Developer's Guide to CZF

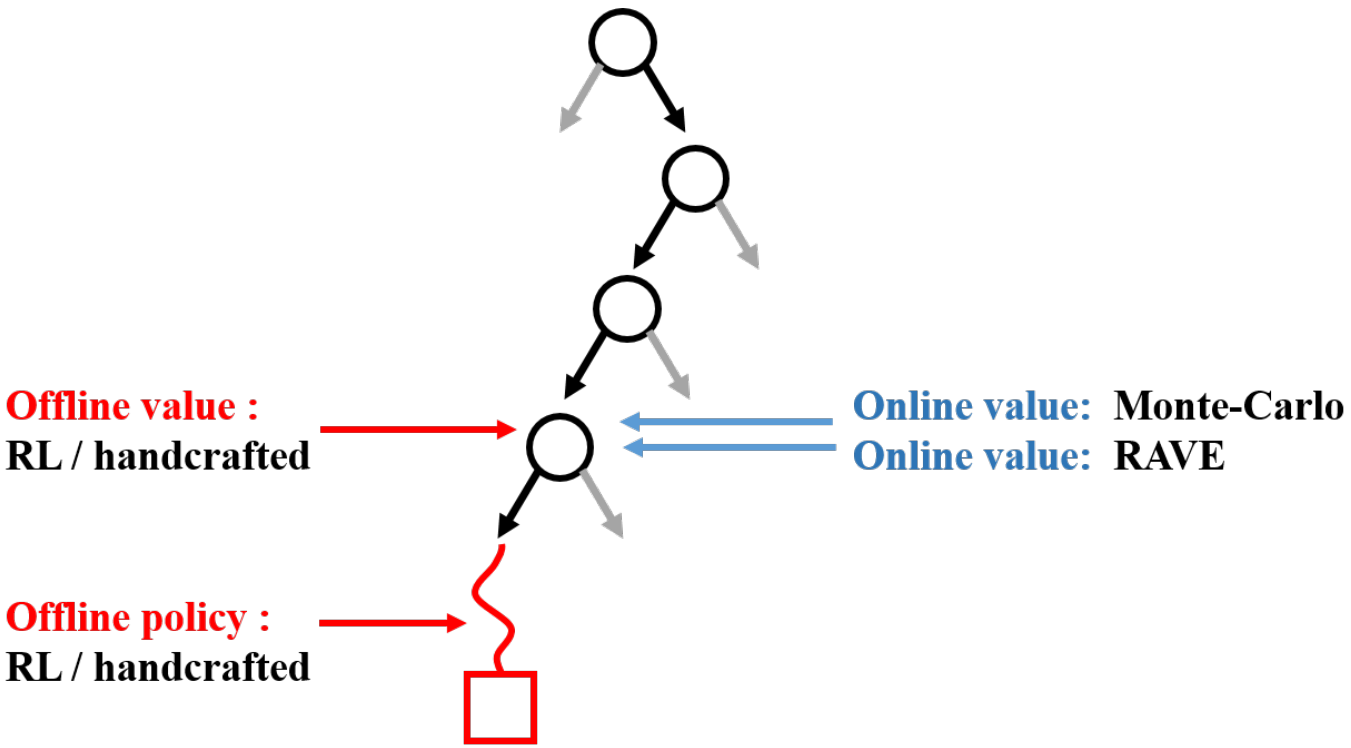
Wei Li

National Chiao Tung University
Computer Games and Intelligence (CGI) Lab

November 22, 2018

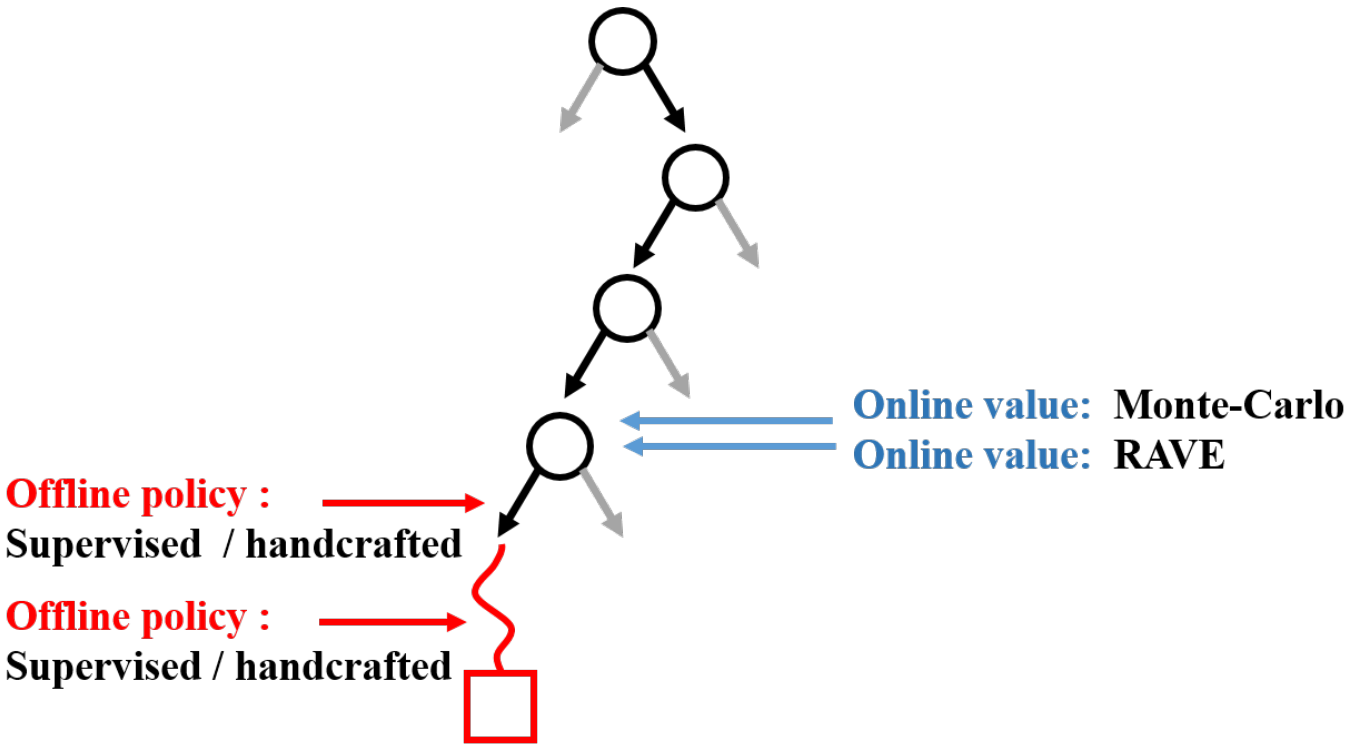
MoGo → AlphaGo → AlphaGo Zero/Alpha Zero

- **MoGo (2007)**
  - Combining Online and Offline Learning in UCT
    - ICML 2007 Sylvain Gelly and David Silver
    - ICML 2017 **Test of Time Award**



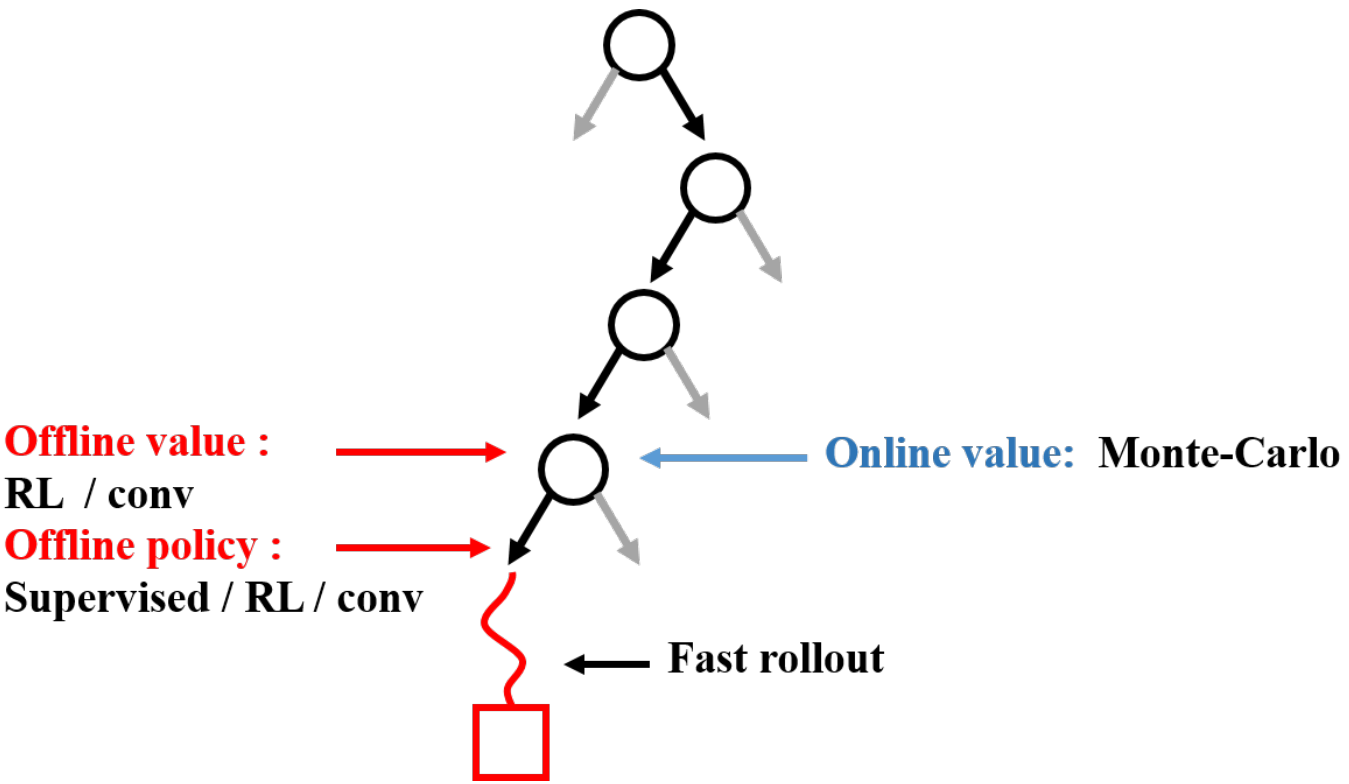
MoGo → AlphaGo → AlphaGo Zero/Alpha Zero

- **Other AI progarms (2008-2015)**
  - Crazy Stone, Zen, etc.



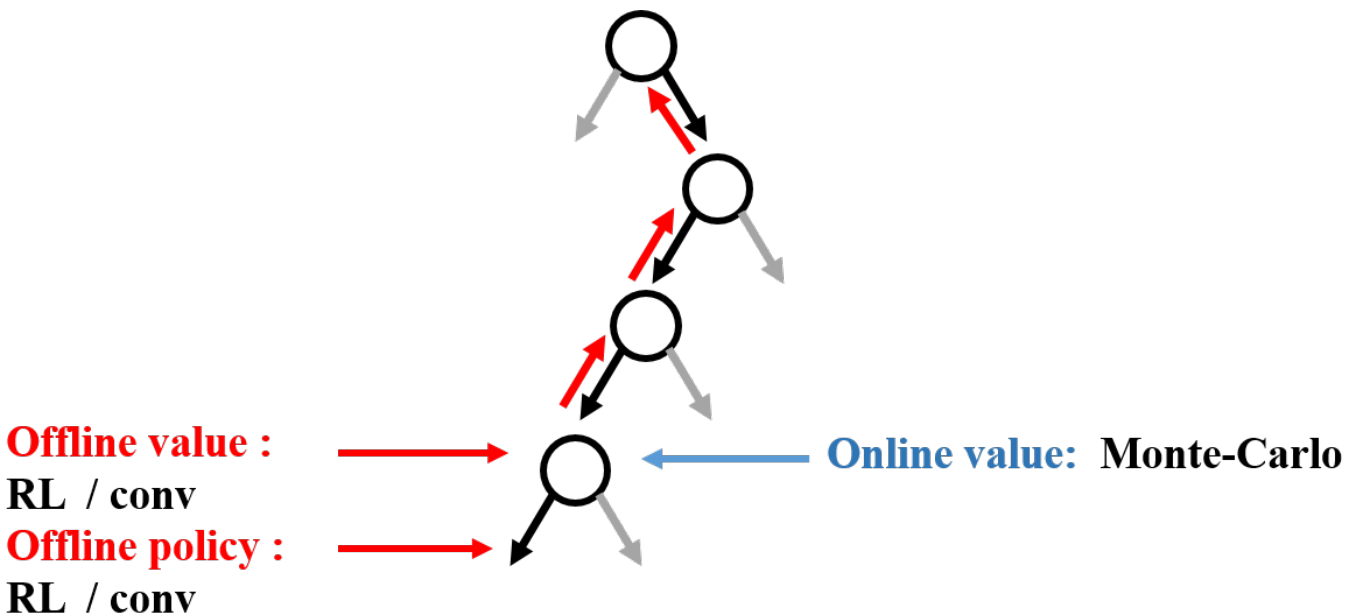
MoGo → **AlphaGo** → AlphaGo Zero/Alpha Zero

- **AlphaGo (2016)**
  - Mastering the game of go with deep neural networks and tree search
    - Nature 529.7587 (2016) David Silver etc.
  - Supervised Learning (**SL**) Policy Network: predict moves
  - Reinforcement Learning (**RL**) Policy Network: improve strength of SL policy network via self-play
  - Value Network (**VN**): estimate value for given position



MoGo → AlphaGo → **AlphaGo Zero/Alpha Zero**

- **AlphaGo Zero/Alpha Zero (2017-2018)**
  - Mastering the game of go without human knowledge
    - Nature 550.7676 (2017) David Silver etc.
  - Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm [arXiv:1712.01815]



What is CZF?

- **CGI Zero Framework**
  - An Clean, Lightweight and Scalable zero training framework
  - Easy to support different games
    - Go, NoGo, Gomoku, Othello, etc.
  - Easy to combine with other algorithms
  - Easy to extend and modify for other research

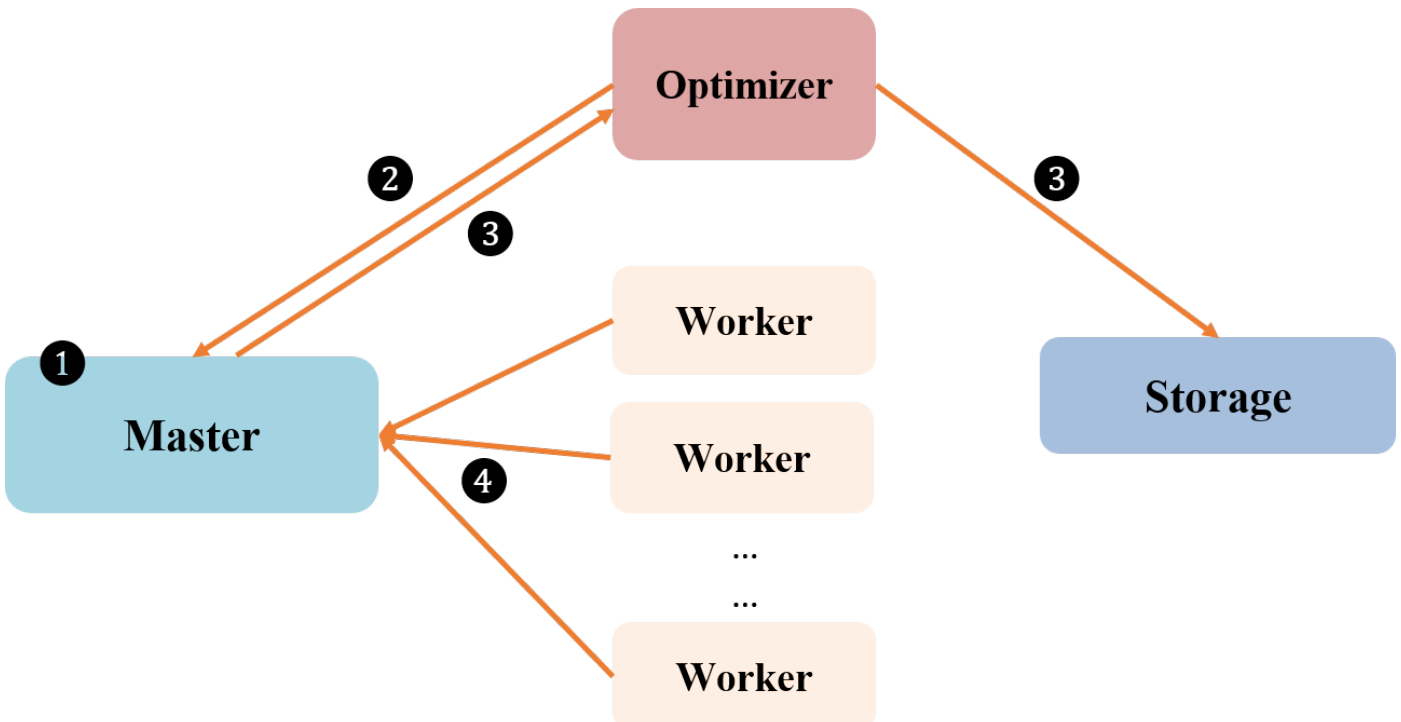
- Three Part of the architecture
  - Master
    - control the whole training
  - Worker
    - (Self-Player) generate game records
    - (Evaluator) evaluate agent strength
  - Optimizer
    - optimize the neural network

Overview of the training workflow

1. Initialization
2. Generation
3. Optimization
4. Evaluation (optional)
5. Repeat 2 - 4 until the network converges

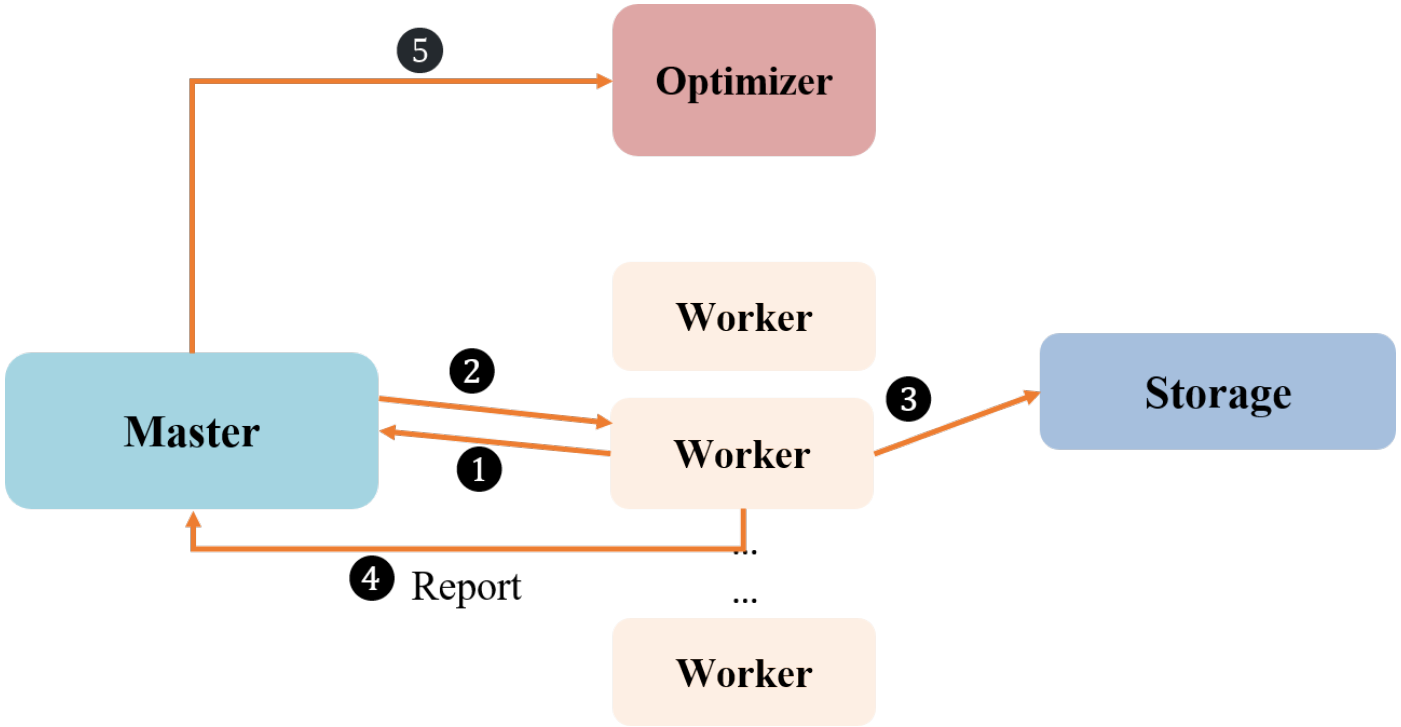
Overview of the training workflow - **Initialization**

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



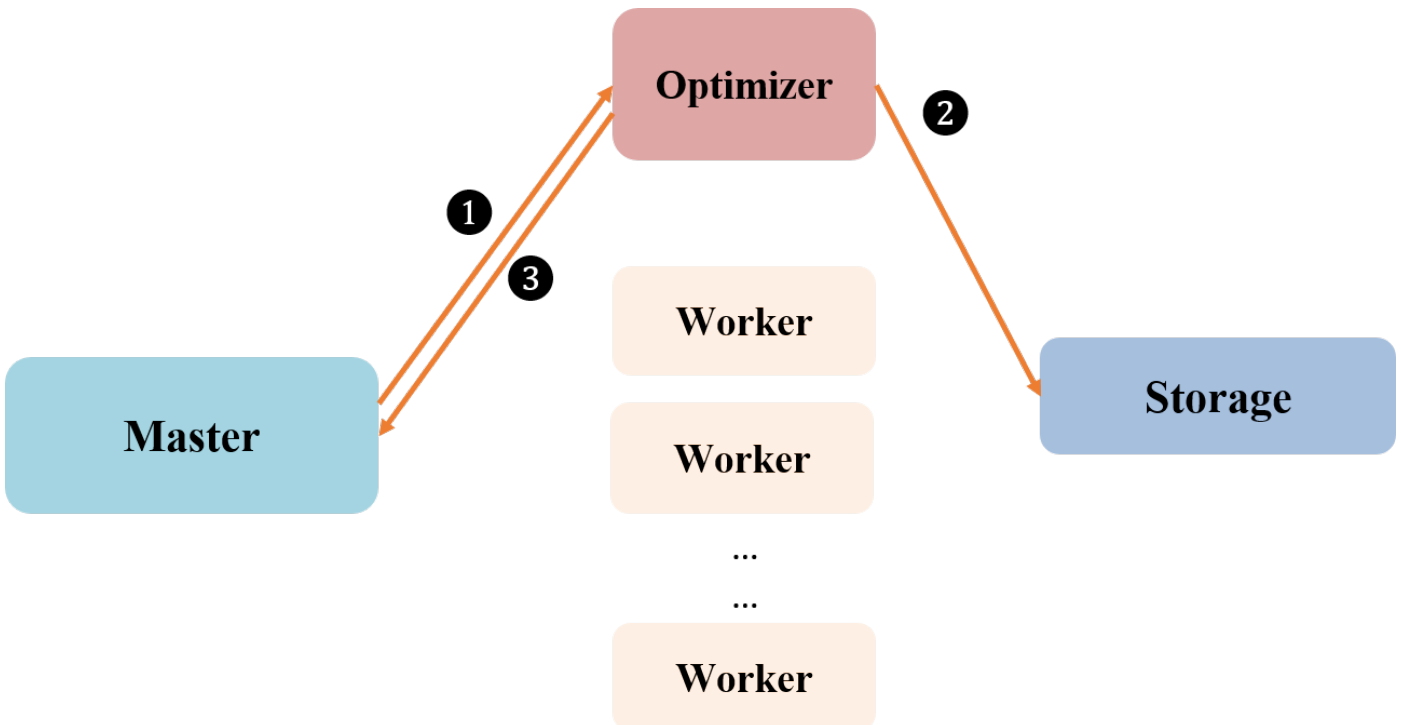
Overview of the training workflow - **Generation**

1. Worker fetch job from Master
2. Master send **SELF_PLAY** job to Worker
3. Worker generate games by self-play, then save records on Storage
4. Worker send report back to Master once the job finished
5. Master send **LOAD_RECORDS** job to Optimizer, then Optimizer will load records and convert to input features



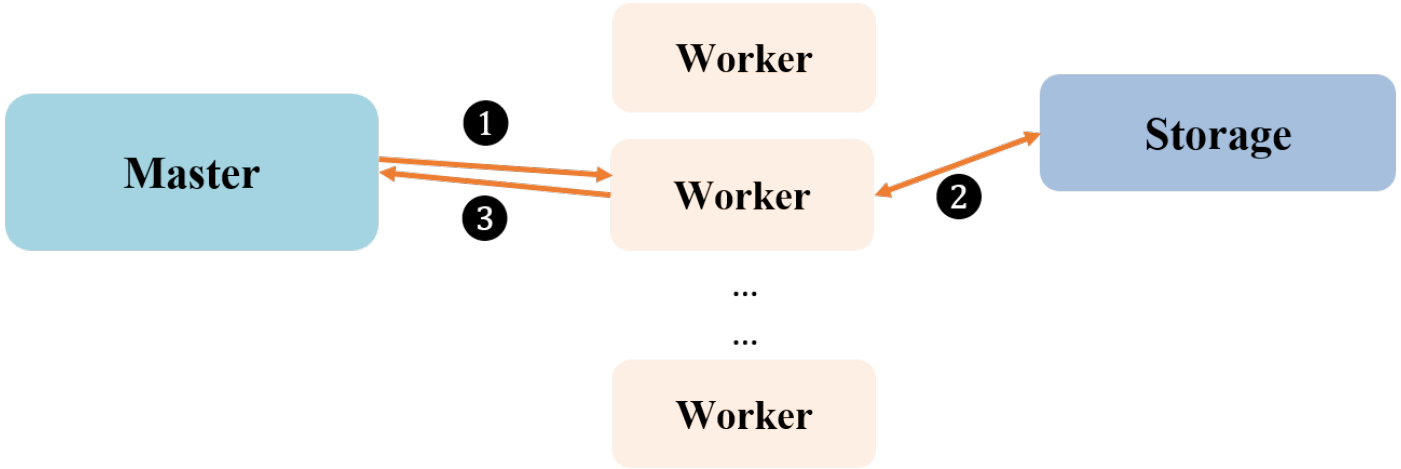
Overview of the training workflow - **Optimization**

1. Master send **OPTIMIZE** job to Optimizer
2. Optimizer update Neural Network, then save them on Storage
3. Optimizer send report back to Master once the job finished



Overview of the training workflow - **Evaluation**

1. Master send **EVALUATE** job to Worker
2. Worker evaluate weights' strength between $f(\theta_*)$ and $f(\theta_i)$ load from Storage
3. Worker send report back to Master once the job finished



Basic Usage of CZF

- Environment and Libraries (setup script)
  - Ubuntu 16.04 +
  - C++ 11 (for inference)
  - Python3.5 + (for control and training)
  - Caffe2 (Tag 0.4.0)

    since there are lots of update between V0.4 and V1.0, you can only build CZF with v0.4 now.
- Compile the program

```
bash setup-cmake.sh
make -j
```

- Package .so files

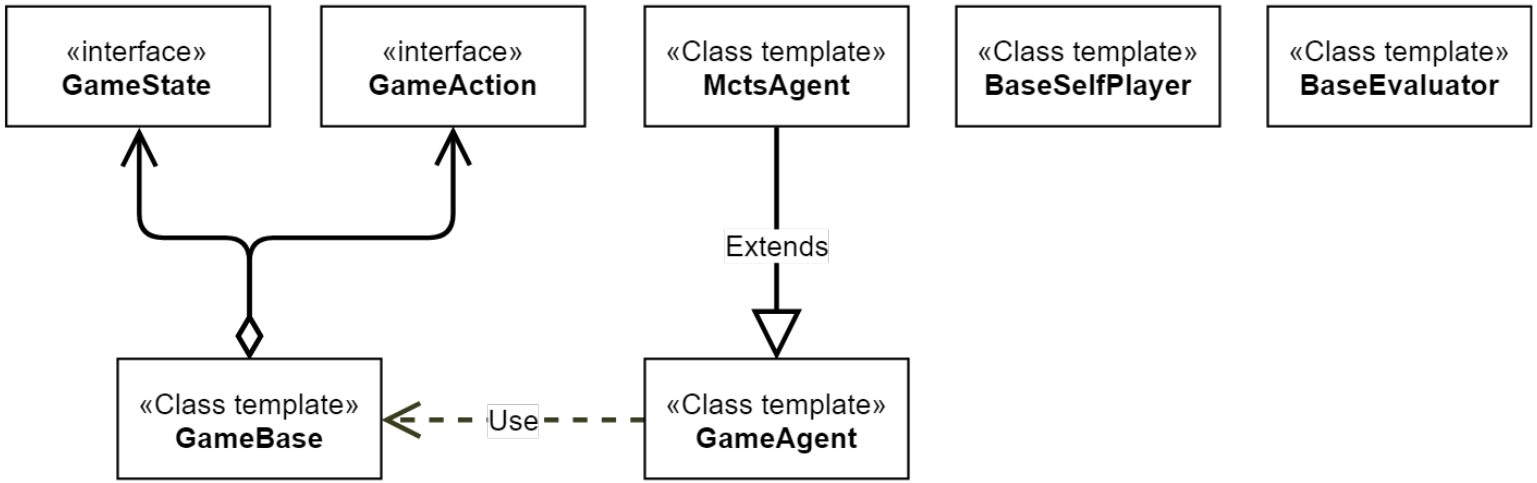
```
ldd Release/ai | grep \=> /" | awk '{print $3}' | xargs -I '{}' cp -v '{}' .
rm libcuda.so*
rm libnvidia-fatbinaryloader.so*
rm libc.so.6
ls -1 lib* | xargs -t -n1 patchelf --set-rpath .
cp /lib/x86_64-linux-gnu/libc.so.6 .
strip Release/ai
chrpath -r . Release/ai
```

After building and packaging .so file, simple copy them to any other computer which has a NVIDIA card.

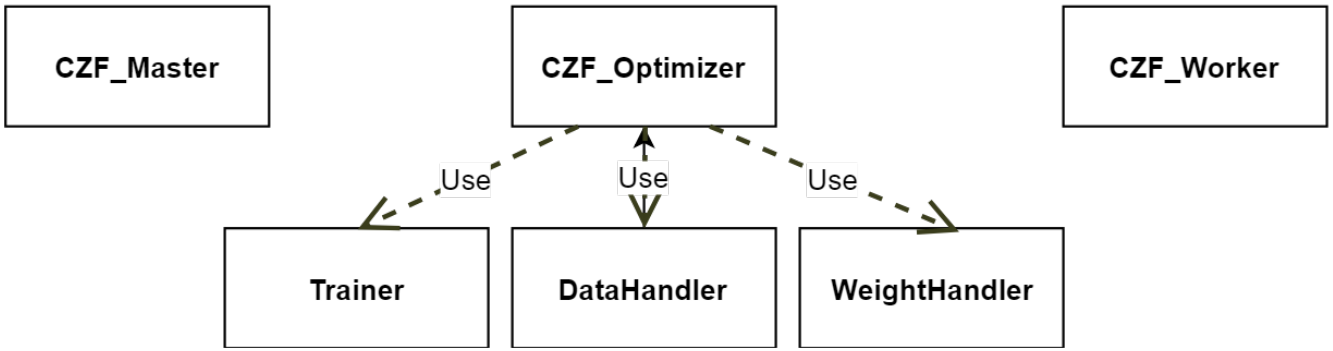
which means you needn't install any software or package (CUDA ,cuDNN, TensorFlow, PyTorch, etc.)

Diagram of CZF

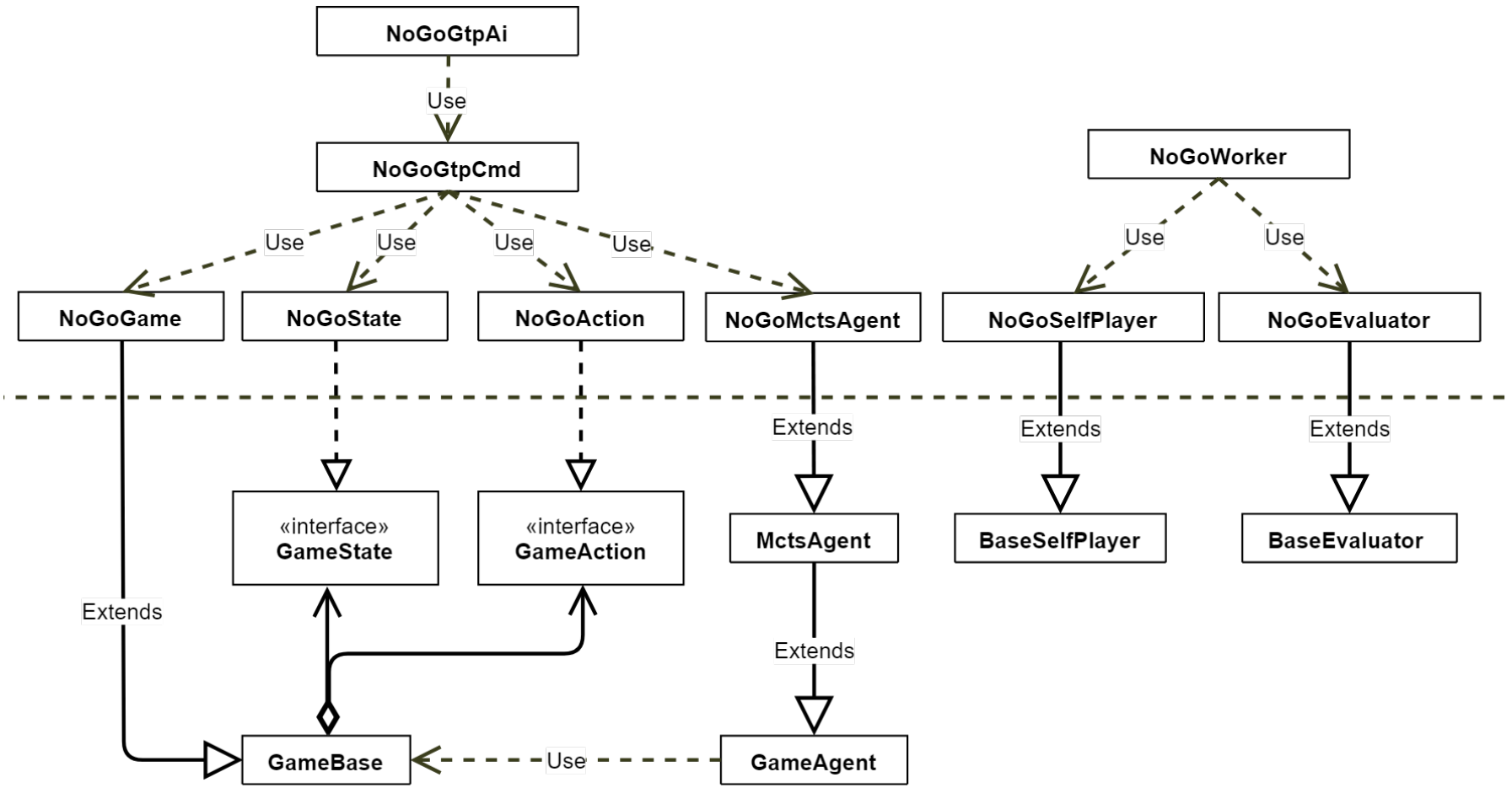
C++ Side



Python Side



Implement an new game (1/2)



Implement an new game (2/2)

- Extend & implement Class you need in C++ side
- Then build C++ executable file (woker and ai)
- Not necessary to modify anything in Python side, Only modify the training config if you need
- Run the Master, Optimizer and Worker, enjoy the Zero training

```
python3 czf_master.py
python3 czf_optimizer.py --host_ip xxxxxx \
                         --continue_to_train 0 \
                         --start_iteration 0 \
                         --learning_rate 0.05
python3 czf_worker.py --host_ip xxxxxx
```

Compare to ELF

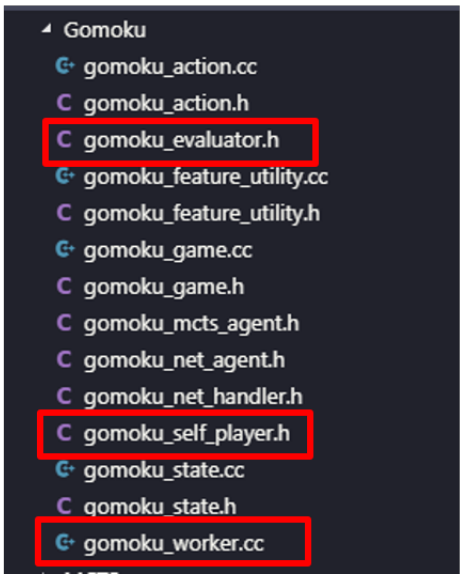
- ELF: An Extensive, Lightweight and Flexible Platform for Game Research
  - On the C++ side, ELF hosts multiple games in parallel with threading
  - On the Python side, ELF returns one batch of game state at a time
  - Both forward and backward need run with Python side
  - Easy to handler RL training problems
- CZF
  - All inferences are on the C++ side, and also can extend more parallel tricks in C++ side
  - Only network's update on the Python side
  - Even change the whole C++ side, training side still work
  - Easy to handler Search based Board Game problems
- Both ELF and CZF use NFS to handler the file transfer

Case Study: Gomoku (1/3)

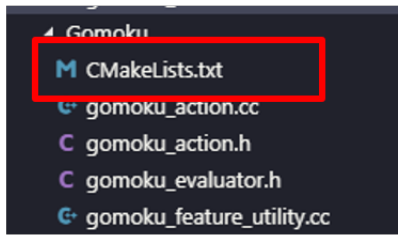
Write game rules

Write NetAgent & MctsAgent

Write Worker

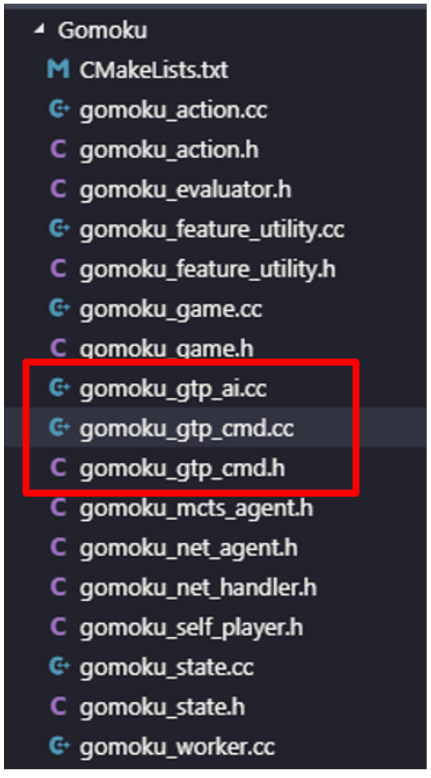


Write CMakeLists



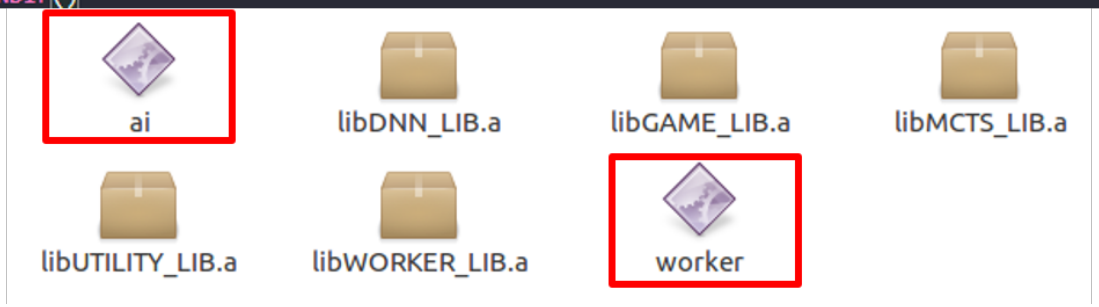
Case Study: Gomoku (2/3)

Write GtpCmd



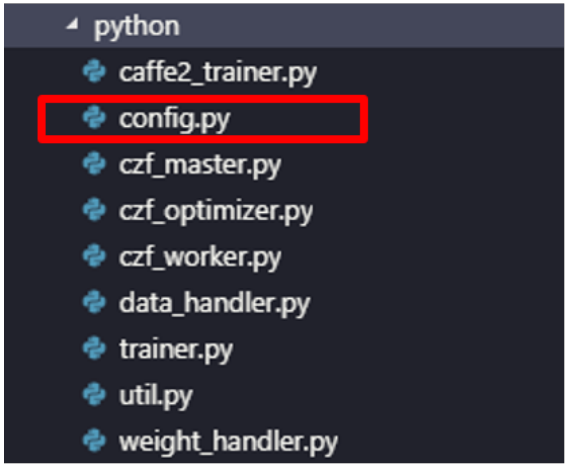
Modify CMakeLists

```
1  SET(SRCS_GOMOKU gomoku_action.cc gomoku_state.cc gomoku_game.cc gomoku_feature_utility.cc)
2
3  SET(SRCS_GOMOKU_WORKER ${SRCS_GOMOKU} gomoku_worker.cc)
4  SET(SRCS_GOMOKU_AI ${SRCS_GOMOKU} gomoku_gtp_cmd.cc gomoku_gtp_ai.cc)
5
6  ADD_EXECUTABLE(worker ${SRCS_GOMOKU_WORKER})
7  ADD_EXECUTABLE(ai ${SRCS_GOMOKU_AI})
8  IF(USE_CAFFE2)
9      # add Deep Learning libraries if use caffe2
10     TARGET_LINK_LIBRARIES(worker WORKER_LIB GAME_LIB MCTS_LIB UTILITY_LIB DNN_LIB ${DNN_LIBRA
11     TARGET_LINK_LIBRARIES(ai WORKER_LIB GAME_LIB MCTS_LIB UTILITY_LIB DNN_LIB ${DNN_LIBRARIES
12 ELSE()
13 TARGET_LINK_LIBRARIES(worker WORKER_LIB GAME_LIB MCTS_LIB UTILITY_LIB DNN_LIB)
14 TARGET_LINK_LIBRARIES(ai WORKER_LIB GAME_LIB MCTS_LIB UTILITY_LIB DNN_LIB)
15 ENDIF()
```



Case Study: Gomoku (3/3)

Set the config



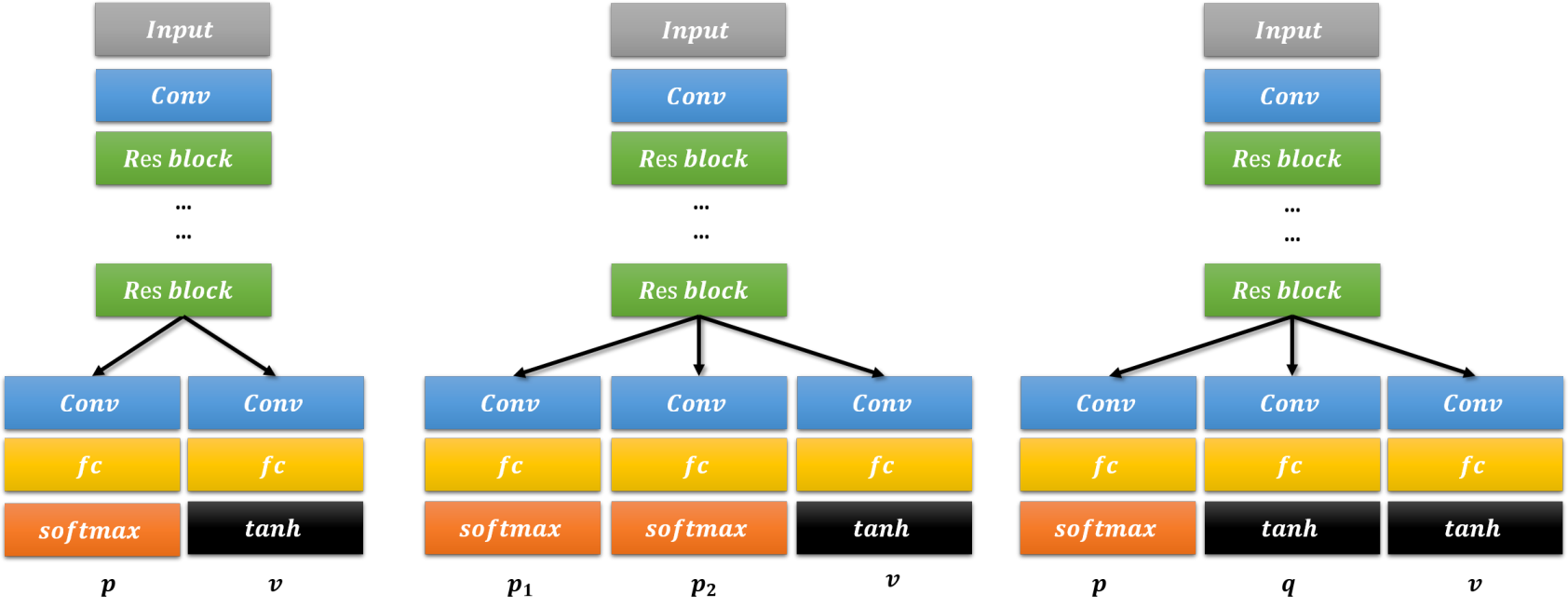

```
# config for game generation
SELF_PLAY_GAMES = 2048
BATCH_SIZE = 32
EVALUATE_GAMES = 64
SAVE_ITERATION = 1000000
# hyper parameters for network architecture
BOARD_LENGTH = 15
BOARD_WIDTH = 15
NUM_INPUT_CHANNELS = 4
NUM_GROUPS = 6
NUM_LABELS = BOARD_LENGTH * BOARD_WIDTH
NUM_FILTERS = 128
TRAINING_BATCH_SIZE = 512
USE_GPU = True
# hyper parameters for replay buffer
MINI_BATCH_NUMBER = 500
REPLAY_BUFFER_SIZE = 3000000
MIN_GAME_TO_TRAIN = 4096
MAX_RENCENT_GAME = 100000
# NFS mount path
NETWORK_SAVE_DIR = '/mnt/czf_gomoku/czf_gomoku_network/'
RECORD_LOAD_DIR = '/mnt/czf_gomoku/czf_gomoku_record/'
```

Some Experiments on CZF

- NoGo
  - Residual Blocks: **5**
  - Simulations: **400**
  - Filters: **64**
  - GPUs: **8 x 1080TI**
  - Training times: about **5** days
  - Generated games: about **2,000,000**
  - **WahahaNoGo** vs. **HahaNoGo**:
    - WahahaNoGo **10K** sim vs. HahaNoGo **1000K** sim
    - **60** Win / **40** lose
- GoMoKu
  - TBD

Basic research you can do with CZF

- Traditional board game training (Go, NoGo, Othello, Hex, Chess, Shogi, etc.)
- Multiple actions game training (Connect6)
- Multiple head neural network architecture
  - Three-Head Neural Network Architecture for Monte Carlo Tree Search (IJCAI 2018)



Advanced research you can do with CZF

- Combine with heuristic algorithms
  - Trained with Zero, use Value with other algorithms (Alpha-Beta search)
  - Prove winning position, use TSS(Threat Space Search) before MCTS
- Extend to POMDP game (Mahjong, Bridge)
- Different Training Methods
  - Train Zero with different neural network architectures
  - Train Zero with multiple neural networks

More About CZF

- Feel free to use it if you want
- Join us and make it more powerful & efficient if you are interested in
- Hope this framework will be helpful to your research