Multi-Agent Reinforcement Learning for Optimized Wildfire Suppression

Doyoung Kwak
dykwak94@dgist.ac.kr

Abstract

In recent years, large-scale wildfires have become more frequent worldwide, including in South Korea. These events highlight the considerable difficulties in wildfire suppression when firefighting personnel and aerial resources are limited. To address this issue, we propose a framework that utilizes multi-agent reinforcement learning within a custom-designed grid-based wildfire simulation environment. In our system, various agents—helicopters, drones, and ground crews—coordinate their actions. We redesigned the observation and reward functions and applied a three-layer convolutional neural network (CNN) in combination with a multi-agent proximal policy optimization (MAPPO) algorithm. The resulting policy demonstrates the ability to reduce fire spread effectively, even with limited resources. This approach offers potential applications in real-world wildfire response scenarios.

Keywords: Wildfire Suppression, Multi-Agent RL, MAPPO, CNN, Simulation Environment

Project Repository: https://github.com/dykwak94/KillFire

1. Introduction

In recent years, the frequency and scale of wildfires have increased significantly. For example, in 2022, wildfires in California burned approximately 8637 hectares [1]. In 2023, Canada experienced wildfires that destroyed about 8.8 million hectares [2]. In 2025, South Korea also experienced numerous huge-scale wildfires [3]. These examples illustrate the growing difficulty in suppressing wildfires, particularly when resources such as personnel and helicopters are insufficient.

Traditional firefighting methods, which often rely on human intuition and experience, are no longer sufficient under modern wildfire conditions. Fires now spread in unpredictable patterns and can ignite in multiple locations simultaneously. Coordinating multiple firefighting agents manually is complex and time-consuming, which allows fires to expand further. To overcome these challenges, we propose a multi-agent reinforcement learning (MARL) approach—specifically using multi-agent proximal policy optimization (MAPPO) to develop fast and efficient suppression strategies.

Section 2 reviews related literature. Section 3 defines the research goal and project scope. Section 4 describes our wildfire simulation environment and the reinforcement learning methodology. Section 5 presents the training process and experimental results. Section 6 provides the conclusion, highlights our contributions, discusses limitations, and suggests future work. Implementation details are included in the appendix.

2. Related Work

Distributed Deep RL for Fighting Forest Fires with a Network of Aerial Robots [4]

Haksar and Schwager (2018) proposed a distributed deep reinforcement learning strategy for a team of unmanned aerial vehicles (UAVs) to autonomously fight forest fires. In their work, the wildfire environment is modeled as a Markov decision process (MDP) on a grid. They demonstrated that exact or approximate solutions are often not practical due to the enormous state space, especially as the number of agents or forest size grows.

To address this, they developed a deep RL approach using Multi-Agent Deep Q Network (MADQN), where each UAV learns its own decentralized policy using only local observations. The policy outperformed a hand-tuned heuristic and scaled well across forest sizes and UAV numbers. Their environment featured grid cells in states such as healthy, burning, or burnt, and agents applied retardant on burning cells.

Their experiment showed that MADQN agents coordinated effectively to suppress fires. They validated their approach both in simulation and with physical robots. However, all agents in their system were identical UAVs with the same action spaces.

Firehose - Wildfire Management simulator using Deep RL [5]

Shen and Curtis (2022) introduced Firehose, an open-source deep RL framework for wildfire management. It uses Cell2Fire as a backend, which considers weather, terrain, and vegetation types. This creates a more realistic simulation for training RL agents.

In Firehose, the environment is a grid where each cell includes features such as fuel, fire, or obstacles. Agents receive observations as a grid image and act to "harvest" cells, meaning suppression by different real-world methods. The reward system penalizes burning cells and rewards suppression. Additional penalties are applied if the agent acts far from fires, encouraging proximity-based action.

They tested several RL methods (PPO, maskable PPO, DQN, A2C supported by StableBaseline3) and found that CNN with maskable PPO performed best. The CNN captured spatial features effectively, and maskable PPO prevented invalid actions.

Firehose supports single-agent scenarios. It uses Python, and its experiments showed that combining CNN with PPO worked best for wildfire suppression in this setting.

3. Problem Definition

3.1 Limits of Previous Studies

Although deep reinforcement learning has proven effective for wildfire suppression, most existing studies are limited to homogeneous agents or single-agent environments. For instance, Haksar and Schwager (2018) used deep Q-networks for UAVs with identical capabilities [4]. The Firehose project supported only single agent learning in a realistic environment.[6]

3.2 Research Goal

The goal of this project is to optimize wildfire suppression strategies for a heterogeneous team of agents—including drones, helicopters, and ground crews—using a CNN-based MAPPO algorithm.

3.3 Assumptions and Scope

- **Grid**: A 20×20 2D-space where each cell has a chance to spread fire to nearby cells.
- **Cell States**: Each cell can be one of three states — TREE (not burning), ON_FIRE (currently burning), SUPPRESSED (already stopped).
- **Agents**: Three types of agents, starting from the grid center. Further details are in Section 4.2.

4. Methodology

4.1 Environment Setup

We initially attempted to use existing wildfire evolving simulator, Cell2Fire. However, Cell2Fire lacks support for step-by-step simulation due to its C++ implementation [5], and complexity to fix all the implementations was high. Alternatively, we tried to modify the Firehose which supports stepwise simulation based on Cell2Fire. However, it is designed for single agent learning and is difficult to introduce multi-agent properties into it. After four days of Cell2Fire revising and another four days of Firehose modifying development efforts, we decided to build our own simulator, **`KillFire`**.

`KillFire` environment features:

- 20×20 grid
- Three cell types: TREE (green), ON_FIRE (red), SUPPRESSED (brown)
- Fire spreads only to nearby cells (not across gaps)
- Two random starting fire spots
- Simple rules for fire spread and agent action

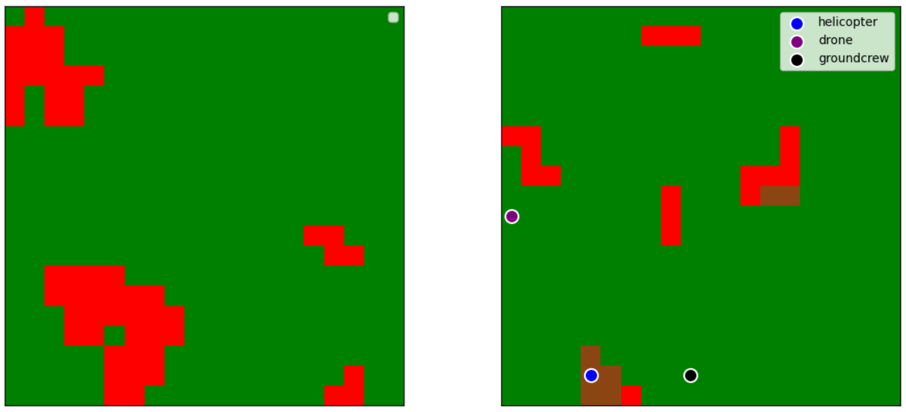

Figure 1 Left grid space shows the fire spreading environment based on the fixed propagation probability. Right grid space contains three agents: helicopter, drone, and groundcrew. In both figures, red cells indicate cells on fire, brown cells mean extinguished cells, and green cells for unburned cells. Each agent has their own distinct color.

4.2 Agent and Simulator Design

`KillFire` includes three types of agents. Each agent observes the full grid and selects actions from the following set: move (up, down, left, right), suppress fire, or stay. If the agents access the fire cell, it executes suppressing action. Suppression mechanics differ by agent type.

Helicopter: Suppress current cell and adjacent 4 cells.

Drone: Suppress current cell and adjacent 2 cells.

Groundcrew: Suppress only the current cell.

Three-layer CNN processes spatial observations for each agent. We used the PettingZoo and Ray RLlib frameworks to implement and train the MAPPO algorithm. CNN used to capture the full grid space has the following structure:

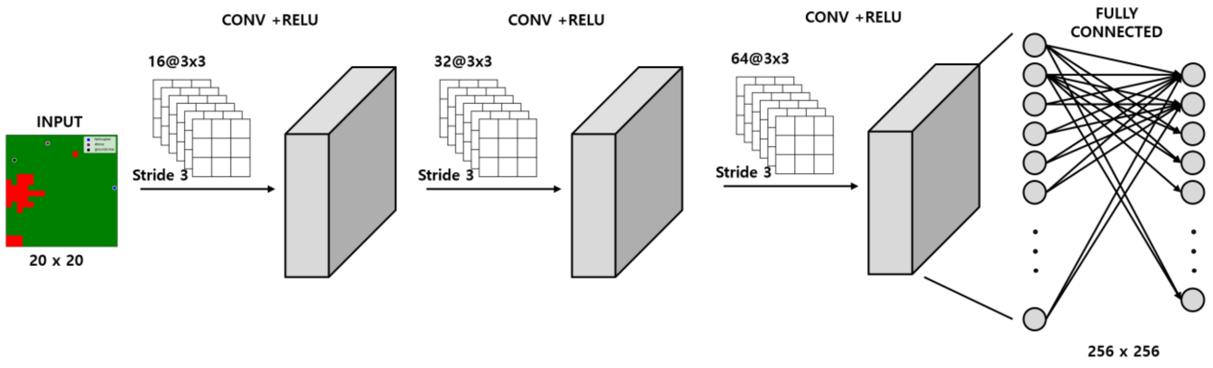

Figure 2 Default structure of CNN in **`KillFire`**

To sum up, **`KillFire`** supports simulation and training process for heterogeneous multi-agent PPO combined with three-layered CNN.

4.3 Reward Function

We designed a reward system with positive points for stopping fires and negative points for ongoing fires:

$$\boldsymbol{R = m \cdot (SUPPRESSED) - n \cdot (ON\ FIRE)}$$

m is called SUPPRESSED coefficient which weighs the positive reward term.

n is called ON FIRE coefficient which weighs the penalty term.

$(SUPPRESSED)$ is the number of extinguished cells in the grid space.

$(ON\ FIRE)$ is the number of burning cells in the grid space.

We used 'Optuna' to find the best values for m and n. More details are in section 5.1.

4.4 PPO Algorithm in multi-agent settings

Since Proximal Policy Optimization (PPO) is one of on-policy reinforcement learning algorithms, PPO is less used than off-policy learning algorithms in multi-agent settings. However, according to Yu.et al., PPO-based multi-agent algorithms showed high performance in four multi-agent testbeds (MPE testbed, SMAC testbed, Google Football testbed, Hanabi testbed). Multi-agent PPO employs centralized training by sharing global information. In addition, MAPPO achieved competitive and ultimate results without complex hyperparameter tuning and domain-specific algorithmic changes.

Therefore, we selected MAPPO for the main algorithm to apply in our wildfire suppressing task. Even though Yu.et al. demonstrated the meaningfulness of MAPPO in multi-agent settings, still it is not applied to various problem including our wildfire management.

5. Training and Results

5.1 Hyperparameter Optimization

We used the 'Optuna' framework for Bayesian optimization of m and n, where $m \in [1, 1000]$ and $n \in [1, 10]$. We set the range of m and n differently to make agents suppress more cells during simulation. Each trial of optimization included 1000 training iterations with a batch size of 400. Due to limited time, we run 20 trials – much lower than 100 trials Yang, et al. used in their research. [7]

First optimization focused on maximizing the area under the cumulative reward curve (AUC) and improving the average reward over the final 10 episodes to make the upward-sloping curve. To reflect the fact that greater return is important than the actual AUC, we weighed final 10episodes term twice bigger than AUC. Since the optimization is designed to indirectly target "steadily increasing" return curve, we visualized the return over iterations to see whether optimized (m, n) set shows increasing return pattern. In Figure 3, all the "top 3 ranked $(m,\ n)$ return curves" do not show increasing tendency. Only the (m=855, n=8.37) return curve rose slightly but it still fluctuated without growing significantly.

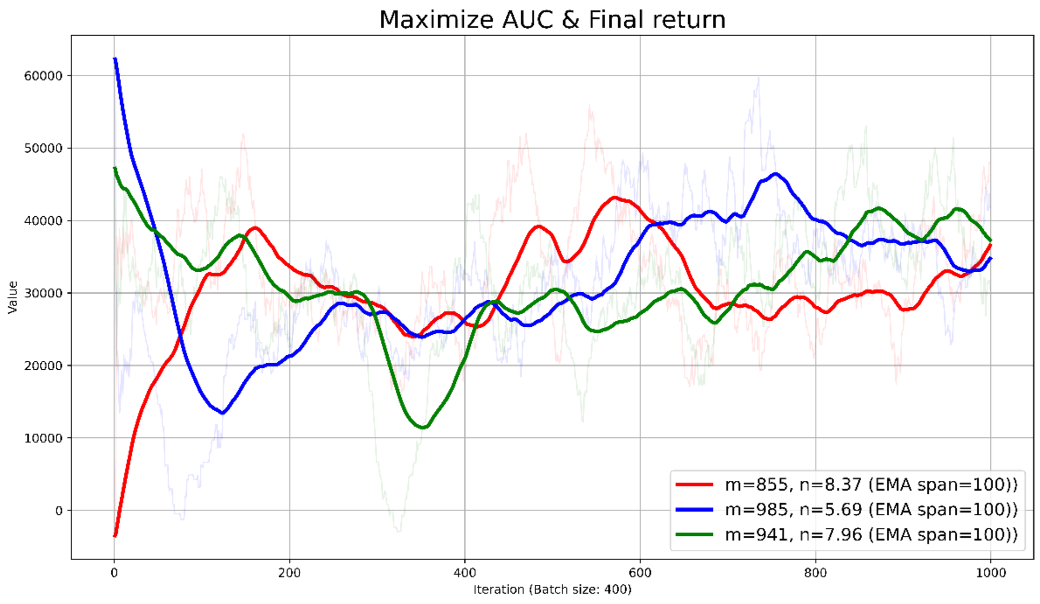

Figure 3 Return curve of top 3 cases in first Bayesian Optimization (n_trials=20)

The first Bayesian optimization's result was not acceptable; therefore, we carried out the second optimization session with different focus point. Again, to find (m, n) set that makes upward-sloping return curve, we adjusted the Bayesian optimization to maximize the difference of final 10 steps return and initial 10 steps return. Second optimization results were more increasing than the first optimization. (Figure 4) Especially, (m=988, n=2.83) set showed steady improvement in entire iteration. Therefore, by considering the result of Bayesian Optimization and return curve observation, we found, (m=988, n=2.83) as the optimal coefficient in reward function.

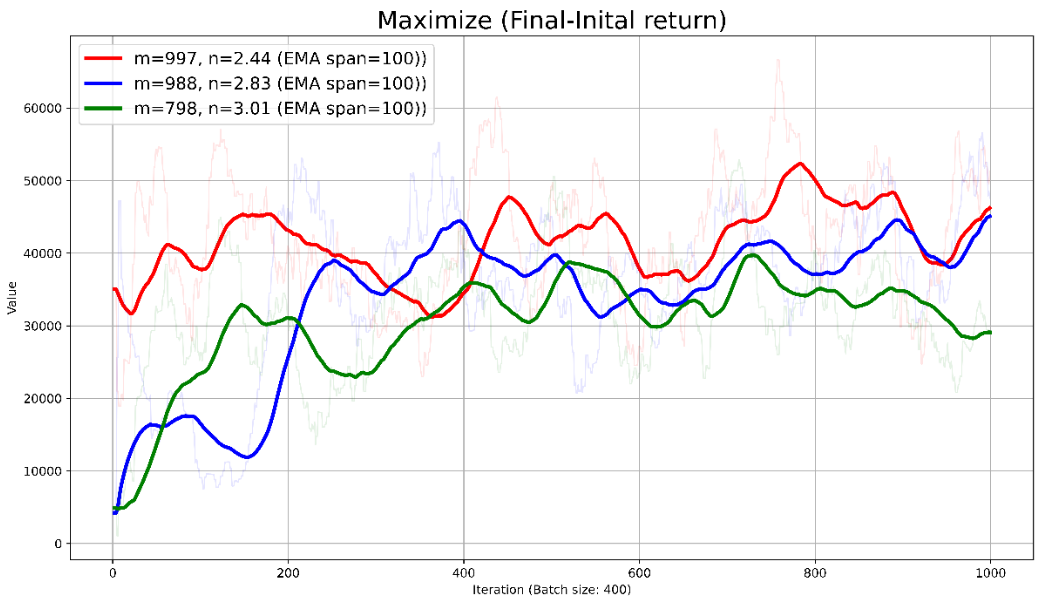

Figure 4 Return curve of top 3 cases in second Bayesian Optimization (n_trials=20)

Second optimization focused on maximizing the extent of rise. To implement it on code, we defined a special reward that takes the difference between average return of initial and final 10 episodes.

5.2 Performance Comparison

We compared the hyperparameter optimized CNN+MAPPO-trained agents against a baseline of agents using random policies. The simulation running condition was.

- Max step per episode: 50
- Training batch size: 400
- Iteration: 10000

The average return of each iteration was calculated in MAPPO and random policy=based, respectively. As Figure 5 displays, in the initial stages, the return of baseline which is pure random policy-based strategy were higher than MAPPO agents. However, the random policy strategy's return curve fluctuates around Return = 10000s. In contrast, MAPPO agents' returns tend to grow upward. Due to the difference of tendency of two reward curve, MAPPO return outstripped random strategy's return. After 2000 iterations, the curve fluctuates without improvement. As a result, MAPPO achieved significantly higher average rewards, indicating successful learning of cooperative fire suppression strategies after the very first stage.

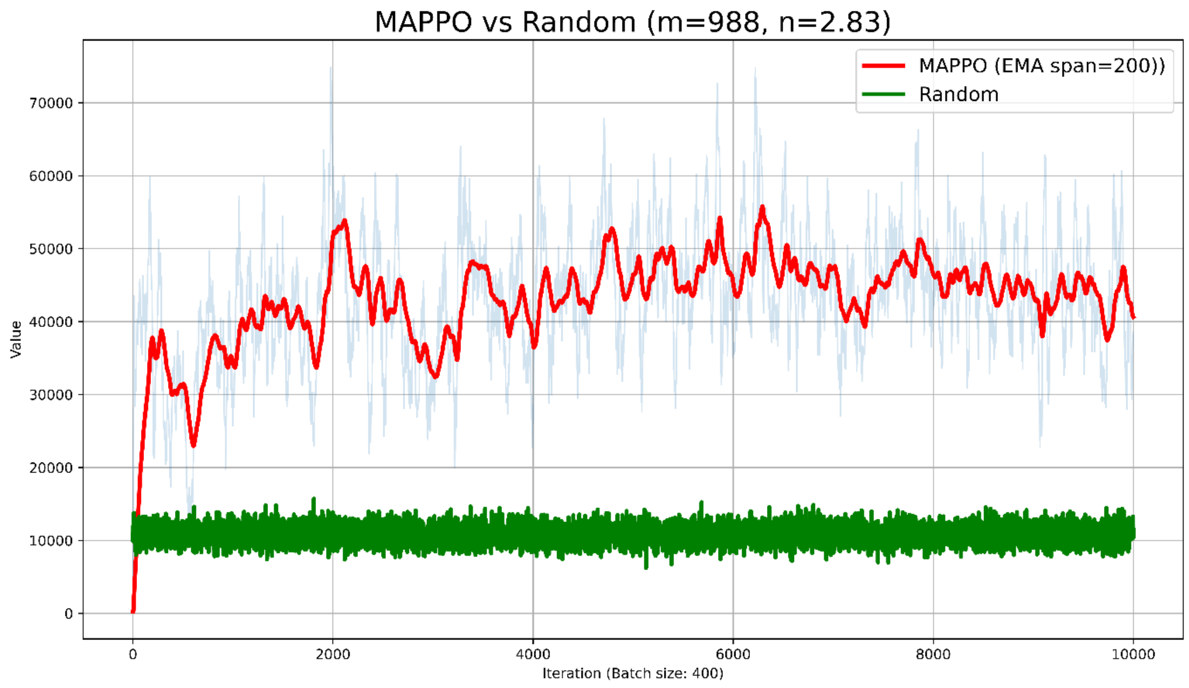

Figure 5 Return of MAPPO and random policy (baseline) smoothen using EMA (span=100).

GIF format files that visualize the movement of agents are posted on our project repository.

6. Conclusion

6.1 `KillFire` Simulator Summary

Feature	Cell2Fire	Firehose	KillFire
Visualization	○	○	○
RL Training Support	X	○	○
Multi-Agent Support	X	X	○
Used Libraries (RL related)	-	SB3	Ray RLlib, PettingZoo
DL Model Types	-	MLP, CNN	CNN
RL Algorithms	-	PPO, DQN, A2C, maskable PPO	MAPPO

`KillFire` is a lightweight yet flexible simulator that supports multi-agent reinforcement learning with heterogeneous agents. Its design allows for future extensions and experimentation. Further possible extensions are in Section 6.4.

6.2 Value of project in RL-based disaster handling field

Developed a custom wildfire simulation framework supporting heterogeneous multiple agents.

Combined and applied CNN and MAPPO to train agents in coordinated suppression tasks.

Demonstrated improved performance over random agent policies.

6.3 Limits

Due to limited development time and too much time consumed work on open-source tools, the simulator lacks some real-world features such as terrain diversity or wind dynamics. Also, the high performance of heterogeneous multiple agent PPO training is not so robust because of lack of comparison with other MARL algorithms.

6.4 Future Work

In further development of **`KillFire`**, reflecting real world components is required. For instance, by adding rock cells or reservoir cells that do not catch fire, or by adding more information like wind or fuel type for each cell. In this way, **`KillFire`** will be able to mimic the real wildfire situation. In addition, expanding the grid space is needed to train agents in vast wildfire sites.

To evaluate or support the robustness of CNN combined – heterogeneous MAPPO algorithm, comparison with another Multi-agent RL such as MADQN must be done.

Compared to the wildfire suppression strategies used in the field, we can argue that our algorithm-trained system can make a meaningful contribution in real-world scenarios.

7. References

[1] CAL FIRE, “Incidents 2022,” https://web.archive.org/web/20220701220944/https://www.fire.ca.gov/incidents/2022/, accessed May 30, 2025.

[2] Euronews Green, “Storms cause record-breaking wildfires in Canada as British Columbia burns,” https://www.euronews.com/green/2023/07/11/storms-cause-record-breaking-wildfires-in-canada-as-british-columbia-burns, accessed May 30, 2025.

[3] The Guardian, “South Korea fires: death toll rises in worst in history,” https://www.theguardian.com/world/2025/mar/27/south-korea-fires-death-toll-rises-worst-in-history, accessed May 30, 2025.

[4] R. N. Haksar and M. Schwager, “Distributed Deep Reinforcement Learning for Fighting Forest Fires with a Network of Aerial Robots,” in Proc. 2018 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 1067–1074, doi: 10.1109/IROS.2018.8593539.

[5] C. Shen, “Firehose: Wildfire Prevention and Management Using Deep Reinforcement Learning,” unpublished manuscript, 2024. [Online]. Available: https://williamshen-nz.github.io/firehose/firehose-paper.pdf. [Accessed: May 30, 2025].

[6] A. Curtis, “Firehose,” GitHub. [Online]. Available: https://github.com/aidan-curtis/firehose. [Accessed: May 30, 2025].

[7] Y. Liu, J. Zhang, and Y. Wang, "The impact of Bayesian optimization on feature selection," Scientific Reports, vol. 14, no. 1, p. 54515, Jan. 2024. [Online]. Available: https://www.nature.com/articles/s41598-024-54515-w

8. Appendix

A. Implementation Details of `KillFire`

A.1 `ForestFireEnv` Class Overview

The `ForestFireEnv` class defines the custom multi-agent wildfire simulation environment. Key features include:

- Agents: 'helicopter', 'drone', 'groundcrew'
- Suppression Ranges:

Helicopter: 5 cells (cross pattern)

Drone: 2 cells (forward line)

Groundcrew: 1 cell (self-only)

- Cell States: 0 (TREE), 1 (FIRE), 2 (SUPPRESSED)
- Actions: Encoded as integers and mapped to meanings (`move_up`, `move_down`, etc.)
- Fire Spread: Probabilistic, to nearby cells only
- Reward Function: `R = m *fire_suppressed – n* (abstract of ON_FIRE cells)`

A.2 Training Script (`train_rllib.py`)

- Framework: Ray RLlib
- Algorithm: MAPPO
- Libraries: PettingZoo (environment wrapper)
- Training Parameters:

  Iterations: 1000

  Batch Size: 400

  Max Steps per Episode: 50
- Logging: Result stored in `progress.csv` of checkpoints generated by Ray RLlib.

B. Computation and Time Consumption

We run the train process or hyperparameter optimization session on the following environment mainly.

- Machine: Apple M4 Pro chip
- CPU: 12-core (Apple Silicon, M4 Pro)
- GPU: 16-core integrated GPU (Apple Silicon, M4 Pro)
- RAM: 24 GB unified memory
- Storage: 512 GB SSD
- Neural Engine: 16-core Neural Engine (Apple Silicon, M4 Pro)
- Operating System: macOS Sequoia 15.5

We used only CPU cores without GPUs.

Executing Speed was approximately 20 minutes per iteration regardless of number of CPU core used. This is because RL lib is able to use multiple cores during initializing session, but fixed CPU was used during each iteration.

Due to the time-consuming iterations, every step of training, comparison, and simulation required a long time. For instance, Bayesian optimization to find optimal hyperparameters, it took at least 6 hours (iteration of Bayesian Optimization = 20).

C. Supplementary Figure, and Data

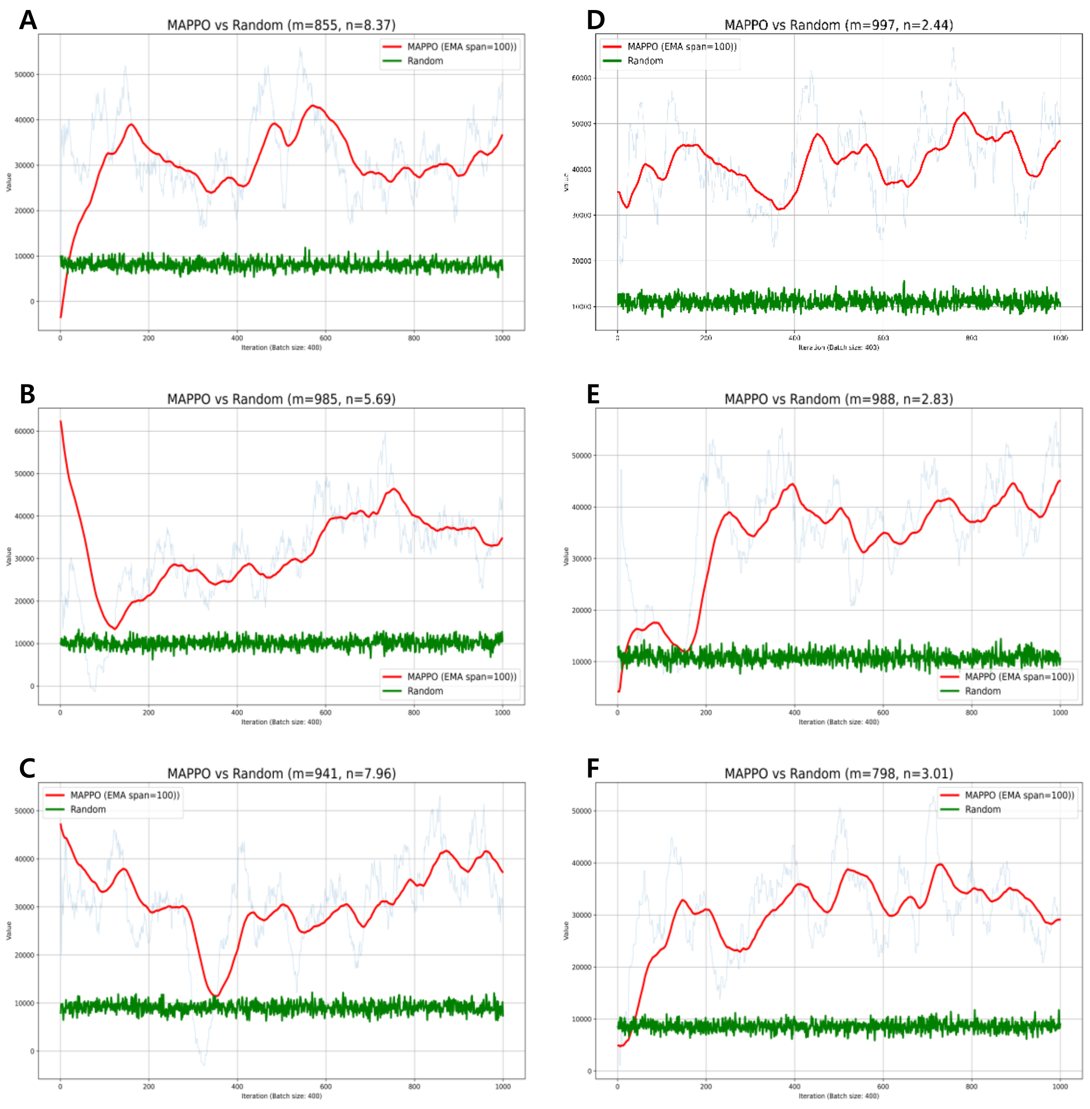

Figure S1 Comparison of random and MAPPO of top 3 ranked cases**.** (A), (B), (C) shows the return curve of three best (m, n) set of first hyperparameter optimization. (D), (E), (F) shows the return curve of three best (m, n) set of second optimization. X-axis stands for iteration and y-axis stands for return. Red plot indicates the return of MAPPO, and green plot is the return of random strategy.

Raw data (csv) of two Bayesian Optimization :

http://github.com/dykwak94/KillFire/hyperparameter_optimization

Tensorboard scalar return_mean plot of Bayesian Optimization :

http://github.com/dykwak94/KillFire/hyperparameter_optimization/svg