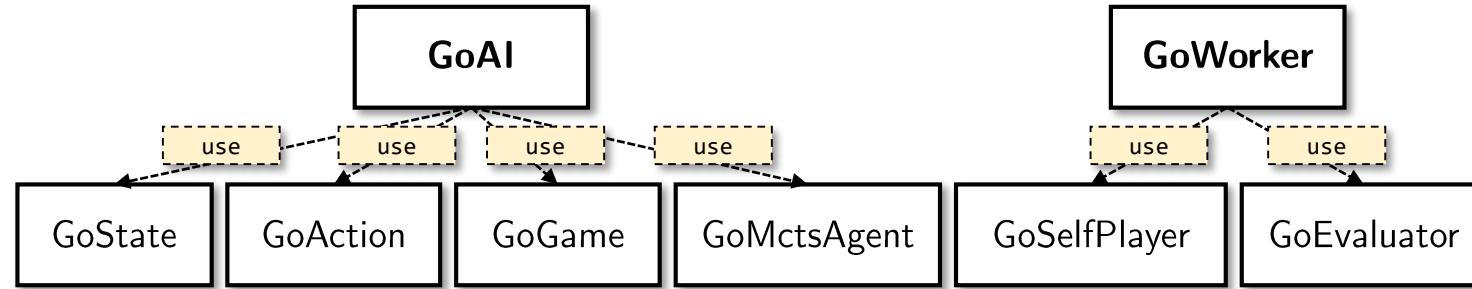
Case Study - Basic Usage



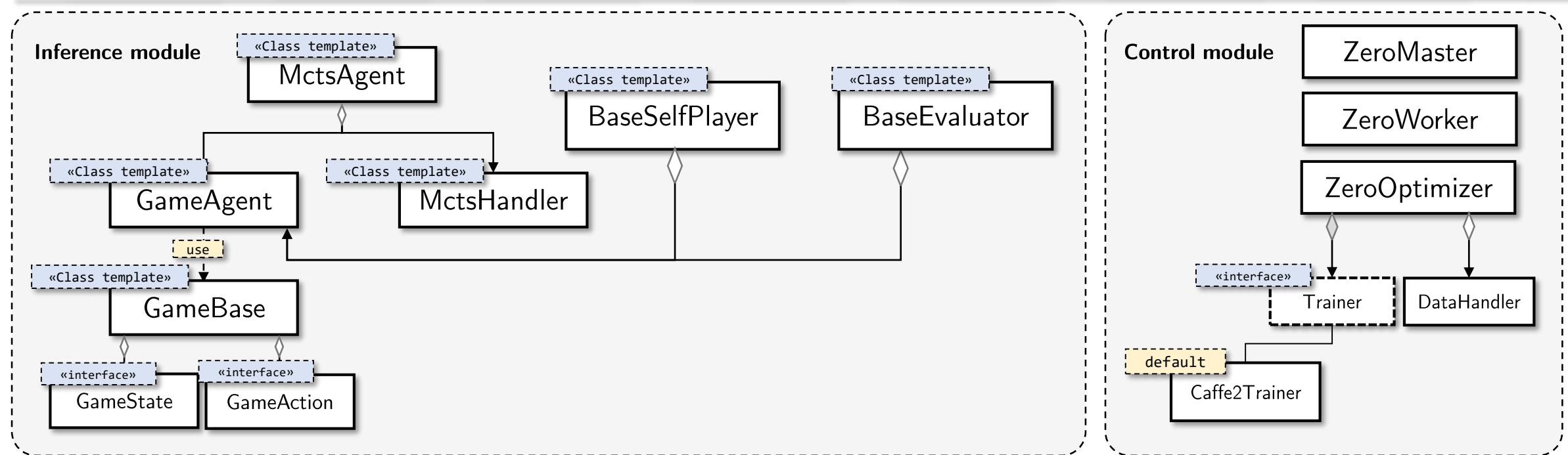
Case Study - Basic Usage



Case Study - Basic Usage

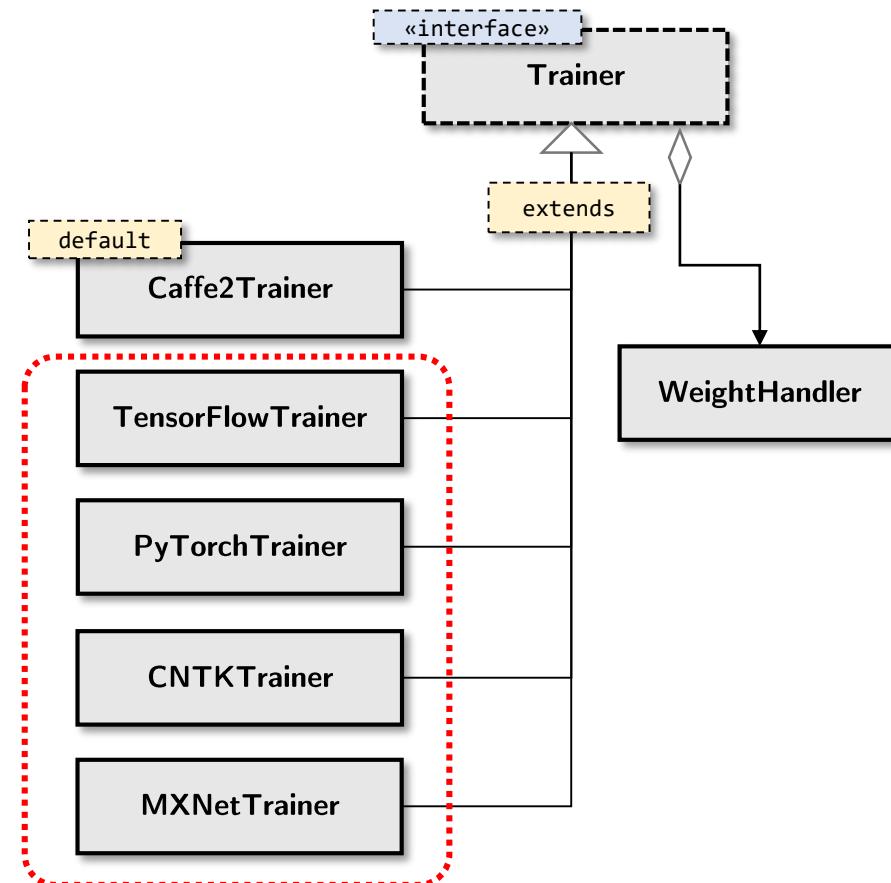
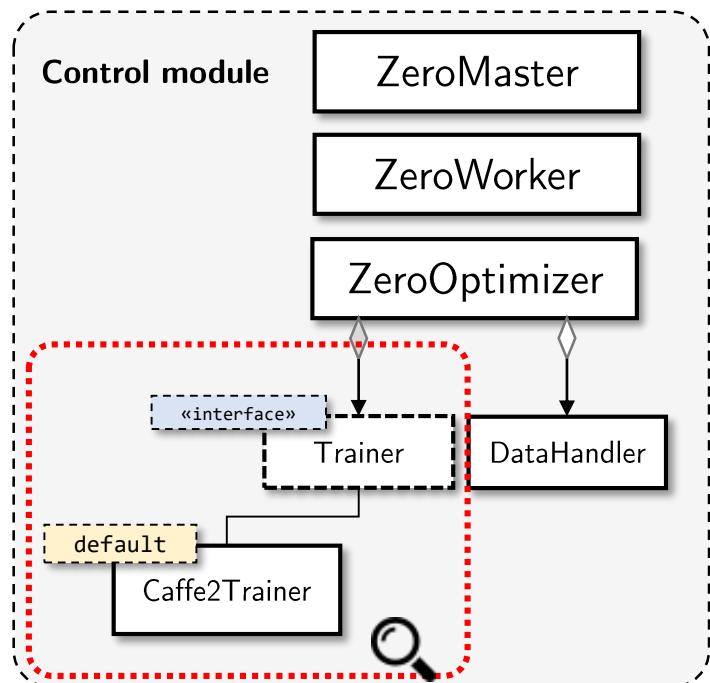


No need to extra implementation
Change the **configuration** if necessary



Case Study - Extend to other DL framework

- Extend the Trainer Class
 - Implement the details by using any other DL framework you want
- Add a function in WeightHandler
 - Convert other weight's format to Caffe2's format



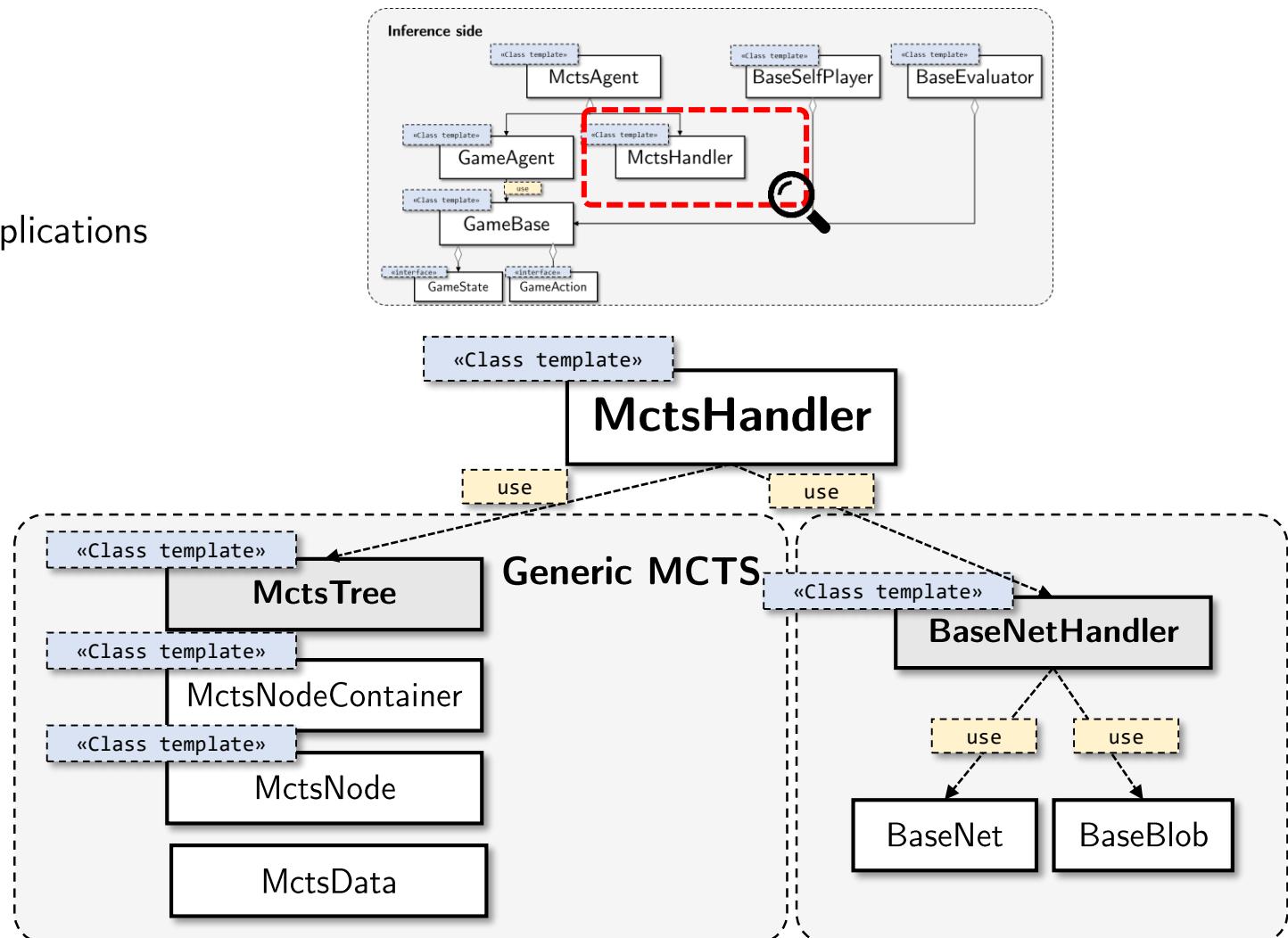
Implementation Issues

Batch-Forward MCTS

Semi-Asynchronous Training

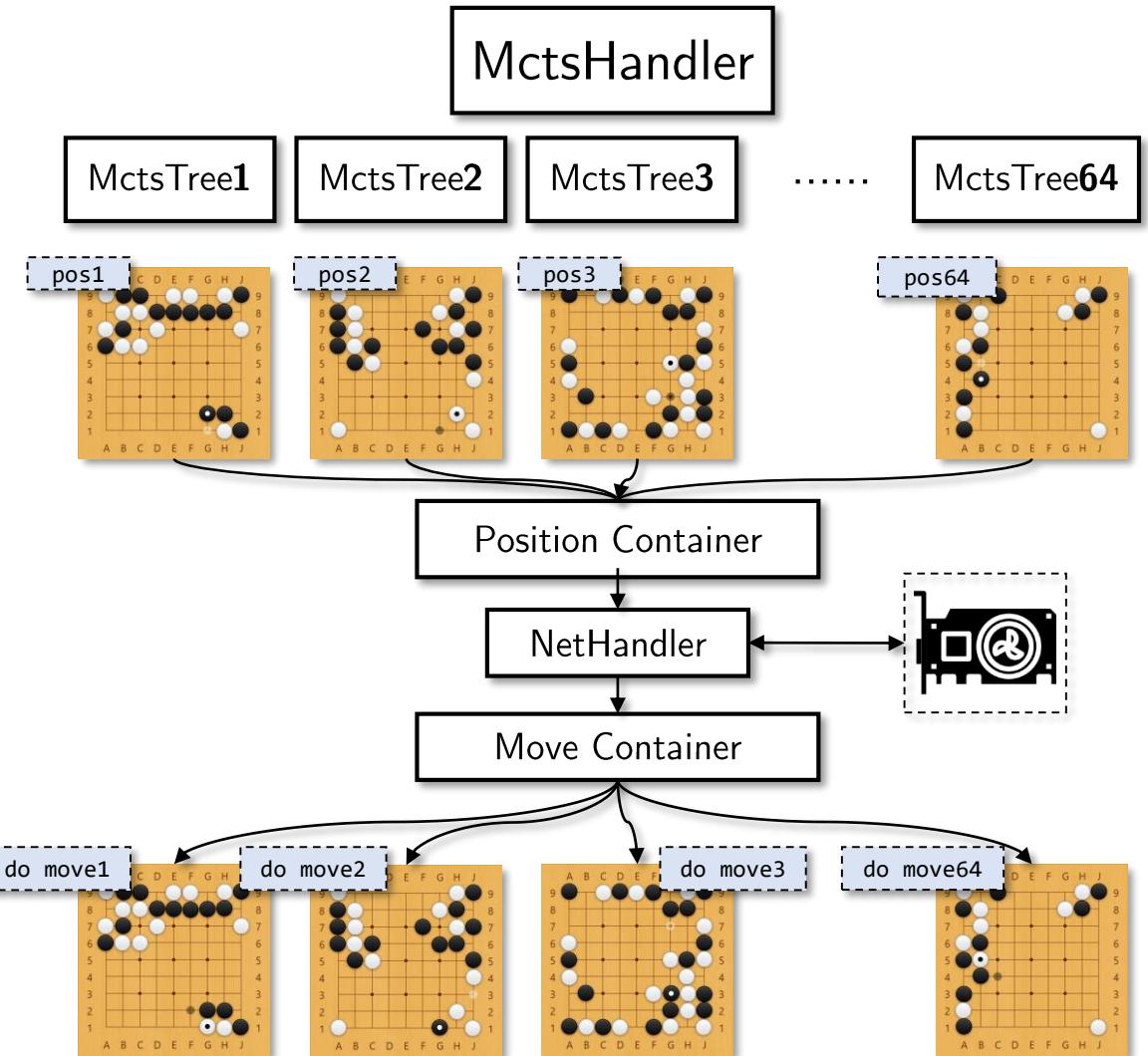
Batch-Forward MCTS

- Generic MCTS
 - An MCTS Class template
 - Support different games
 - Can be easily separated for other applications
- BaseNetHandler
 - An DNN Class template
 - Handle neural netowrk evaluation



Batch-Forward MCTS

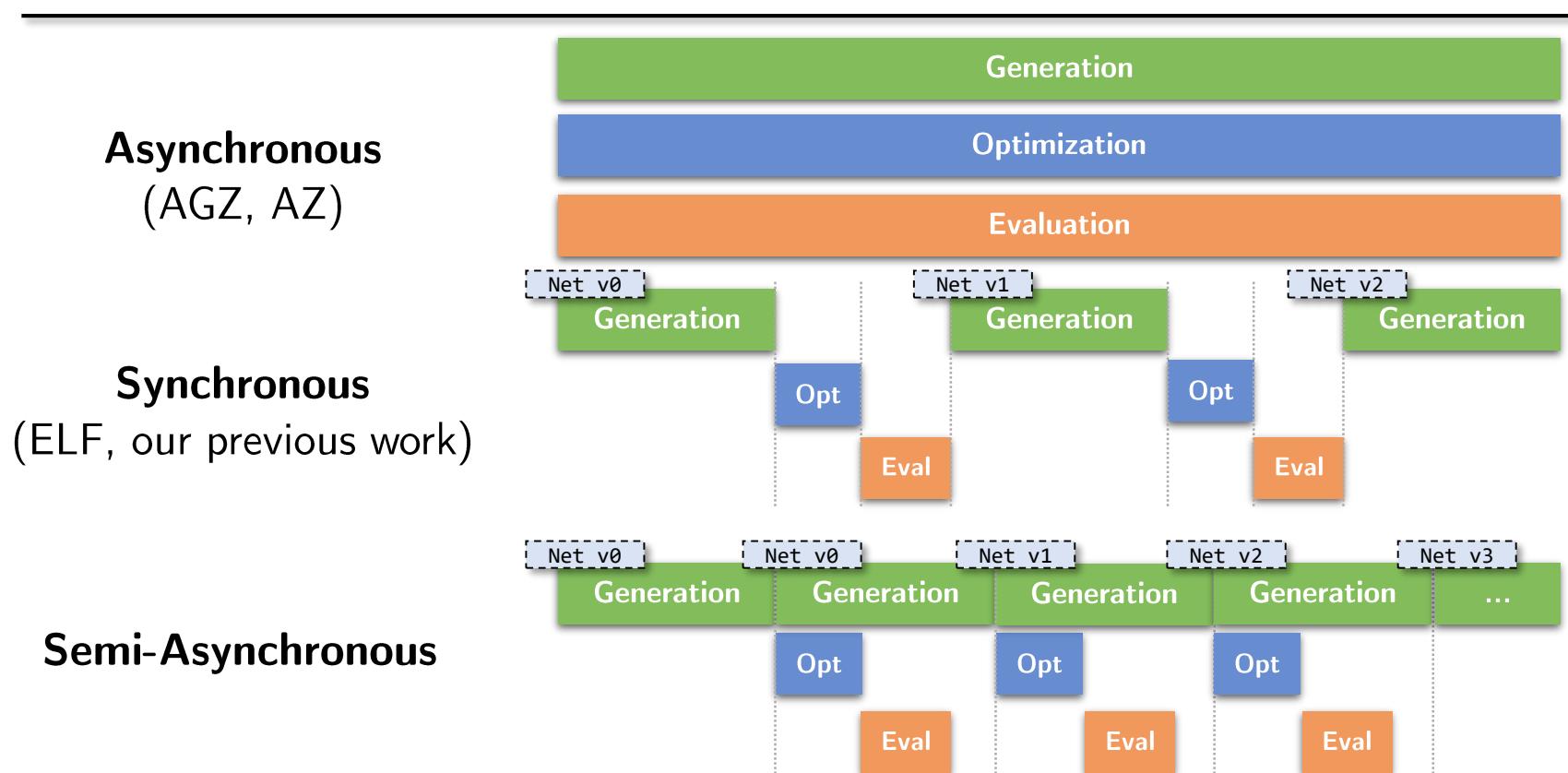
- Generic MCTS
- BaseNetHandler
- Batch-Forward MCTS
 - Maintain a container consist of many trees
 - E.g., **64**, which mean we play 64 games together
 - Do MCTS simulation
 - Collect all positions of the trees
 - Send to GPU evaluation
 - Get Moves
 - Do move in each tree



Semi-Asynchronous Training

- **AZ** and **AGZ** used Asynchronous Training
- **ELF** and **Our previous work** used Synchronous Training
- We proposed a new training method **Semi-Asynchronous Training**

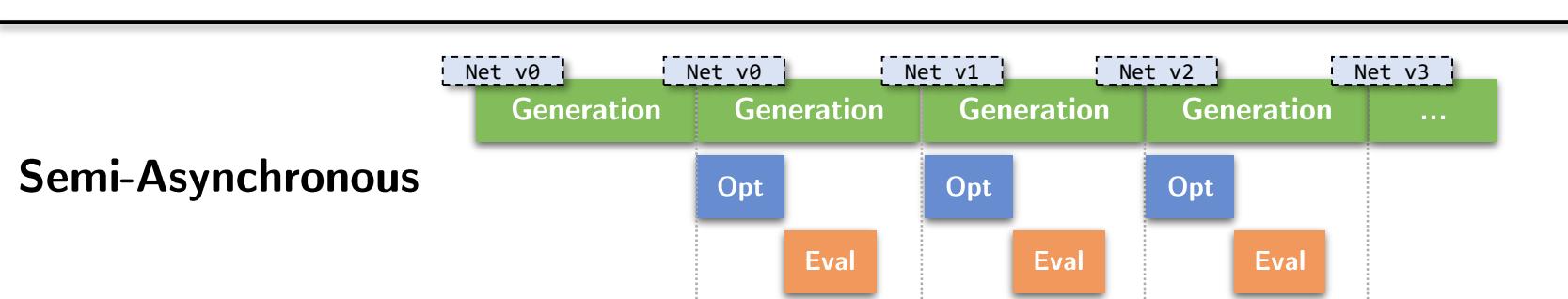
Timeline



Semi-Asynchronous Training

- **AZ** and **AGZ** used Asynchronous Training
- **ELF** and **Our previous work** used Synchronous Training
- We proposed a new training method **Semi-Asynchronous Training**
 - Increase the training speed than Synchronous Training
 - Work well when there are only a few resources
 - E.g., 16 GPUs, 32 GPUs
 - Consider Asynchronous Training to get better performance if we have enough resources
 - E.g., 2000 GPUs, 5000 GPUs

Timeline



Experiments

Environment

Baseline

Depth and Width of Networks

Sample Rate of Positions

Larger Network

Strength of a few simulations

Environment

- Most of our experiments Based on NoGo
 - NoGo is short for No Capture Go
 - A variation game of Go
 - 9 x 9 board
 - We use HaHaNoGo as benchmark
 - TAAI 2016 Game Tournament **champion**
 - The state-of-the-art NoGo program
 - 500 games per experiment (250 black, 250 white)
- Environment
 - GPU:
 - **33** x NVidia GeForce GTX 1080 Ti
 - **32** for self-play
 - **1** for network update
 - CPU:
 - Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz

Baseline Parameters

- We trained a baseline model with the following parameters
 - Constant of PUCT
 - Determining the level of exploration, the same as ELF
 - Self-play simulations
 - MCTS simulation number, 800 for Go, 400 here
 - Dirichlet noise
 - Epsilon: the same as AZ
 - Alpha: inverse proportion to the approximate number of legal moves
 - 0.03 for Go, 0.135 for NoGo
 - Residual blocks
 - The more layers, the more training time is required
 - Filter number
 - The more filters, the more training time is required
 - Position sample rate
 - The average sample number per position
 - Replay buffer size
 - The container where we sampled positions from

Parameters	Baseline
Constant of PUCT	1.5
Dirichlet noise epsilon	0.25
Dirichlet noise alpha	0.135
Replay buffer size	recent 100k games
Position sample rate	2
Filter number	64
Residual blocks	5
Self-play simulations	400

Baseline Results

- Training details
 - Sequential training
 - Generated **2** million games
 - About **a** week (**8** GTX 1080 Ti)
- Results of play against HaHaNoGo program

Simulations (wahaha vs. haha)	Baseline (64 filters, 5 blocks)
400 vs. 50000	58.00%
400 vs. 100000	56.00%
1600 vs. 100000	58.50%
10000 vs. 1000000	59.50%

Depth and Width of Networks

- With the same data as baseline, we trained the other three model with different depth or width
 - SL-32-5
 - SL-128-5
 - SL-128-10
- The results show the large network is important
 - It can't learn more information if the size of network is too small
- We won the championship in TAAI 2018 Game Tournament
 - 15 minutes for each program (20 sec per move)
 - Used model **SL-128-5**

Simulations (wahaha vs. haha)	SL training (32 filters, 5 blocks)	Baseline (64 filters, 5 blocks)	SL training (128 filters, 5 blocks)	SL training (128 filters 10 blocks)
400 vs. 50,000	40.50%	58.00%	68.50%	73.50%
400 vs. 100,000	33.50%	56.00%	53.00%	63.50%
1,600 vs. 100,000	36.00%	58.50%	69.00%	75.00%
10,000 vs. 1,000,000	32.50%	59.50%	69.50%	72.00%

Average Sample Rate of Positions

- With the same architecture as baseline, we compare the training result of the model with different position sample rate
 - Training details
 - Semi-Asynchronous Training
 - About **1.5** days (**33** GTX 1080TI)
 - Generated games:
 - 2** million for baseline
 - 1** million for others

Simulations (wahaha vs. haha)	Sample rate 1	Baseline (Sample rate 2)	Sample rate 4
400 vs. 50,000	< 10%	58.00%	65.40%
1,600 vs. 100,000	< 10%	58.50%	63.60%

Larger Network

- We also trained a larger model
 - Used 10 residual blocks, 128 filters, 800 simulations
 - Changed simulations from 800 to 1600 after generating 1 million games
 - Training details
 - Semi-Asynchronous Training
 - About **6** days (**33** GTX 1080TI)

	Baseline	More sample	TAAI Version	Larger network	Larger network (continue)
Self-play simulations	400	400	-	800	1600
Residual blocks	5	5	-	10	10
Filter number	64	64	-	128	128
Position sample rate	2	4	-	4	4
Generated games	2 million	1 million	-	1 million	0.5 million
Result (wahaha vs. haha)	58.50%	63.60%	69.00%	87.20%	90.60%
1,600 vs. 100,000					

Strength of a few simulations

- Policy only

Policy only (wahaha vs. haha)	Policy vs. 1,000	Policy vs. 2,000	Policy vs. 2,500
Larger network	56.40%	30.40%	25.60%

- 50 simulations only

- Time cost per moves
 - wahaha (50 sim): 0.04 sec (40 ms)
 - haha:
 - 100,000 sim : 1 sec (1,000 ms)
 - 1,000,000 sim: 10 sec (10,000 ms)

50 sim only (wahaha vs. haha)	50 vs. 20,000	50 vs. 50,000	50 vs. 100,000	50 vs. 1,000,000
Larger network	89.40%	85.40%	77.40%	52.00%

Conclusions and Future Work

Conclusions and Future Work

- Conclusions
 - We designed a software framework that support AGZ/AZ Training
 - On top of the framework, we developed a powerful NoGo AI program
 - Convincingly beat the state-of-the-art program HaHaNoGo
 - Won the championship in TAAI 2018 Game tournament
 - We are also training other games based on the framework
 - E.g., Othello, Gomoku
- Future Work
 - Try to train multiple action games
 - E.g., Connect6
 - Try to train incomplete information games
 - E.g., Mahjong, Bridge
 - Try to combine with more other algorithms
 - E.g., Threat Space Search, Alpha-Beta search
 - Hope our framework will be helpful to AlphaZero-Like research

Q & A



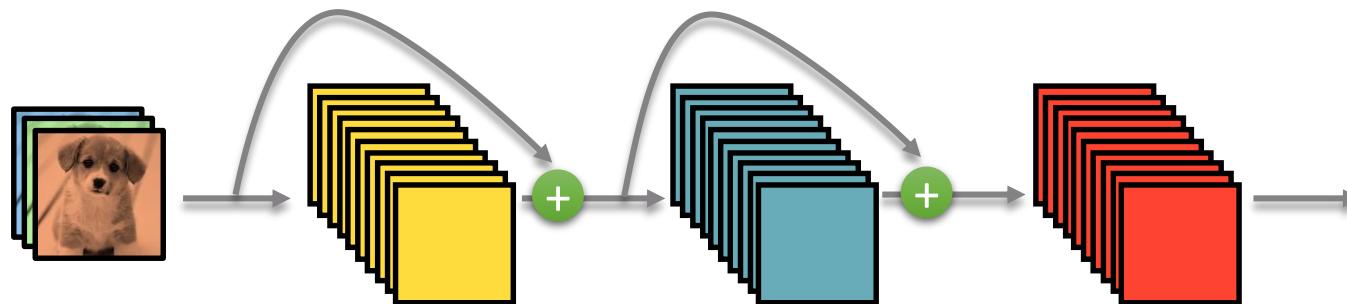
Thanks for your attention

Appendix

Deep Convolutional Neural Network (DCNN)

- **Residual Network^{[1][2]} (ResNet)**

- It is intuitive that deeper networks have stronger capacity to learn, achieving a higher accuracy.
- But the training and test error get higher when we **simply stack layers**
- ResNet addressed this problem by adding shortcut connections into the layers.



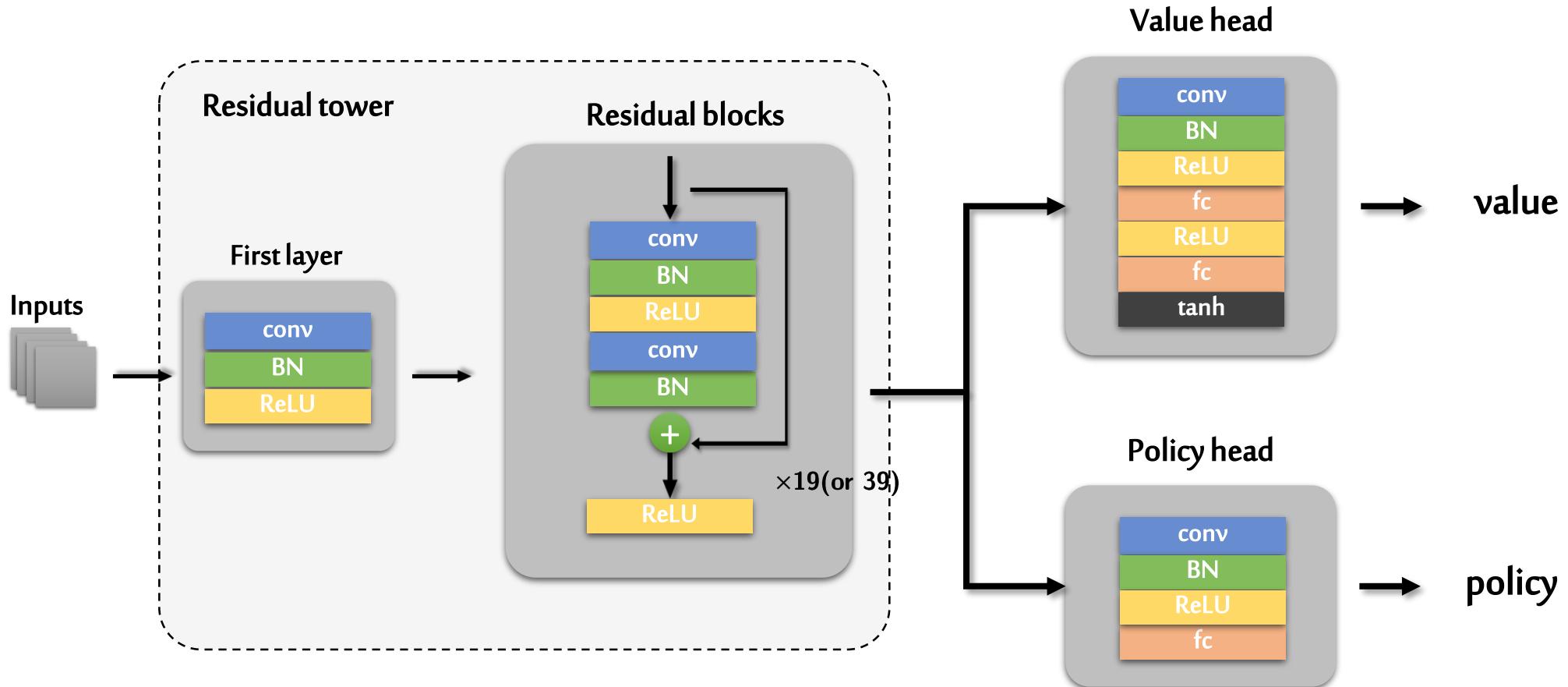
- CVPR 2016 Best Paper Award
- **1st places in three ILSVRC 2015 tasks:**
 - Classification: Ultra-deep 152-layer nets
 - Detection: 16% better than 2nd
 - Localization: 27% better than 2nd

[1] He, Kaiming, et al. "Deep residual learning for image recognition." IEEE conference on computer vision and pattern recognition. 2016.

[2] He, Kaiming, et al. "Identity mappings in deep residual networks." European conference on computer vision. Springer, Cham, 2016.

Deep Convolutional Neural Network (DCNN)

- Residual networks for Go^[1]

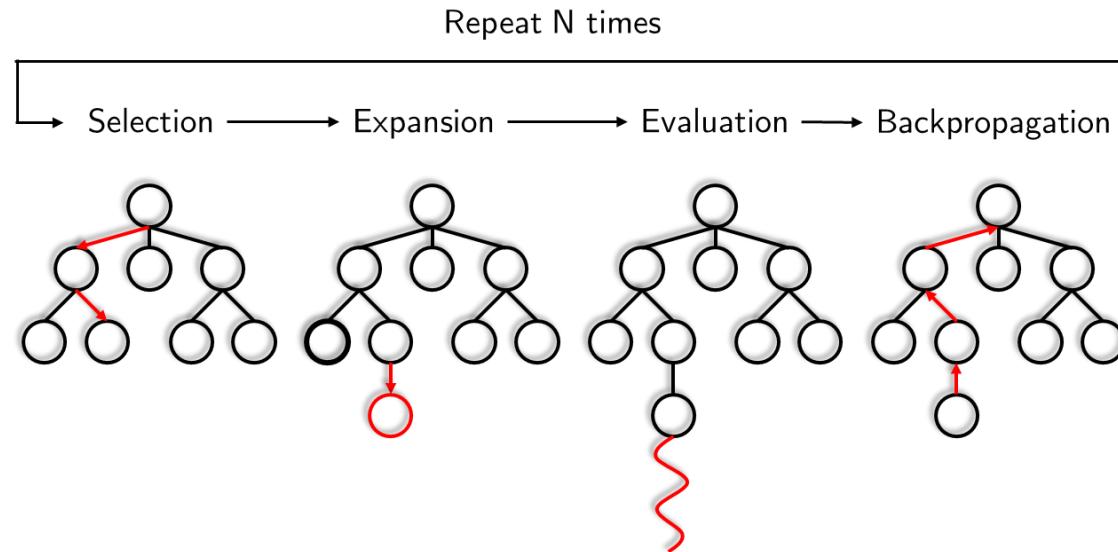


[1] Cazenave, Tristan. "Residual networks for computer Go." IEEE Transactions on Games 10.1 (2018): 107-110.

Monte Carlo Tree Search (MCTS)

- MCTS^[1] is a heuristic search algorithm for decision making
 - MCTS will repeat multi-times of simulations
 - The more simulations we perform, the more convincing the decision is
 - For each simulation, there are four stages:

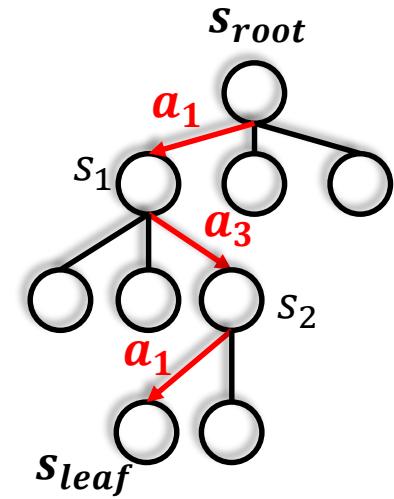
- Selection
- Expansion
- Evaluation
- Backpropagation



[1] Coulom, Rémi. "Efficient selectivity and backup operators in Monte-Carlo tree search." International conference on computers and games. 2006.

Selection

- Start from s_{root} keep select actions until reach leaf state s_{leaf}
- For every selection step t:
 - Select action a_t that has highest score
 - $Score_a = Q(s_t, a) + U(s_t, a)$
 - $Q(s, a)$: the action value (win rate)
 - $U(s, a)$: the bonus value for exploration
 - **UCT**^[1] and **PUCT**^[2] are two widely used select algorithms
 - Succeed in resolving the **exploration** and **exploitation** tradeoff



[1] Kocsis, Levente, and Csaba Szepesvári. "Bandit based monte-carlo planning." European conference on machine learning. 2006.

[2] Silver, David, et al. "Mastering the game of Go without human knowledge." Nature 550.7676 (2017): 354.

Upper Confidence Bound for Trees (UCT)

- Uses (Upper Confidence Bound) UCB^[1] formula:
 - When node p wants to score a child i

$$\bullet \textcolor{red}{Score}_i = Q(s_t, i) + U(s_t, i)$$

$$\bullet Q(s, i) = \frac{W(s, i)}{N(s, i)}$$

$$\bullet U(s, i) = c_{\text{uct}} * \sqrt{\frac{\log(N_p)}{N(s, i)}}$$

- $W(s, i)$: The total win number of node i
- $N(s, i)$: The total visit number of node i
- N_p : The total visit number of node p
- c_{uct} : A constant determining the level of exploration

- Node p will select the child which has max score

[1] Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem." Machine learning 47.2-3 (2002).

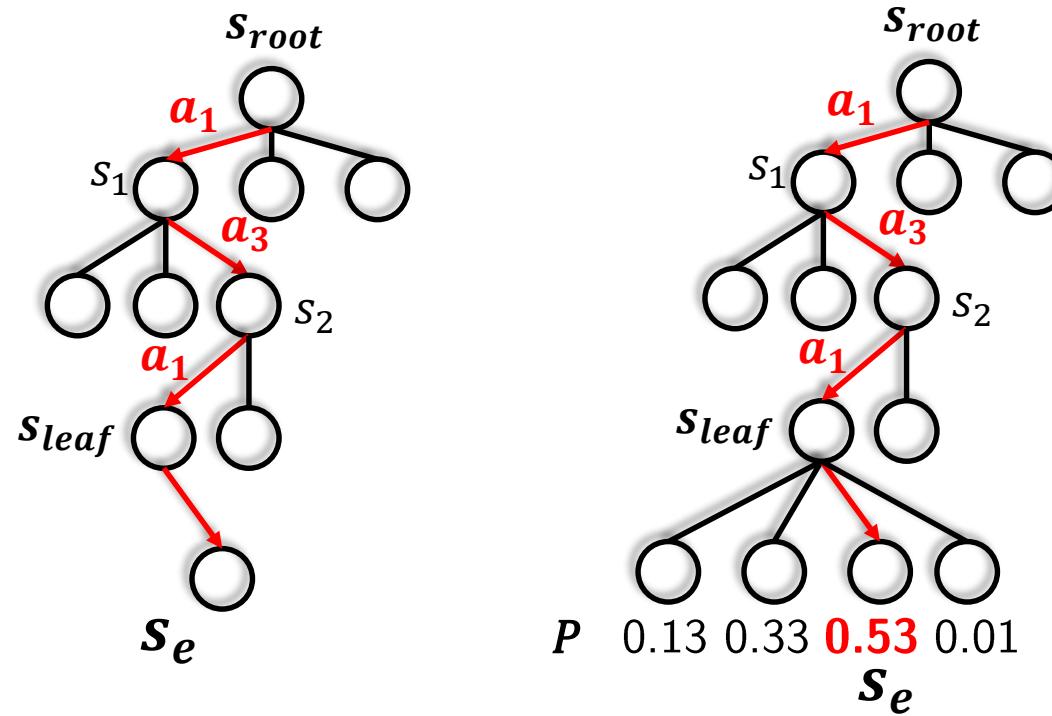
PUCT

- A variant algorithm of **PUCB(Predictor +UCB)**^[1] for Trees
 - Uses the following formula proposed by AlphaGo Zero:
 - When node p wants to score a child i
 - $\text{Score}_i = Q(s_t, i) + U(s_t, i)$
 - $Q(s, i) = \frac{W(s,i)}{N(s,i)}$
 - $U(s, i) = c_{\text{puct}} * P(s, i) * \frac{\sqrt{N_p}}{1+N(s,i)}$
 - $W(s, i)$: The total action value of node i
 - $N(s, i)$: The total visit number of node i
 - N_p : The total visit number of node p
 - $P(s, i)$: **The prior probability of selecting** node i
 - c_{puct} : A constant determining the level of exploration
 - Node p will select the child which has max score

[1] Rosin, Christopher D. "Multi-armed bandits with episode context." Annals of Mathematics and Artificial Intelligence 61.3 (2011): 203-230.

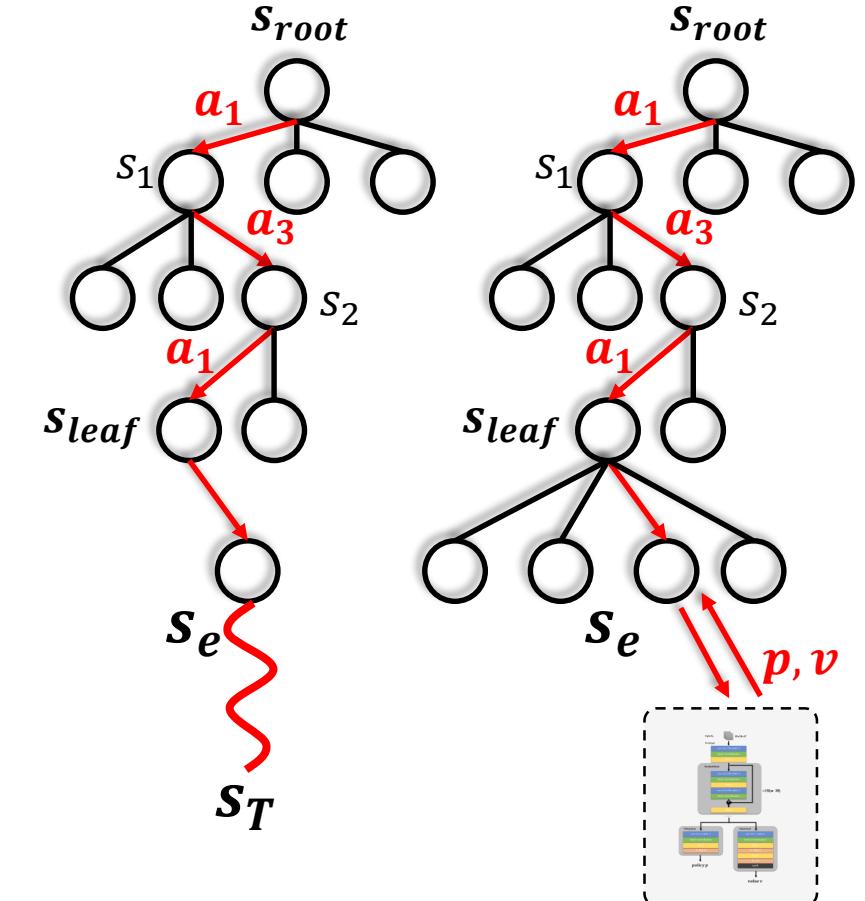
Expansion

- Create one (or more) child nodes and choose node s_e from one of them.
 - Child nodes are any valid moves from the game position
- Expansion policy
 - Random policy
 - PUCT based policy



Evaluation

- Evaluate the new leaf state s_e
 - Rollout (Until reach a terminal state s_T)
 - By random policy
 - By any other rollout policy
 - Evaluate via neural network
 - Policy and Value MCTS (**PV-MCTS**)
 - Just one inference to get the values of the state



Backpropagation

- Update the value on the tree path after Evaluation Stage done
- For Rollout:
 - $N(s_t, a_t) += 1$
 - If win, $W(s_t, a_t) += 1$
- For Evaluate via neural network:
 - $N(s_t, a_t) += 1$
 - $W(s_t, a_t) += v$

