🔥 Firehose 🚿: Wildfire Prevention and Management using Deep Reinforcement Learning

William Shen*
MIT CSAIL
willshen@mit.edu

Aidan Curtis*
MIT CSAIL
curtisa@mit.edu

Abstract

With the advent of climate change, natural disasters including wildfires have been occurring with increasing frequency and intensity. Mitigation strategies including forest and wildfire emergency management often require decision making by humans, where time constraints and pressures can result in suboptimal decisions leading to an inefficient use of resources and potential unwanted damages. We present Firehose, an open-source deep reinforcement learning (DRL) framework for training and evaluating wildfire management agents[1]. Underlying Firehose is Cell2Fire, a high-fidelity wildfire simulator that considers factors including weather, topology, and vegetation types to model our environment and generate observations. We detail the state spaces, action spaces, reward functions, policy architectures and RL algorithms we considered, along with a study of this parameter space. Our experimental results demonstrate that we are able to train RL agents which successfully control and suppress the spread of wildfires in simple yet realistic wildfire environments. In certain scenarios, our trained agents are able to match or outperform human-designed baselines. Videos of our learned policies in action are available at: https://williamshen-nz.github.io/firehose

1 Introduction

In the last decade, we have observed a consistent increase in the number natural disasters around the world including wildfires. The 2019-2020 bushfires in Australia caused over 479 human fatalities, 1 billion animal deaths, and $75 billion USD of damage. Although wildfires are inevitable, there are several approaches we can take to mitigate their harmful effects and limit damage.

Wildfire spread is a complex process with many interdependent factors including vegetation, topography, and weather. Significant progress has been made in predicting and modelling the long-term ecological and immediate physical effects of forest fires [Taylor et al., 2013, Coen, 2018]. Several machine-learning based modelling approaches such as support vector machines [Mayr et al., 2018], artificial neural networks [de Bem et al., 2018], and cellular automata [Alexandridis et al., 2008] have been shown to successfully predict the dynamics of fire spread.

Such models have been used as justification for the efficacy of certain fire-fighting policies and strategies over others. For example, a new wildfire prevention policy instituted in Mediterranean France in 1987 limited fuel concentrations and acted more aggressively in the beginning stages of a fire [Curt and Fréjaville, 2017]. While these strategies are very general and have been proven to work in many real-world situations, they are by no means optimal and fail to consider many specifics that change drastically on a per-fire basis. In this work we build a framework for extracting high-throughput fire-specific reactive policies from wildfire spread simulators.

[1] https://github.com/aidan-curtis/firehose

6.484 Computational Sensorimotor Learning Final Project (Spring 2022).

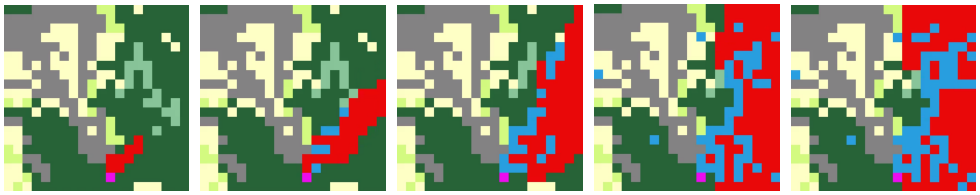


Figure 1: Visualization of our learned policy as time progresses in the fixed ignition point setting. The pink cell corresponds to the ignition point, red cells to active fire, blue cells to harvested cells (i.e., actions), grey to rocks, and different shades of green correspond to different vegetation types. Evidently, our RL agent first builds a perimeter around the fire to prevent it from spreading further, and then begins extinguishing cells within the perimeter.

Recently, deep reinforcement learning (DRL) has gained popularity as a method performing black-box optimization from high-dimensional input given a simulator. DRL has been used to learn controllers that outperform humans in Atari games [Mnih et al., 2015], government policies that maximize economic productivity [Zheng et al., 2022], and skills for robotic manipulation [Levine et al., 2016].

To enable rapid experimentation with various learned and hand-designed fire fighting policies, we built `Firehose`, a DRL framework for training and evaluating agents to combat wildfires. `Firehose` is driven in the backend by Cell2Fire [Pais et al., 2021], a state-of-the-art wildfire simulator. By using DRL, we hope to directly reduce the severity of these threats through active wildfire suppression and intelligent forest management. Our hope is that our approach will allow more effective decision making to minimize the threats and damages caused by such natural disasters.

We demonstrate our framework with a number of experiments comparing DRL baselines to hand-designed policies based on real-world fire fighting strategies. Our results show that the learned policies are capable of outperforming the hand-designed policies in tasks where the ignition point of a fire is known a-priori, but struggle with randomized ignition points. Lastly, we show that `Firehose` enables reward shaping to prioritize the protection of particular regions. In all tasks, we observed that our RL agents demonstrate emergent behavior such as building perimeters around the fire, building a wall to protect privileged zones, and harvesting dense vegetation at choke points in the forest.

2 Related Work

Past research in using machine learning (ML) methods in wildfire applications are often associated with detecting the ignition point and active perimeter of a fire, or predicting the spread and effects of wildfires [Jain et al., 2020, Radke et al., 2019]. Ganapathi Subramanian and Crowley [2018] use spatial reinforcement learning to predict the dynamics of a wildfire from satellite images. Hodges and Lattimer [2019] use supervised learning to train Convolutional Neural Networks (CNNs) to predict wildfire spread using factors including topography and weather, and demonstrate that their approach is significantly more computationally efficient in comparison to traditional simulators.

On the other hand, the use of ML methods to manage and actively combat wildfires has seen less attention. McGregor et al. [2017] use black-box optimization to guide high-level policy decision making on whether to combat a wildfire. They consider factors including the monetary expenses of suppressing fires, timber revenues and air quality to guide high-level decision making. Coffield et al. [2019] use decision trees to predict the final size of the fire given an ignition point – information which may be used to efficiently allocate resources when combating a wildfire based on a fire's intensity and structure. In contrast to these works, we are concerned with learning low-level policies that select which areas of a forest to target in managing and suppressing actively spreading wildfires.

Most closely related to our approach is Haksar and Schwager [2018], who use deep reinforcement learning to train a team of Unmanned Aerial Vehicles (UAVs) to combat a wildfire in a less dynamic forest environment. They train Multi-Agent Deep Q-Networks and demonstrate that it outperforms a hand-tuned heuristic. In contrast to this work, we consider a single-agent environment where the dynamics of our wildfire simulation are significantly more complex and based on factors including vegetation type, topology and weather. Additionally, we consider a significantly larger action space (e.g. 400 actions for a 20×20 environment) in comparison to the nine possible actions per UAV they consider, albeit we operate in a single-agent environment.

3 🔥 Firehose 🚿

Firehose is an open-source deep reinforcement learning (DRL) framework for training and evaluating wildfire management agents in realistic environments. Firehose allows researchers to easily train and evaluate a variety of RL agents on diverse environments, extend our state/action spaces and reward functions, and design hand-crafted baselines. In this section, we discuss the underlying simulation environment Cell2Fire, state spaces, action spaces, and reward functions we considered.

3.1 Simulation Environment

Firehose uses the recent state-of-the-art simulator Cell2Fire [Pais et al., 2021] to drive its low-level simulations. Cell2Fire is an open-source and comparatively efficient wildfire spread simulator written in C++, and is based on the Canadian Forest Fire Behavior Prediction System. Cell2Fire uses several factors to determine the spread of a wildfire, including:

- **Vegetation Type**: different vegetation act as different sources of fuel, and hence result in different rates and intensities of spread. For example, open grasslands have a low amount of fuel in comparison to dense forests and thus fire may spread with less intensity in the former.
- **Wind Speed and Direction**: these factors influence how fast wildfires are able to spread.
- **Topography** (slope/elevation): wildfires spread faster uphill because heat and ember rise.

We modified Cell2Fire to support consuming input actions at every simulation time step. An action in our framework is to **harvest** a cell in a grid – this permanently extinguishes a fire if the cell is on fire, and prevents any future fire from spreading to the given cell. Harvesting is analogous to real-world strategies including controlled burns, dumping fire retardant and active suppression using water.

Firehose exposes a custom Gym environment [Brockman et al., 2016] which wraps around Cell2Fire and provides additional functionality. We terminate an episode once the Cell2Fire simulation finishes, which occurs either when the fire has been completely put out, or when we run out of weather data.

3.2 State Space

We assume that our agent is combating wildfires in a forest area of fixed size. In a real-world environment, a human operator may use satellite or aerial imagery to manage and combat wildfires. In a similar vein, we use a generalized bird's-eye view grid representation of the forest, where each cell represents the type of entity that occupies it (e.g., trees, grass, water, or fire) along with any relevant features. Examples of this grid representation as RGB images are depicted in Figure 1.

We represent a forest area as a grid of size $H \times W \times C$ for height, width, and channels (features) of each cell, respectively. Every episode in an environment has an **ignition point**, which is the point from which the fire begins spreading. This ignition point may stay fixed or change between episodes.

In our default implementation we use $C = 3$ channels, where the feature for each cell is a vector $[r, g, b]$ representing the color and hence the type of entity within the cell – e.g., vegetation type, water, ignition point, fire. These represent vital input signals to the agent. As we discuss in Section 4, this RGB representation is fed as input to a Convolutional Neural Network (CNN)-based policy.

Firehose additionally provides a simplified state space where the number of channels $C = 1$. In this representation, the value within each 1D feature vector for each cell is the discrete class of the given cell (-1 = harvested cell, 0 = vegetation, 1 = on fire, etc.). We use a flattened representation of this formulation as input to our Multi-layer Perceptron (MLP)-based Policy to allow for more efficient learning and a fairer comparison with our CNN model.

3.3 Action Space

Recall that an action in the Firehose framework is to **harvest** a cell. Harvesting a cell which is on fire will permanently extinguish it and prevent fire from spreading to it in the future. Similarly, harvesting a cell which is not on fire will prevent it from catching fire in the future.

At each time step, our agent selects a single cell to harvest with unit cost. However, note that the rate at which fire spreads could be faster than the rate at which we put it out. Hence, a naive strategy which harvests all the cells on fire would not suffice as the wildfire would continuously spread.

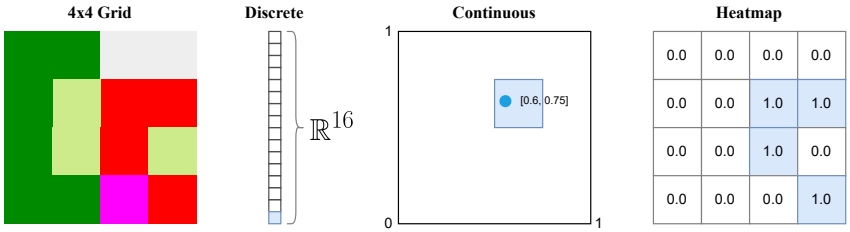


Figure 2: Visual representation of the action spaces for a toy 4×4 grid. The discrete action space is a flattened 1D vector of size 16 – the last blue element shows that we selected to harvest the cell in the bottom-right corner of the 4×4 grid. The continuous action space may be interpreted as a continuous box, where each continuous 2D vector $[x, y]$ may be mapped into a discrete cell in the grid. The heatmap action space outputs a policy directly in the 2D discrete representation of the environment, allowing us to potentially better exploit the spatial equivariance of the problem.

A toy example of the action spaces we discuss below is depicted in Figure 2 to aid in understanding.

Discrete Action Space. We flatten the 2D grid to a 1D vector of size $\mathbb{R}^{H \times W}$ using row-major ordering, where each element indexes a cell and its value specifies whether we should harvest it. This representation results in a high-dimensional action space (e.g., a 40×40 grid has 1600 discrete actions). Nevertheless, our experiments in Section 4.3 show that a discrete action space worked best.

Continuous Coordinate Action Space. We define an action as a 2D vector $[x, y]$ where $x \in [0, 1], y \in [0, 1]$. This represents the normalized x and y coordinates where we want to apply an action and defines a 'continuous box' as shown in Figure 2. `Firehose` discretizes the continuous vector to the closest cell before applying the action in Cell2Fire. The motivation here is to more compactly represent the action space and capture the spatial structure of actions. However, we found that this results in significantly worse performance in comparison to our discrete action space.

Heatmap-based Action Space. We maintain the grid formulation of an environment and represent the action space as a matrix of shape $H \times W \times 1$, where each element $\in [0, 1]$ represents the 'confidence' of whether we should harvest the given cell. We believe combining this heatmap representation with a fully convolutional network would best exploit the spatial equivariance of the state space. Unfortunately, we did not have time to complete our implementation of this representation as we could not easily integrate it with Stable Baselines 3, the library we use to train our RL algorithms.

3.4 Reward Function

An informative and accurate reward function is critical to learning a useful and generalizable policy. We use dense rewards as we are able to determine the number of cells on fire at each state.

The **SIMPLE reward function** for a state s is the proportion of cells on fire in the grid multiplied by -1. Hence, we apply a higher penalty for states in which more cells are on fire. This encourages the agent to minimize the number of cells on fire, reducing the spread and damage of the wildfire.

More formally, let $n = H \times W$ be the total number of cells in the environment and $F(\cdot)$ be a function that for a given state s, returns the set of discrete x and y-coordinates of the cells on fire. Now:

$$R_{\text{SIMPLE}}(s) = -c \cdot \frac{|F(s)|}{n}, \tag{1}$$

where c is a constant that allows us to scale our reward and $|F(s)|$ represents the number of cells on fire in state s. In our experiments, we used $c = 10$.

Reward Engineering. We attempted reward engineering and embedding additional terms that we believed would encourage the agent to choose better actions and learn more efficiently.

One particular reward we experimented with was adding a penalty if the agent selected an action that was more than 2 units away from any cell on fire in terms of L2 distance. This encourages the agent to select actions that are close to the existing fire, potentially making the exploration problem easier. We call this the DIST-PENALTY reward, which we define for a state s and action a as:

$$R_{\text{DIST-PENALTY}}(s, a) = R_{\text{SIMPLE}}(s) - \frac{\max(0, \min\{\|a - f\| - 2 \colon f \in F(s)\})}{d}, \tag{2}$$

where we assume $\|a-f\|$ computes the L2 distance between the action a and cell on fire f represented as their respective x and y-coordinates, and d is a normalizing term which we set in our experiments to the maximum distance between any two cells in the environment multiplied by 5. The $\max$ acts as an if statement to check whether the action was more than 2 units away from a cell on fire.

In practice, we found that training with the DIST-PENALTY reward did not result in any significant improvement or speed-up in learning as shown in Section 4.3. Ultimately, we found that it was difficult to specify a good reward beyond our SIMPLE reward as the dynamics of wildfires are very complex. We did not focus our time on tuning the reward function as we wanted to explore whether our RL agents were able to learn from scratch without prior engineering.

Privileged Reward. We also experimented with weighted reward functions which prioritize protecting certain areas of the map over others, an example of which is shown in Figure 7. Such a reward function could reflect real-world fire fighting and prevention objectives, such as protecting civilian areas and limiting ecosystem damage.

The PRIORITY reward is parameterized by a bitmap M of the same dimensions as the forest, where bits with high value indicate important regions of the map; and a priority hyperparameter α:

$$R_{\text{PRIORITY}}(s, M, \alpha) = \alpha R_{\text{SIMPLE}}(s) \odot M + (1-\alpha) R_{\text{SIMPLE}}(s) \odot (1-M) \tag{3}$$

where $\odot$ is element-wise multiplication. Our experiments set $\alpha = 1$ to have an exaggerated effect.

4 Experiments

Our experiments aim to demonstrate the ability of our RL agents in three task settings.

1. **Fixed Ignition Point:** for a given map and a fixed ignition point, we train and evaluate a RL agent to test its optimization ability.
2. **Random Ignition Points:** for a given map, we train and evaluate a RL agent on ignition points that vary between episodes at random. This test the ability of our agents to learn reactive policies which generalize between different initial states.
3. **Privileged Reward:** as discussed previously in Section 3.4, privileged rewards allow us to prioritize protecting certain areas of the map over others. In this task setting, we test our RL agents' ability to adapt to different rewards.

In the remainder of this section we will discuss the baselines we considered, our experimental setup, and then present our results for the task settings outlined above.

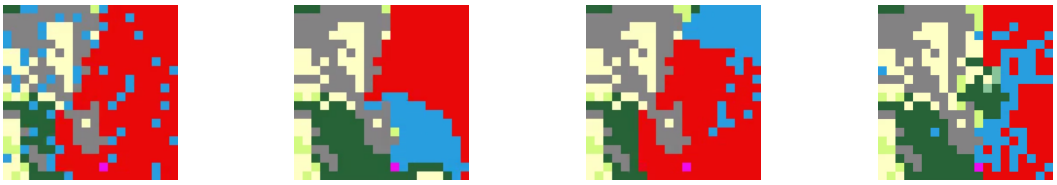
4.1 Baselines

Recall that the ignition point is where the wildfire begins spreading from. A good policy should select actions that prevent the fire from spreading further from the ignition point. We consider three baselines as depicted in Figure 3:

1. **Random**: randomly select any cell to harvest.
2. **Min-L2**: harvest the cell currently on fire which is closest to the ignition point in terms of L2 distance. This strategy is equivalent to actively suppressing the fire as it spreads.
3. **Max-L2**: harvest the cell currently on fire which is furthest away from the ignition point in terms of L2 distance. This strategy is equivalent to building a 'frontier' at the forefront of the fire to prevent it from spreading any further.

4.2 Experimental Setup

`Firehose` uses Stable Baselines 3 [Raffin et al., 2021] to simplify the training of our RL agents. We take advantage of vectorized environments to leverage parallel computing and run multiple `Firehose` simulations in parallel. We experimented with both policy gradient and Q-Learning algorithms including A2C, DQN, PPO, and a variant of PPO called Maskable PPO, which prevents repeated actions from being sampled during training and evaluation [Huang and Ontañón, 2020].



(a) Random (b) Min-L2 (c) Max-L2 (d) Learned Policy

Figure 3: Visualizations of the final state in a 20×20 experiment with the fixed ignition point shown in pink. This particular ignition point results in a significant spread of the fire which all approaches struggle to keep up with. The learned policy performs best and results in the least cells on fire.

Network Architecture. We consider CNN and MLP-based architectures and compare the performance of each approach. We use the RGB image state space as input to a CNN to extract latent features. The output is fed to a fully-connected (FC) layer to return the policy in our desired action space. We use the CNN architecture from Mnih et al. [2015]. For our MLP-based architecture, we fed the simplified state space with $C = 1$ channel to our MLP feature network with 2 layers with hidden sizes of 64. The output of this MLP is passed to a FC layer to output the policy.

Training. Unless otherwise specified, we trained all RL algorithms for 5 million steps. We experiment with different configurations of network architectures, discount factors γ, action spaces, and reward functions. We ran all our experiments on the MIT Supercloud [Reuther et al., 2018] on CPU-only nodes. We utilized the example 20×20 and 40×40 environments provided by Cell2Fire in addition to our custom built environment for the privileged reward task to train and evaluate our DRL agents.

4.3 Results

Videos of our learned policies are available at: `https://williamshen-nz.github.io/firehose`

Fixed Ignition Point. In this setting, we use the same location as the ignition point for the fire when an episode is reset. We select an ignition point which results in a rapid spread of the wildfire if it is not managed efficiently and effectively.

Our reward curves in Figure 4 demonstrate that we are able to train RL agents which quickly converge to efficient and well-optimized policies that outperform our handcrafted baselines. Our qualitative results on our website and in Figures 1 and 3 additionally illustrate that our RL agents exhibit emergent behavior, such as blockading the fire by harvesting choke points and building barriers around the fire using the topography and vegetation type.

Random Ignition Points. Here, we randomize the ignition point on each episode reset – hence, we test the feasibility of learning a reactive policy which generalizes to arbitrary ignition points. This is a more difficult task and requires the learned policy to localize the fire and take actions to restrict it.

We performed a study of the parameter space with discount factors $\gamma \in \{0.9, 0.95, 0.99\}$, CNN and MLP-based architectures, and RL algorithms. Our results in Figures 5 and 6 show that none of the RL algorithms are able to match the Min-L2 and Max-L2 baselines within 5 million steps. Interestingly, DQN performs worse than random – this could be due to a poor selection of hyperparameters.

Based on these experiments, we may establish that Maskable PPO with a CNN architecture and $\gamma = 0.9$ gives the best performance. We believe Maskable PPO improves the sample efficiency over vanilla PPO by increasing the likelihood of sampling trajectories that lead to a higher reward by 'masking' out repeated actions. We use this configuration in all our other experiments.

We trained this configuration in a 20×20 environment for 20 million steps, and were able to nearly match the Min-L2 and Max-L2 baselines as shown in Figure 4. The video of our learned policy demonstrates that it successfully generalizes between ignition points and exhibits emergent behavior by building perimeters around fires and closing choke points to prevent further spread. When we compare the agent trained using Maskable PPO to our baselines on a per-ignition point basis, we find that our agent outperforms the baselines and successfully contains the fire more often as shown in Table 1. In the cases where the learned policy did not contain the fire, the reward was much more negative because such cases led to out-of-distribution scenarios. Thus, with continued training, we believe we can further improve the learned policy.

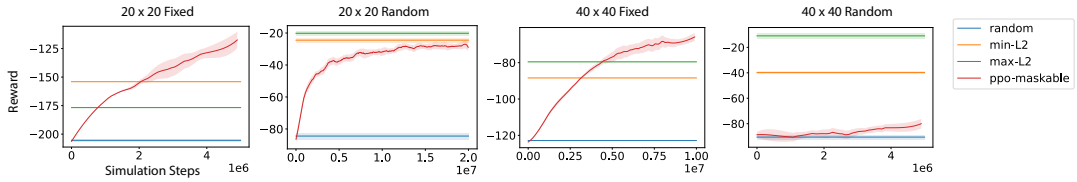


Figure 4: Reward Curves (mean and standard error) for our top-performing RL algorithm, Maskable PPO with the CNN architecture and $\gamma = 0.9$ over 3 seeds in a: (a) 20×20 map with a single fixed ignition point, (b) 20×20 map with random ignition points (20 million steps), (c) 40×40 map with a single fixed ignition point (10 million steps), and (d) 40×40 map with random ignition points.

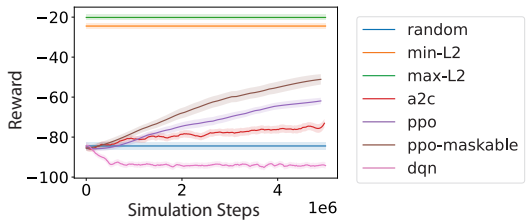


Figure 5: Reward curves (mean and standard error) for several RL algorithms we trained in comparison with our baselines on the random ignition point setting in a 20×20 map. We average the performance of the RL algorithms over all selections of γ and MLP/CNN architectures, and trained over 3 seeds. We evaluated the baselines over 1000 random ignition points.

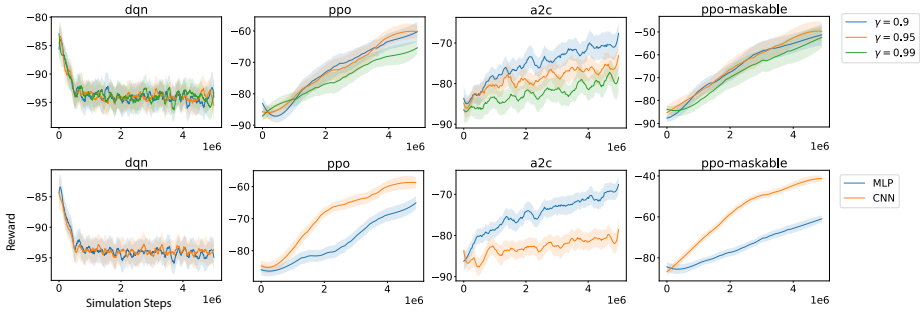


Figure 6: Results from a hyperparameter search over discount factors γ and network architectures.

	Random	Min-L2	Max-L2	Maskable PPO
% Trials Won	0.8%	16.4%	36%	46.8%

Table 1: Comparison of our baselines to the RL agent trained on 20 million steps in the random ignition point task in a 20×20 map. We sample 1000 ignition points, run trials for each approach and compare the rewards. The approach with the highest reward for each ignition point is given a 'win'. This demonstrates a more fine-grained analysis of the performance in comparison to reward curves.

Additional Experiments. We were unsuccessful in training a generalized policy for the 40×40 map as depicted in Figure 4. We believe that the high action space coupled with our limited reward engineering leads to sample inefficient training. We also compared our discrete and continuous coordinate-based action space, and found that the agent trained on the latter only achieved an average reward of -80 in contrast to -40. Furthermore, we observed that training on the DIST-PENALTY reward did not result in any speed-up in learning or substantial improvements in the performance of the learned policies. For the sake of brevity, we include these results in Appendix A.1.

Privileged Reward. Our results for this task are purely qualitative, and can be seen in Figure 7 where we train our RL agent to protect the regions specified by the letters "MIT". Our results demonstrate that Maskable PPO is capable of learning to protect privileged regions with random and fixed ignition points, and exhibits emergent behavior in building a wall around the privileged zones.

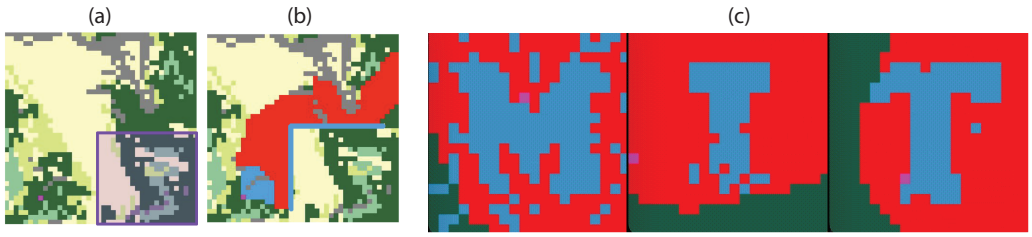


Figure 7: (a) Depicts an example of a privileged zone highlighted in purple and (b) shows an example optimal strategy for protecting the privileged zone. (c) depicts a frame from the execution of the learned policy on the MIT task, in which the letters are designated as privileged zones.

5 Unexpected Issues

Unfortunately, we ran into several issues when running the Cell2Fire simulator at the large scales we encounter in reinforcement learning, the most critical of which was the slow simulation speed of Cell2Fire. In certain environments, we were only able to achieve 2 steps per second on a single CPU core. We expand upon these issues in Appendix A.2. Although we were ultimately able to workaround many of these issues, our ability to quickly iterate and test our ideas were hindered.

6 Conclusion and Future Work

Thousands of decisions are made by first responders, firefighters, and experts on the ground during wildfires. Given the tense situation and limited time horizon, we may not expect the best decision to be made every single time. We have presented `Firehose`, a DRL framework for building and evaluating wildfire management agents in realistic environments which is driven by the recent Cell2Fire simulator. We hope that our work illustrates the utility of using learned controllers based on realistic simulations to guide decision making in the real-world.

We have demonstrated that by using `Firehose`, we are able to train policies which match and in certain situations outperform our hand-designed baselines, although the generalization performance of our reactive policies is sub-optimal. We are able to successful exploit the underlying spatial structure of the environment by using CNN-based policies and prevent undesirable actions using Maskable PPO. Additionally, by embedding additional information into the reward in terms of privileged zones, we are able to prioritize protection of important regions in an environment.

Our qualitative results indicate that our RL agents are able to learn emergent behaviors including building perimeters around fires, building walls to protect privileged zones, and harvesting dense vegetation at choke points to prevent the further spread of a wildfire.

Potential techniques to resolve existing limitations include improving the sample efficiency of training a RL algorithm – we could achieve this through utilizing off-policy or offline RL algorithms, or experimenting with the heatmap-based action space with a fully-convolutional network architecture as discussed in Section 3.3. To further improve the generalization performance of our learned policies, we may consider additional reward engineering, experimenting with more hyperparameters, and training our agent for longer periods of time.

Contributions

We both contributed equally to this project. William created the Gym environment with the corresponding state and action spaces, experimented with reward functions, implemented the evaluation script with video generation, integrated maskable PPO, and did a lot of debugging.

Aidan set up the Cell2Fire simulator and modified it to support consuming input actions, integrated Stable Baselines 3, created the training scripts, experimented with the privileged reward task and plotted the results. We both ran experiments on MIT's Supercloud and contributed equally to the writing of the proposal, presentation, mid-term and final report.

References

Alex Alexandridis, D Vakalis, Constantinos I Siettos, and George V Bafas. A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through spetses island in 1990. *Applied Mathematics and Computation*, 204(1):191–201, 2008.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Janice Coen. Some requirements for simulating wildland fire behavior using insight from coupled weatherwildland fire models. *Fire*, 1(1):6, 2018.

Shane R Coffield, Casey A Graff, Yang Chen, Padhraic Smyth, Efi Foufoula-Georgiou, and James T Randerson. Machine learning to predict final fire size at the time of ignition. *International journal of wildland fire*, 28(11):861–873, 2019.

Thomas Curt and Thibaut Fréjaville. Wildfire policy in mediterranean france: How far is it efficient and sustainable?: Wildfire policy in mediterranean france. *Risk Analysis*, 38, 07 2017. doi: 10.1111/risa.12855.

Pablo de Bem, Osmar de Carvalho Júnior, Eraldo Matricardi, Renato Guimarães, and Roberto Gomes. Predicting wildfire vulnerability using logistic regression and artificial neural networks: a case study in brazil. *International Journal of Wildland Fire*, 28, 01 2018. doi: 10.1071/WF18018.

Sriram Ganapathi Subramanian and Mark Crowley. Using spatial reinforcement learning to build forest wildfire dynamics models from satellite images. *Frontiers in ICT*, 5:6, 2018.

Ravi N Haksar and Mac Schwager. Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1067–1074. IEEE, 2018.

Jonathan L Hodges and Brian Y Lattimer. Wildland fire spread modeling using convolutional neural networks. *Fire technology*, 55(6):2115–2142, 2019.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

Piyush Jain, Sean CP Coogan, Sriram Ganapathi Subramanian, Mark Crowley, Steve Taylor, and Mike D Flannigan. A review of machine learning applications in wildfire science and management. *Environmental Reviews*, 28(4):478–505, 2020.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

M.J. Mayr, K.A. Vanselow, and C. Samimi. Fire regimes at the arid fringe: A 16-year remote sensing perspective (20002016) on the controls of fire activity in namibia from spatial predictive models. *Ecological Indicators*, 91:324–337, 2018. ISSN 1470-160X. doi: https://doi.org/10.1016/j.ecolind.2018.04.022. URL `https://www.sciencedirect.com/science/article/pii/S1470160X18302759`.

Sean McGregor, Rachel Houtman, Claire Montgomery, Ronald Metoyer, and Thomas G Dietterich. Fast optimization of wildfire suppression policies with smac. *arXiv preprint arXiv:1703.09391*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Cristobal Pais, Jaime Carrasco, David L. Martell, Andres Weintraub, and David L. Woodruff. Cell2fire: A cell-based forest fire growth model to support strategic landscape management planning. *Frontiers in Forests and Global Change*, 4, 2021. ISSN 2624-893X. doi: 10.3389/ffgc.2021.692706. URL `https://www.frontiersin.org/article/10.3389/ffgc.2021.692706`.

David Radke, Anna Hessler, and Dan Ellsworth. Firecast: Leveraging deep learning to predict wildfire spread. In *IJCAI*, pages 4575–4581, 2019.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

S. W. Taylor, Douglas G. Woolford, C. B. Dean, and David L. Martell. Wildfire Prediction to Inform Fire Management: Statistical Science Challenges. *Statistical Science*, 28(4):586 – 615, 2013. doi: 10.1214/13-STS451. URL `https://doi.org/10.1214/13-STS451`.

Stephan Zheng, Alexander Trott, Sunil Srinivasa, David C. Parkes, and Richard Socher. The ai economist: Taxation policy design via two-level deep multiagent reinforcement learning. *Science Advances*, 8(18):eabk2607, 2022. doi: 10.1126/sciadv.abk2607. URL `https://www.science.org/doi/abs/10.1126/sciadv.abk2607`.

A Appendix

A.1 Additional Results

Our additional results are presented in Figure 8 and Table 2.

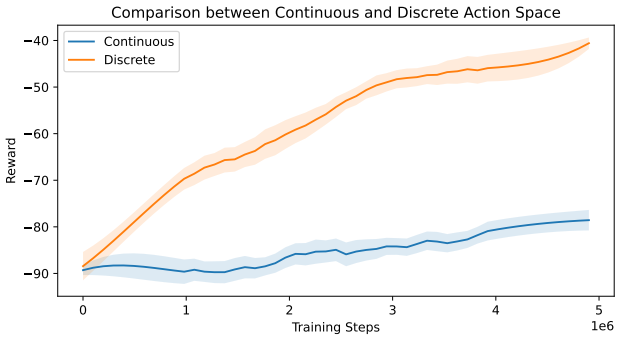


Figure 8: Reward curves for our comparison between the continuous and discrete action spaces described in Section 3.3. We use the default parameter configuration described in Section 4.3 to train our agents with the exception that vanilla PPO was used for the continuous action space, as Maskable PPO does not support continuous actions.

	Reward
Random	-82.16 ± 3.78
Min-L2	-22.76 ± 2.54
Max-L2	-20.44 ± 2.58
SIMPLE Reward	-36.67 ± 3.11
DIST-PENALTY Reward	-34.74 ± 2.75

Table 2: Mean reward and 95% confidence interval over 1000 random ignition points in the 20×20 environment for our baselines and the Maskable PPO algorithm trained on the SIMPLE and DIST-PENALTY rewards described in Section 3.4. Evidently, using the DIST-PENALTY reward only results in a minor improvement over the SIMPLE reward. We evaluate on the SIMPLE reward as it measures the proportion of cells on fire, the metric we want to ultimately target.

A.2 Issues Encountered

Unfortunately, we ran into several issues when running the Cell2Fire simulator at the large scales we encounter in reinforcement learning. We outline some of the more critical issues below.

Firstly, we needed to manually modify the Cell2Fire C++ code to support consuming input actions at each time step. This proved to be difficult, as the code is not well documented or structured, variables are left unused, etc. It was difficult to ascertain what exactly needed to be changed in order to support our action space. We also encountered sporadic issues with Cell2Fire throwing memory errors, segmentation faults, and not writing results to disk – we had to wrap around these issues in our Python code otherwise our entire training script would be interrupted.

Moreover, the simulation speed slows down as the size of the fire increases within a map because of the localized message-passing architecture of Cell2Fire. This made it unfeasible for us to scale beyond grids of size 40×40, as the simulation speed would slow down to less than 2 steps per second on a single CPU core.

Another issue we encountered was in relation to disk space. Cell2Fire writes a CSV file with the grid state after each simulation step has been executed – resulting in hundreds of small CSV files being written to disk for every episode. We found that we very quickly ran out of inodes on our hard drives which would break our experiments and slow down our computing environments. We overcame this issue by deleting these files as we progressed through the experiments – this inevitably slowed down our experiments as deleting a large number of files takes a non-trivial amount of time.

Although we were ultimately able to workaround many of these issues, our ability to quickly iterate and test our ideas were hindered by the issues aforementioned. The speed issues also meant that a single experiment on a 20×20 environment took around 240 CPU core-hours to run, which led to a slow feedback loop when experimenting. In total, we used almost two years of single CPU core-hours to run our experiments.