

PyTorchFire: A GPU-accelerated wildfire simulator with Differentiable Cellular Automata

Zeyu Xia^a, Sib0 Cheng^b,*

^a University of Virginia, Charlottesville, VA 22904, USA

^b CERE, ENPC, EDF R&D, Institut Polytechnique de Paris, Île-de-France, France

ARTICLE INFO

Keywords:

Wildfire simulation
Differentiable Cellular Automata
PyTorch-based software
Parallel computing techniques
GPU-acceleration

ABSTRACT

Accurate and rapid prediction of wildfire trends is crucial for effective management and mitigation. However, the stochastic nature of fire propagation poses significant challenges in developing reliable simulators. In this paper, we introduce PyTorchFire, an open-access, PyTorch-based software that leverages GPU acceleration. With our redesigned differentiable wildfire Cellular Automata (CA) model, we achieve millisecond-level computational efficiency, significantly outperforming traditional CPU-based wildfire simulators on real-world-scale fires at high resolution. Real-time parameter calibration is made possible through gradient descent on our model, aligning simulations closely with observed wildfire behavior both temporally and spatially, thereby enhancing the realism of the simulations. Our PyTorchFire simulator, combined with real-world environmental data, demonstrates superior generalizability compared to supervised learning surrogate models. Its ability to predict and calibrate wildfire behavior in real-time ensures accuracy, stability, and efficiency. PyTorchFire has the potential to revolutionize wildfire simulation, serving as a powerful tool for wildfire prediction and management.

Code availability

Name of software: PyTorchFire

Year first available: 2024

Hardware requirements: Preferred NVIDIA GPU

Software requirements: Python 3.8+, preferred 3.11

License: MIT License

Program language: Python 3

Homepage: <https://github.com/xiazeyu/PyTorchFire>

Source code: <https://doi.org/10.5281/zenodo.13132218>

PyPI package: <https://pypi.org/project/pytorchfire>

Document: <https://pytorchfire.readthedocs.io>

Dataset: <https://doi.org/10.17632/nx2wsksp9k.1>

1. Introduction

There has been an increasing number of devastating effects caused by human-induced climate change. One of its outcomes is the more frequent occurrence of wildfires, which are larger in both scale and duration worldwide (Xu et al., 2020). There are approximately 60,000 undesirable wildfire events (not intentionally set by humans for forest health and safety of nearby communities) with an average burned area of 7.02 million acres annually in the last decade just in the United States (National Interagency Fire Center, 2024). These fire events have not only put properties at high risk of being damaged by wildfires (Auer and Hexamer, 2022), but also pose a significant threat to human health (Chen et al., 2021).

Having an accurate and effective prediction, especially in the early stages of a fire event, is crucial for decision making in firefighting and evacuation strategies, as well as for short-term emergency response and long-term fire risk assessment (Hanson et al., 2000). Numerous studies have focused on predicting fire spread using various models, such as the semi-empirical model (Finney et al., 2023; Doyle et al.,

* Corresponding author.

E-mail address: sibo.cheng@enpc.fr (S. Cheng).

<https://doi.org/10.1016/j.envsoft.2025.106401>

Received 4 August 2024; Received in revised form 7 February 2025; Accepted 21 February 2025

Available online 7 March 2025

1364-8152/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

Main Notations	
Notation	Description
<i>Notations for Dataset</i>	
H, W	Height and width of data
V_w	Wind velocity
θ_w	Wind direction
θ_s	Slope to neighboring cells
p_{veg}	Scaling factor for vegetation type
p_{den}	Scaling factor for vegetation density
S_{init}	Initial fire state
$E, \hat{E}$	Altitude of a cell and its neighbors
l	Side length (resolution) of dataset
<i>Notations for Wildfire Cellular Automata</i>	
c_1	Scaling factor for wind velocity
c_2	Scaling factor for wind direction
a	Scaling factor for ground elevation
p_h	Base burning probability
$p_{continue}$	Probability of continued burning
p_w	Scaling factor for wind
p_s	Scaling factor for slope
c	Constant for normalization function
f_p	Probability-like normalization function
$p_{propagate,i,j}, p_{propagate,j}$	Probability for cell i to propagate fire to cell j
$p_{ignite,j}, p_{ignite}$	Probability of cell j becoming ignited
$S_{current}$	Current fire state
S_{next}	Next fire state
<i>Notations for Parameter Calibration Trainer</i>	
$r_{first}, r_{between}, r_{last}$	Constants for step rings attached to accumulator
t	Time step
it	Iteration step
$y, \hat{y}$	Observed fire region binary label and predicted ignition probability
$A, \hat{A}$	Observed and predicted fire region

2024), Computational Fluid Dynamics (CFD)-based models (Valero et al., 2021), data-driven models (Maffei and Menenti, 2019), and CA-based models (Hernández Encinas et al., 2007; Alexandridis et al., 2008; Papadopoulos and Pavlidou, 2011; Trucchia et al., 2020; López-De-Castro et al., 2024). However, these models typically utilize the CPU only, which is not fast enough to provide real-time predictions and cannot make predictions in a reasonable time when the fire scale is large or the resolution is high. Recent advances in neural networks (Guo et al., 2024; Xia et al., 2023; Gui et al., 2023; Zhong et al., 2021) have introduced a novel approach to the construction of wildfire prediction models. Numerous studies (Cheng et al., 2022a,b; Jain et al., 2020) have leveraged Machine Learning (ML) surrogate modeling to enhance the efficiency of wildfire simulations. Nevertheless, these methods exhibit significant limitations regarding generalizability. They are typically trained in specific ecoregions, rendering them applicable solely to those particular regions and unsuitable for deployment in diverse ecoregions with varying geophysical conditions. There has been recent research leveraging Message Passing Interface (MPI) for parallelization on CPUs (Li, 2018) and utilizing GPU acceleration (Hoang, 2008; Ntinis et al., 2017; Denham and Laneri, 2018) to enhance wildfire prediction. However, these implementations are frequently developed in C, Compute Unified Device Architecture (CUDA), or MATLAB programming language, posing challenges for integration into existing wildfire prediction systems due to specific system requirements or the necessity for commercial licenses. It may also cause trouble for the user to use user-provided datasets. In contrast, our proposed

approach aims to achieve portability such that, by merely altering the device string, the same software can be executed across different architectures, including CPUs and GPUs from various vendors.

Another significant challenge in achieving reliable wildfire prediction is parameter calibration or model correction. As summarized in Papadopoulos and Pavlidou (2011), a variety of algorithms (Alexandridis et al., 2008; Cheng et al., 2022b; Yassemi et al., 2008) have been designed to forecast the progress of regional fires based on local characteristics such as terrain, wind and fuel. However, due to the complexity and chaotic nature of wildfire systems, prediction models often lack accuracy, and the results heavily depend on the initial parameters (Alessandri et al., 2021). Parameter calibration is crucial for wildfire models to reduce prediction bias. Despite traditional brute-force methods (Alexandridis et al., 2008; Denham and Laneri, 2018), several studies focus on parameter identification using Genetic Algorithm (GA) (Denham and Laneri, 2018; Ntinis et al., 2017), variational data assimilation (Cheng et al., 2022a), or the ensemble Kalman filter (Zhang et al., 2019). However, these methods are either mathematically unstable, computationally expensive or require a large amount of data, making it challenging to perform parameter calibration on a large scale or in real-time. It is also worth mentioning that, due to the stochastic nature of wildfire modelings, different runs yield varying outcomes, complicating parameter calibration.

In this paper, we developed PyTorchFire, a PyTorch-based simulator designed to provide a fast and reliable wildfire simulation platform capable of self-correction using real-world observation data of wildfires. Our simulator incorporates a redesigned, Differentiable Cellular Automata (DCA) model that is efficient and supports real-time parameter calibration with excellent generalizability. With the support of PyTorch, we achieve high computing efficiency without compromising scalability and portability. GPU acceleration not only allows us to perform single simulation steps at the millisecond level but also enables large-scale simulations and higher resolutions that were previously challenging. The probability of arrival can be computed using an ensemble of numerous CA runs. However, tracking gradients from these ensembles is challenging, and parameter calibration can be computationally intensive. By averaging these runs, we can effectively leverage the model's stochastic nature. Our model can also conduct real-time parameter calibration using gradient descent, allowing it to self-correct with the latest real-world data.

We have first tested the prediction running time of PyTorchFire and MPI-CA (modified from (Li, 2018)). It turns out PyTorchFire significantly outperforms MPI-CA and scales well on GPUs. Then, we tested the parameter calibration performance on actual locations in California. We first calibrated fire parameters on 32 simulated fire events across three locations (Bear 2020, Brattain 2020, and Pier 2017), achieving excellent results compared to the targets. Finally, we performed parameter calibration using 2 recent real-world fire events (Bear fire in 2020 and Pier fire in 2017) in California. The results indicate that PyTorchFire can accurately mimic real fire events.

As our work functions similarly to traditional ML models, we will use some terms from ML to describe the behavior of our model. Specifically, we will use the terms *forward propagation* and *prediction* to describe the process of simulating the fire event. We will also use *backward propagation* and *training* to describe the process of calibrating the fire parameters.

Table 1 compares the capabilities of various well-known fire simulators. To the best of our knowledge, we are the first to apply Differentiable Cellular Automata (DCA) to wildfire simulation and to demonstrate its performance using real-world fire events.

The structure of the rest of this paper is outlined as follows: Section 2 presents a detailed description of our software design, including both wildfire DCA and algorithm for parameter calibration. The experiment design and results analysis on both simulated and real-world fire events are discussed in Section 3. The paper concludes in Section 4 with a synthesis of the key insights gathered and potential future work. The Appendix section provides additional information on the datasets, experiment environment, and learning rates used in this study.

Table 1
Comparison of various fire simulators' capabilities.

	GPU-acceleration	Generalizable	Differentiable	Real-time calibration	Test on real fire event
SimFire (Doyle et al., 2024)	✗	✓	✗	✗	✗
Flammap (Finney et al., 2023)	✗	✓	✗	✗	✓
MPI-CA (Li, 2018; Alexandridis et al., 2008; Trucchia et al., 2020)	✗	✓	✗	✗	✓
ML Models (Cheng et al., 2022b)	✓	✗	✓	✓	✓
FCA (Ntinis et al., 2017)	✓	✓	✗	✓(unstable)	✗
CUDAFires (Denham and Laneri, 2018)	✓	✓	✗	✓(unstable)	✗
PyTorchFire	✓	✓	✓	✓	✓

2. Methods

2.1. Inputs, outputs, and parameters

Our DCA model requires several tensors as input. If a specific tensor is missing, an empty tensor will be used instead. All required inputs, including environmental data and initial fire spread, are described in the following list, with H denoting the height from the rasterized ecoregion, and W denoting the width:

- **Wind impact:** There are two input tensors related to wind:
 - **Wind velocity** V_w : $H \times W$ float tensor specifying the average wind speed per cell, measured in m/s.
 - **Wind direction** θ_w : $H \times W$ float tensor indicating the average wind direction per cell, starting from East and progressing counterclockwise, measured in degrees.
- **Slope** θ_s : $H \times W \times 3 \times 3$ float tensor representing the slope from the current cell to its neighbors, measured in degrees.
- **Canopy** p_{veg} : $H \times W$ float tensor representing a probability scaling factor for the type of vegetation.
- **Density** p_{den} : $H \times W$ float tensor describing a probability scaling factor for the density of vegetation.
- **Initial fire** S_{init} : $H \times W$ boolean tensor specifying the initial burning state, serving as the starting condition for the simulation.

The slope θ_s in our model can be calculated from the altitude map using Eq. (1), where E and $\hat{E}$ are the altitudes of current cell and its neighbors, respectively, and l is the side length of the cell, often equal to the resolution of the dataset.

$$\theta_s = \arctan \left(\frac{E - \hat{E}}{k \cdot l} \right) \quad (1)$$

Here,

$$k = \begin{cases} 1 & \text{for adjacent cells} \\ \sqrt{2} & \text{for diagonal cells} \end{cases}$$

The output is a 2-channel boolean tensor of shape $2 \times H \times W$, representing the predicted burned area. The first channel indicates if a cell is burning, while the second channel indicates if a cell is burnt out. If the user is performing parameter calibration, the calibrated parameters will be included as additional output. Users can access an internal function to obtain a real-time ignition map p_{ignite} , where each element represents the probability that the corresponding cell is ignited.

In our model, there are five parameters that control the fire spreading process. All of them are floating-point numbers represented as tensors. During forward propagation, these parameters are treated as inputs only. During backward propagation, they are treated as initial values and are updated accordingly, except for $p_{continue}$. After all epochs, the optimal values with the best metrics are output as the final calibrated parameters.

The parameters we used are as follows:

- c_1 : The scaling factor for wind velocity.
- c_2 : The scaling factor for wind direction.
- a : The scaling factor for slope.
- p_h : The base probability that a fire propagates from a burning cell to an adjacent cell under normal conditions.
- $p_{continue}$: The probability that a burning cell continues to burn in the next time step.

The $p_{continue}$ parameter was introduced by Li's work (Li, 2018). We adopted this parameter to enable adjustment to the shape of the fire boundary. Users can disable this feature by setting $p_{continue} = 0$.

Depending on the dataset type, end-users may need to perform preprocessing to ensure compatibility with our code. Utilities for common conversions are provided with our code. Our program is designed to be modular and flexible with map resolutions, enabling users to easily build subclasses on top of it. This modularity allows users to run simulations on their specific datasets or implement revised algorithms with ease.

2.2. Differentiable wildfire cellular automata

CA, first popularized through Conway's Game of Life (Bak et al., 1989), have proven to be effective tools for simulating complex stochastic processes using simple transition rules. In PyTorchFire, we build on the work of Alexandridis et al. (2008) to create a DCA model for wildfire spread prediction.

State and transition rules. The model is based on a 2D grid of cells, each representing a square unit of geological area with one of three states: burnable, burning, or burned. As shown in Fig. 1, with the model being repeatedly updated in discrete time steps, each cell's state $S_{current}$ at time step t will be updated to state S_{next} depending on the states of its neighbors at the previous time step $t - 1$. The transition rules are as follows:

- A cell will have a default state of burnable.
- Initial ignition cells are set to burning.

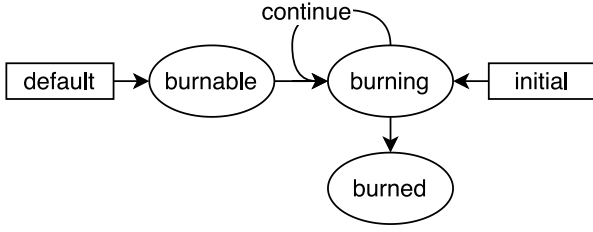


Fig. 1. State transition diagram illustrating dynamic changes between wildfire states.

- A burnable cell will ignite with probability p_{ignite} if at least one of its neighbors is burning.
- A burning cell will remain burning with probability p_{continue} ; otherwise, it will burn out and become burned.
- A burned cell will always remain burned.

This state and transition design remains clear yet effective and can be easily implemented using a tensor with two boolean channels: burning and burned, which requires only 2 bits of space to store per cell. The original work (Alexandridis et al., 2008) include a state of ‘no forest fuel’, which is considered redundant in our proposed model. This state can be represented by assigning the corresponding area a very small p_{veg} , resulting in a fire propagation probability of zero.

Ignition probability calculation. To make the CA model differentiable, we need to replace the stochastic ignition probability calculation with a differentiable function (Martin, 2017; Mordvintsev et al., 2020).

The original CA model (Alexandridis et al., 2008) provides a probability $p_{\text{propagate}}$ denoting a cell’s likelihood to propagate fire to its neighbors, calculated as a product of various scaling factors, given by:

$$p_{\text{propagate}} = p_h(1 + p_{\text{veg}})(1 + p_{\text{den}})p_w p_s \quad (2)$$

Here, p_h is the base probability of a cell igniting, p_{veg} is the vegetation factor, p_{den} is the density factor, p_w is the wind factor, and p_s is the slope factor.

Specifically, p_w can be calculated by:

$$p_w = \exp(c_1 V_w) \exp(c_2 V_w (\cos(\theta_w) - 1)) \quad (3)$$

where θ_w is the wind direction towards its neighbors, and V_w is the wind velocity. The wind velocity constant c_1 and wind direction constant c_2 are derived from experimental data.

The scaling factor for slope p_s can be calculated by:

$$p_s = \exp(a\theta_s) \quad (4)$$

where a is a constant for the slope factor derived from experimental data, and θ_s is the slope angle in degrees.

To apply non-local processes such as spotting fire (López-De-Castro et al., 2024), we propose performing in-place operations after each simulation step. Although these fire points ignited by the spotting effect cannot be directly tracked on the computation map, they can be treated as initial ignition points. The remaining pipeline can then seamlessly execute fire prediction and parameter calibration without any modifications.

As the product of scaling factors in Eq. (2) may exceed 1, causing issues when applying transformations as probabilities, we propose a probability-like normalization function f_p to normalize the propagation probability $p_{\text{propagate}}$, defined as:

$$f_p(x) = \tanh(c \cdot x) \quad (5)$$

where c is a hyperparameter.

The function is the optimal one selected from several candidates, where c is a parameter that influences the behavior of the functions:

$$f(x) = -c^{-x} + 1, \quad (6)$$

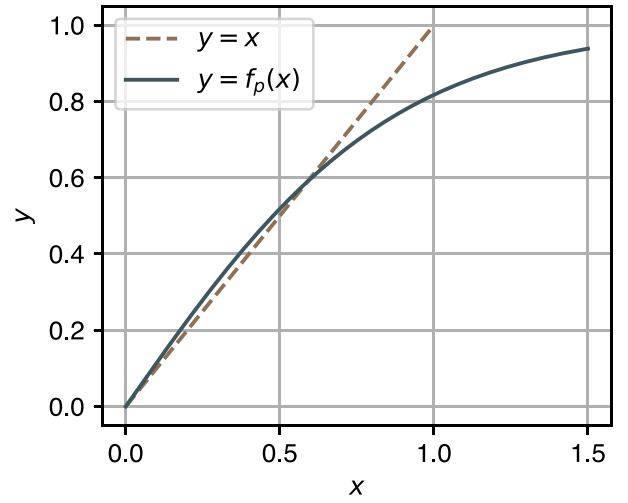


Fig. 2. Comparison of our probability-like normalization function $y = f_p(x)$ with $y = x$, illustrating its effectiveness in maintaining valid probabilities within $[0, 1]$ and smoothly adjusting values outside this range.

$$f(x) = -(x + 1)^{-c} + 1, \quad (7)$$

$$f(x) = \tanh(cx). \quad (8)$$

We aim to find the best differentiable function that preserves most valid values and can adjust invalid values to the valid range $[0, 1]$. Given that $p_{\text{propagate}}$ produces non-negative output, we focus on correcting the positive invalid input. Using the Nelder–Mead algorithm, we fitted these functions within the range $[0.2, 0.8]$ to determine the optimal value of c for each function, and compared the sum of squared differences in this interval. Ultimately, we selected the optimal function with the minimal sum of squared differences from Eq. (8) where $c = 1.1486328125$.

As shown in Fig. 2, our function is nearly linear in the range of $[0, 0.7]$, and as x approaches positive infinity, f_p approaches 1, which perfectly meets our expectations.

Since Eq. (2) only provides the propagation probabilities $p_{\text{propagate}}$, it is challenging to apply gradient descent directly. Introducing a per-cell ignition probability p_{ignite} is highly beneficial, as it can be calculated by aggregating $p_{\text{propagate}}$ from a cell’s Moore neighbors. As illustrated in Fig. 3, given that the fire model assumes all ignition events from the eight neighbors are independent and identically distributed, and our normalized propagation probabilities $f_p(p_{\text{propagate}})$ now represent true probabilities, we can apply the inclusion-exclusion principle to determine the probability of a cell being ignited by its neighboring cells. The formula is given by:

$$p_{\text{ignite},j} = 1 - \prod_{i=1}^8 (1 - p_{\text{propagate},i,j}) \quad (9)$$

where $p_{\text{ignite},j}$ is the probability of cell j being ignited, i represents cells adjacent to the current cell, and $p_{\text{propagate},i,j}$ is the probability of cell j being ignited by its neighbor i .

Data flow

Fig. 4 illustrates the data flow for a time step computation of our DCA in PyTorchFire. By following this data flow, the model state is updated from the current state S_{current} to a new state S_{next} . The data flow direction varies depending on whether it is performing a prediction task or a parameter calibration task. Environmental data and parameters are used to calculate $p_{\text{propagate}}$ for each cell. This value is processed through our probability-like normalization function f_p to ensure it is a valid probability for subsequent steps. The $f_p(p_{\text{propagate}})$ values of burning cells are then selected for a reduction operation to

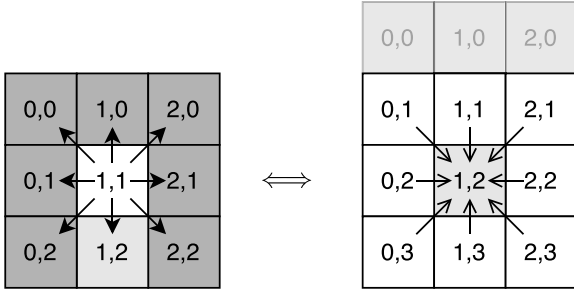


Fig. 3. Propagation of $p_{\text{propagate}}$ to neighboring cells in a Moore neighborhood and the application of the inclusion-exclusion principle to determine p_{ignite} .

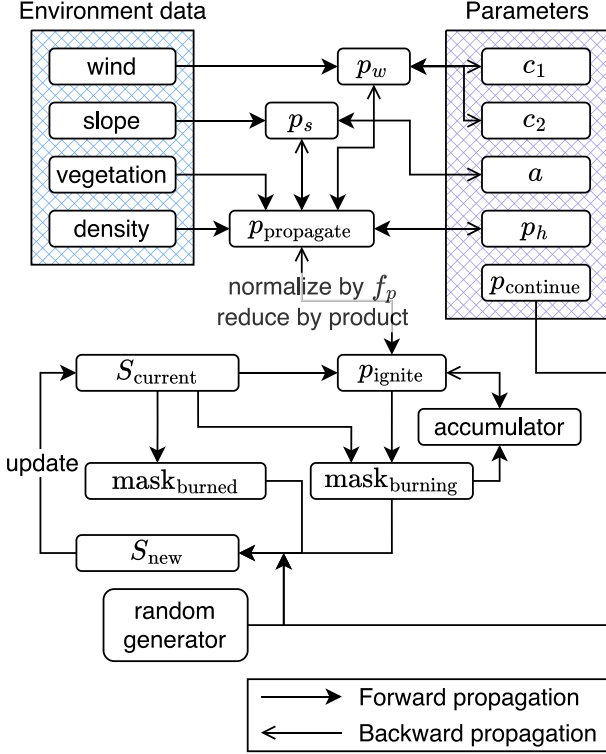


Fig. 4. Flow chart depicting data flow and dependencies in PyTorchFire, emphasizing connections between data, parameters, and outputs.

determine the burning probability p_{ignite} for each cell. These probabilities are compared against two random matrices drawn from a uniform distribution, and based on the comparison results, the cell states are updated from S_{current} to S_{next} . Finally, the p_{ignite} values of newly ignited cells are accumulated for use in the back-propagation process during the parameter calibration task. Details about parameter calibration will be discussed in the following sections.

2.3. Parameter calibration on the fly

Parameter calibration based on real-time satellite observations is often necessary for wildlife predictive models to adjust parameters for more accurate future predictions. This process enhances the model's ability to predict affected areas more precisely, thereby improving its usability.

Introduction to DCA. The concept of performing gradient descent on DCA originates from Martin's preprint (Martin, 2017). However, Martin's work focused on a single parameter and was applied in a one-dimensional context. In our research, we extended this approach

to a two-dimensional space and experimented with real-world wildfire scenarios. Instead of using a simplistic fire transition model, we enabled gradient computation on a more complex model that incorporates real-world landscapes, dynamic wind conditions, and other critical factors.

Back-propagation algorithm. Performing parameter calibration for a stochastic DCA necessitates substantial effort to stabilize the calibration process. To address this, we developed a comprehensive algorithm based on the widely-used gradient descent algorithm, as illustrated in Algorithm 1. The algorithm incorporates three nested loops: epoch step, iteration step it , and time step t .

The outermost epoch loop facilitates multiple complete calibrations using the entire dataset. This loop ensures that the model undergoes extensive training across several iterations, enhancing the robustness of the calibration process.

Within the epoch loop, the iteration loop processes individual data points from the dataset. Given that real-world data is often obtained in a discrete manner, this loop calibrates the parameters using all available observations, thereby achieving a temporal fire similarity.

The innermost step loop is responsible for simulating the fire using the most recently calibrated parameters each time they are updated. This approach accelerates the training process by continuously refining the parameters.

To ensure stability in the training process, we employ techniques such as using the same seed for each epoch. Additionally, after each calibration, we clip the parameters to a proper range, maintaining numerical stability.

In terms of optimizer selection, we found that AdamW outperforms SGD in convergence speed and stability. Therefore, we selected AdamW as our optimizer. For learning rate selection, we tested values ranging from 1 to 10^{-4} . We found that learning rates larger than 10^{-1} do not converge, while learning rates smaller than 10^{-3} converge extremely slowly. Consequently, we chose a learning rate of 5×10^{-3} for the AdamW optimizer. As the optimal learning rate varies depending on the data, we recommend users to try multiple values and use the best one.

Loss function design. Considering that our model performs parameter calibration by minimizing the difference between simulated fire spread and observed fire spread, we need to design an appropriate loss function to measure the difference between the two in terms of shape and overall scale. We propose the following loss function to quantify the difference between the simulated and observed fire spread:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N \left(-[y_i \cdot \log(\sigma(\hat{y}_i)) + (1 - y_i) \cdot \log(1 - \sigma(\hat{y}_i))] + (\text{AvgPool2d}(y_i) - \text{AvgPool2d}(\hat{y}_i))^2 \right) \quad (10)$$

where $\sigma(x) = \frac{1}{1 + \exp(-x)}$, y denote the binary label of the observed fire region and $\hat{y}$ represents the probability of predicted ignition.

The loss function is designed to be differentiable, making it suitable for the optimization process. It comprises two parts: the Binary Cross Entropy (BCE) with logits loss, which measures the difference in fire spread shapes as a binary classification task, and the Mean Squared Error (MSE) loss after applying a 2D average pooling, which measures the difference in fire scale as a regression task. Our trials indicate that the BCE component effectively reconstructs the fire shape, while the MSE component accurately mimics the fire scale by approximating the number of affected cells.

To accelerate BCE loss computation, we first crop the region to the union of non-zero predictions and targets, then apply the BCE with logits loss to this region. For the MSE loss, we use average pooling with a window size of 4 and a stride of 4, allowing us to compute the expectation in each region and subsequently measure the MSE from these averages. Through experiments, we found that larger window

Data: wind V_w, θ_w , slope θ_s , vegetation p_{veg}, p_{den} , initial fire S_{init} , observed fire targets

Input: maximum epochs max_epochs , update interval $steps_update_interval$, initial constants $c_1, c_2, a, p_h, p_{continue}$

Output: calibrated model parameters c_1, c_2, a, p_h

```

1 Initialize wildfire model model( $V_w, \theta_w, \theta_s, p_{veg}, p_{den}, S_{init}, c_1, c_2, a, p_h, p_{continue}$ )
2 Initialize optimizer optimizer()
  // Set max_iterations to the number of available data points in the dataset
3 max_iterations ← data.count()
4 for epoch ← 1 to max_epochs do
  // Reset the model to the initial state, and get a new random seed
5  model.reset()
  // Save the seed to stabilize training in current epoch
6  epoch_seed ← model.seed
7  for iteration ← 1 to max_iterations do
8    iter_max_steps ← iteration * steps_update_interval
9    for step ← 1 to iter_max_steps do
10     // Update wind if needed
11     model.wind ← data.wind[iteration]
12     model.compute(attach=check_if_attach(step, iter_max_steps))
13   end
14   outputs ← model.accumulator
15   targets ← data.targets[iteration]
16   loss ← criterion(outputs, targets)
17   // Compute gradients
18   loss.backward()
19   // Update weights
20   optimizer.step()
21   // Reset gradients
22   optimizer.zero_grad()
23   with torch.no_grad()
24     // Clip the parameters to avoid numerical instability
25     model.a.clamp(min=0.0, max=1.0)
26     model.c_1.clamp(min=0.0, max=1.0)
27     model.c_2.clamp(min=0.0, max=1.0)
28     model.p_h.clamp(min=0.2, max=1.0)
29   // Compute from beginning using new parameters
30   model.reset(seed=epoch_seed)
31 end
32 end

```

Algorithm 1: Parameter calibration using gradient descent

sizes result in smaller absolute MSE values. While smaller MSE values tend to improve spatial similarity, they also increase the potential for overfitting, given the stochastic nature of the simulator.

Step rings for back-propagation. When performing parameter calibration, gradients can only be computed from ignited cells. As shown in Eq. (9), only ignited cells have $p_{propagate}$ contributing to p_{ignite} , so the chain rule can only be used to compute gradients of parameters from these cells. This presents a challenge: tracking gradients for all ignited cells requires substantial memory, which can slow down gradient back-propagation and potentially lead to out-of-memory errors. Conversely, tracking too few cells can result in an unevenly calibrated landscape, leading to poor performance due to underestimation or overestimation.

To address this issue, we proposed a novel update technique called ‘update by step rings’. When forming the computation graph, we always recompute the entire simulation from the beginning using the latest parameters. During this process, we determine which steps to attach to the computation graph. This is achieved by selecting the cells ignited from the first r_{first} steps, last r_{last} steps, and $r_{between}$ steps distributed evenly in between. As shown in Fig. 5, this results in a selection of multiple cells in the shape of rings, representing ignited cells from different steps. These cells are then added to an accumulator. For those cells that do not need to be connected with the computation graph, they are treated as constants and added to the accumulator as well for loss computation. This approach balances memory usage and gradient computation efficiency despite the actual state of the fire.

Through trials, we observed that increasing r_{last} yields better results. Increasing $r_{between}$ also improves performance, but not as significantly as r_{last} . Increasing r_{first} only has a slight impact on calibration results.

Software description.

To enhance accessibility, we have published our package on Python Package Index (PyPI). Users can easily install the package by running `pip install pytorchfire`. A Quick Start guide and comprehensive documentation are available online at <https://pytorchfire.readthedocs.io/>.

Our package relies on several essential libraries. The code is primarily based on PyTorch (Ansel et al., 2024), with version 2.0.0 or higher being preferred to ensure optimal performance. Einops (Rogozhnikov, 2021) is employed to facilitate readable tensor operations. In our experiments, we used h5py (Collette et al., 2024) for dataset management. We recommend running our code on either CUDA or CPU platforms.

3. Numerical results

In this section, as a proof of concept, we present and analyze the numerical results from PyTorchFire predictions, calibrated predictions using simulated data, and satellite observations.

Experiment setup and study areas. To evaluate the prediction performance, we assess the running time (after warm-up) of 300 time steps predictions on a simulated geographical environment. For parameter

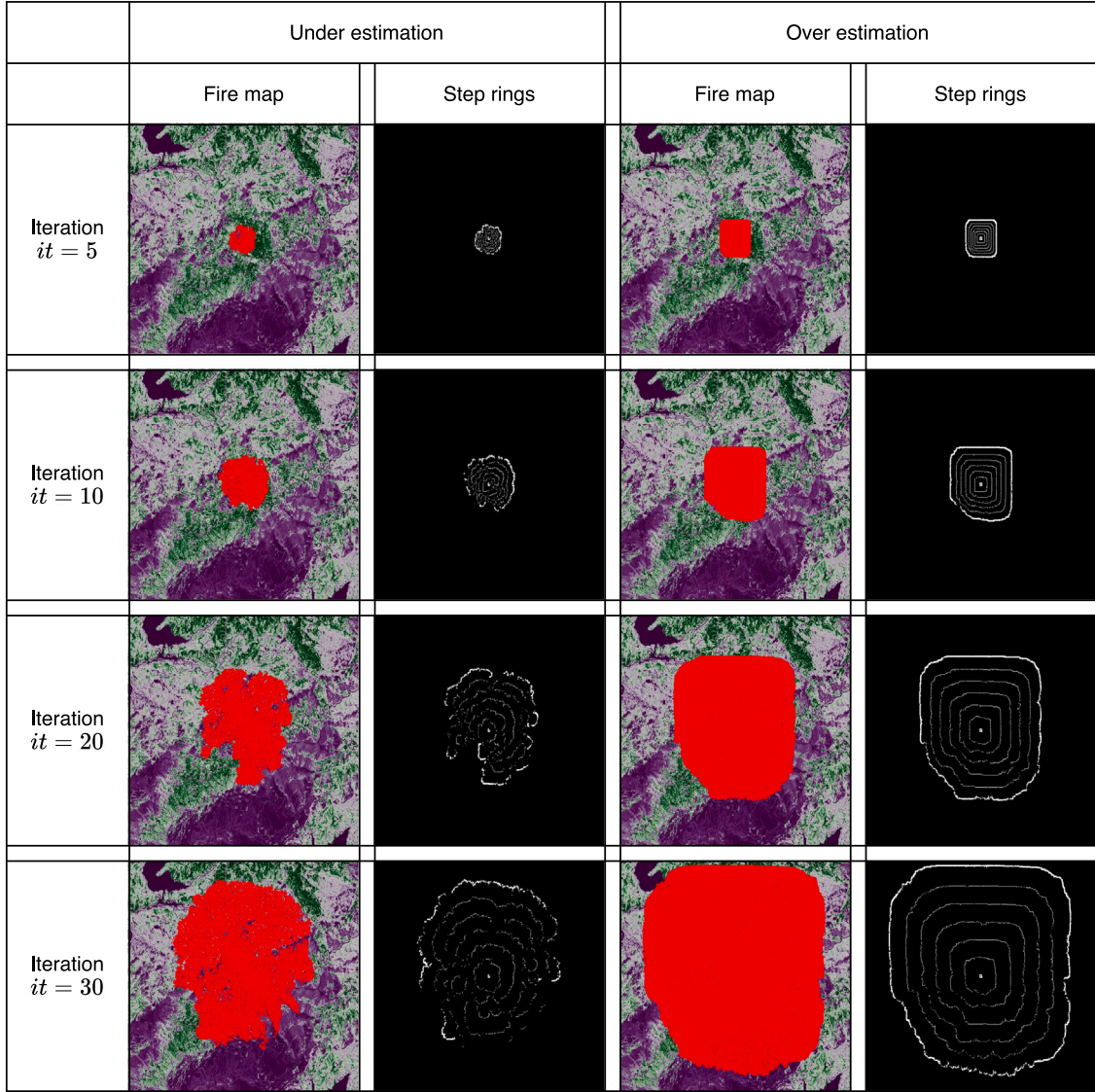


Fig. 5. Selected rings of cells for back-propagation, illustrating the balance between memory usage and diversity cell coverage.

calibration performance, we selected two metrics: Jaccard Index (also known as Intersection over Union (IoU)) and Manhattan Distance.

The Jaccard Index measures the spatial similarity between prediction and observation. It is defined as the size of the intersection divided by the size of the union of the sample sets, as shown in Eq. (11).

$$\text{Jaccard Index} = \frac{|A \cap \hat{A}|}{|A \cup \hat{A}|} \quad (11)$$

where A is the simulated or observed reference of the burned area (i.e., ground truth) and $\hat{A}$ denotes the predicted burned area. This metric indicates how well the prediction overlaps with the ground truth.

The Manhattan Distance, also known as the L1 norm, measures the absolute temporal difference in fire spread and is defined as follows:

$$\text{Manhattan Distance} = \sum_{t=1}^n |A_t - \hat{A}_t| \quad (12)$$

where $|A_t|$ is the number of observed cells on fire at time t and $|\hat{A}_t|$ is number of affected cells in prediction at time t . This metric indicates how well the prediction matches the ground truth in terms of fire growth and size.

Since the CA is stochastic in nature, the same set of parameters can lead to different results. To mitigate this, we obtain target metrics by

repeating the simulation of target fires 5 times, and report the average and deviation of the different runs. If, after parameter calibration, our model achieves metrics comparable to the average metrics of the target fire, we consider our parameter calibration a success.

For the real wildfire data, since we cannot repeat the fire nor know if the data reflects the time when it was burning. We report only the metrics of our best run. We consider it a success if the fire scale is quantitatively similar to the real fire.

We used a spatial resolution of 30 meters for the side length of a cell, that is the highest resolution from our dataset. Our parameter calibration experiments are conducted using environmental data and fire events located in California, as shown in Table 2. Each fire ecoregion has its unique features. The ecoregion of the 2020 Bear fire is characterized by mountainous terrain, a canyon traversing the map, and an area devoid of vegetation. The ecoregion of the 2020 Brattain fire is expansive, with high altitude and gentle slopes. It features Summer Lake in the northeast, a hill in the west, and overall sparse vegetation density. The 2017 Pier fire ecoregion consists of mountainous terrain with several mountains and hills, relatively steep slopes, and most areas covered with dense vegetation.

Forward prediction with simulated wildfires. To evaluate the performance of PyTorchFire, we compared it with another CPU

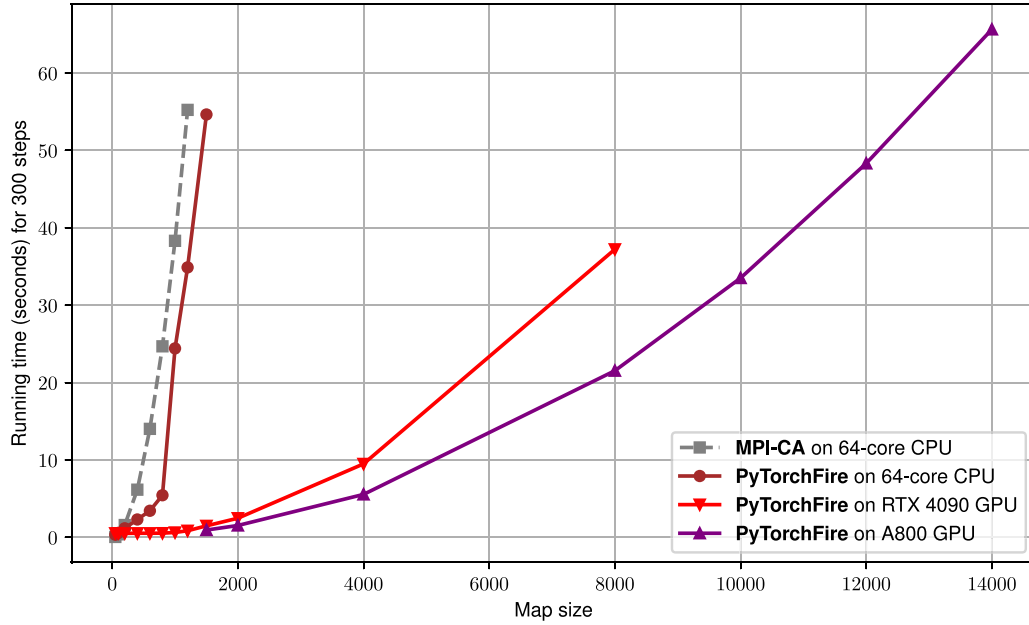


Fig. 6. Prediction time (in seconds) for completing 300 steps of simulation across various map sizes ($50 \times 14,000$ to $14,000 \times 14,000$). The performance of PyTorch on both CPU and GPU outperforms MPI-CA, demonstrating high efficiency and scalability.

Table 2

Overview of selected wildfire datasets, including coordinates, dates, and areas.

Dataset name	Latitude	Longitude	Start date	Duration	Area
Bear 2020	39.8173	-121.0893	2020-08-19	23 days	$\approx 107 \text{ km}^2$
Brattain 2020	42.6179	-120.5761	2020-09-07	22 days	$\approx 458 \text{ km}^2$
Pier 2017	36.1226	-118.7074	2017-08-29	31 days	$\approx 250 \text{ km}^2$

parallel simulator, MPI-CA (Li, 2018), a wildfire CA implemented in Python and parallelized using MPI techniques. The map in this experiment is a square plain landscape, with fixed wind to mitigate the impact of data loading. We conducted a series of experiments with simulated wildfires running 300 time steps across various map sizes and devices. The results are presented in Fig. 6, with the map size representing the side length of the simulation space. Both MPI-CA and PyTorchFire complete the simulation of size 200 in 2 s on a CPU. However, as the map size increases, there is a rapid escalation in runtime for both simulators. When the map size reaches 1000, which is reasonable after adding buffering areas, both simulators require approximately 30 s to complete. We also tested the maximum map size, with MPI-CA handling 1200 and PyTorchFire handling 1500 in one minute.

When utilizing a GPU, the performance potential of our solution is fully realized. PyTorchFire can complete 300 time steps of simulation for a map size under 1200 in less than 1 s on an RTX 4090 GPU. With an A800 GPU, the simulator can handle 300 time steps for a map size up to 1500 in less than 1 s, proving the simulator's high efficiency and scalability.

For all map sizes under 800 on an RTX 4090, we observed a consistent runtime of 0.52 s, suggesting that the GPU may not be fully utilized for smaller map sizes. We also explored the limits of our simulator. The RTX 4090 can support simulations on map sizes up to 8000. With a more advanced GPU, PyTorchFire can handle simulations of map sizes up to 14000, completing the entire 300-step simulation in less than a minute on a single GPU system.

These results demonstrate that PyTorchFire is highly efficient and scalable, capable of handling large-scale wildfire simulations. This offers significant research potential for developing large-scale wildfire simulations and providing real-time decision support.

Inverse modeling with simulated wildfires. To test the effectiveness of parameter calibration via gradient descent across different scenarios using PyTorchFire, we selected three maps: “Bear 2020”, “Brattain 2020”, and “Pier 2017” for wildfire simulations. All experiments utilized random p_{continue} values ranging from [0.4, 0.6], with wind updates at fixed intervals.

To conduct a fundamental test of inverse modeling for both initial parameter underestimates and overestimates, we will demonstrate the performance on two fire ecoregions (maps): *Bear 2020* and *Brattain 2020*. In each map, one simulation time step t is equivalent to 1.6 h in the real world.

To generate a target fire for inversion, initial ignition points of 3×3 grid cells and random parameters were manually set. We first ran a fire simulation as the *target* fire, then repeated the same simulation five times to mitigate the randomness of the CA simulation. The average and deviation of metrics were recorded as the *metrics of target* fire, which also served as the goal for parameter calibration. Using the initial parameter set ($p_h = 0.15$ or 0.8 , with other parameters set to 0), we performed the uncalibrated simulation, recorded as *initial*. Finally, we calibrated the model using the target fire and used the calibrated parameters to perform the simulation, recorded as *calibrated*.

When illustrating the simulated and observed burned area, vegetation and slope were mixed and added to the figure as background, where purple indicates low vegetation or gentle slopes, and green indicates dense vegetation or steep slopes. Time slices were selected in the figure to observe how the simulation changes at different time steps with parameters and how metrics change with calibration. The dynamics of affected cell count was shown to assess the temporal performance of the calibration, and the Jaccard Index was shown to evaluate the stability of the accuracy in different iterations.

As shown in Fig. 7, in the ecoregion of “Bear 2020”, the target grows at a relatively fast speed and gradually forms a shape. However, this shape may not be stable, as the Jaccard Index between the target and other simulations (with same ignition cells) is only 82.5%. The uncalibrated simulation significantly differs from the target, with Jaccard Indices of 17% and 37%, and a Manhattan Distance that is 20 to 40 times larger than the target. After parameter calibration, the simulation aligns with the target fire scale, showing noticeable improvements in both metrics. The metrics for the underestimation case are closer to the target, and the overestimation case even falls within the target range,

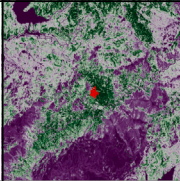
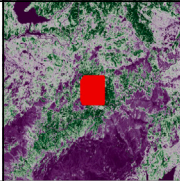
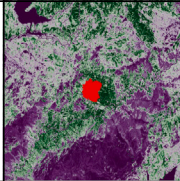
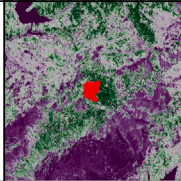
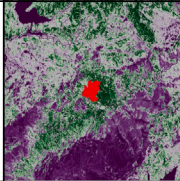
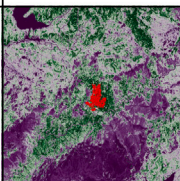
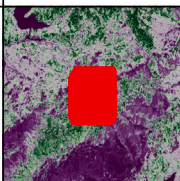
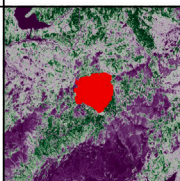
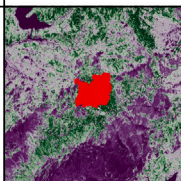
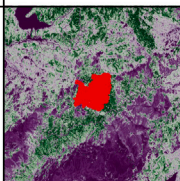
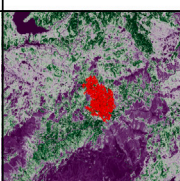
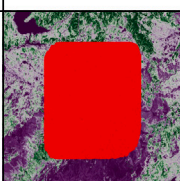
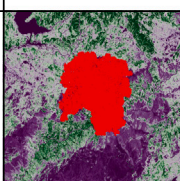
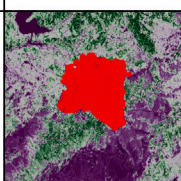
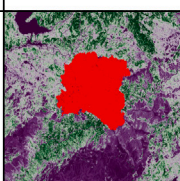
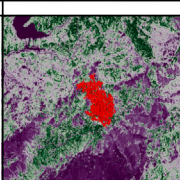
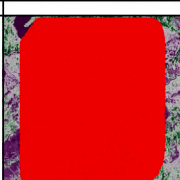
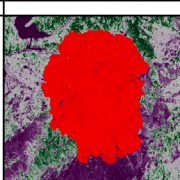
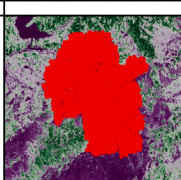
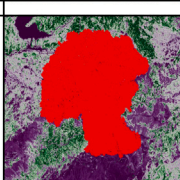
	Under estimation $p_h = 0.15$ (initial)	Over estimation $p_h = 0.8$ (initial)	Under estimation $p_h = 0.15$ (calibrated)	Over estimation $p_h = 0.8$ (calibrated)	Target
Time step $t = 50$					
Time step $t = 100$					
Time step $t = 200$					
Time step $t = 300$					
Jaccard index	0.174	0.379	0.792	0.864	0.825 ± 0.04
Manhattan distance ($\times 10^4$)	113.33	236.17	19.27	4.65	5.87 ± 2.63

Fig. 7. Comparison of wildfire simulation results at various time steps in the Bear 2020 fire ecoregion. Two initial parameter sets (under-estimation and over-estimation) are used. Post-calibration, the simulations closely align with the target fire scale, demonstrating effective recovery despite the inherent randomness of CA.

exhibiting a similar unique shape of fire. From Fig. 8, it is evident that the calibrated model's dynamics of affected cell count aligns with the target, and its Jaccard Index remains stable at 0.8 across iterations, both demonstrating significant improvements over the uncalibrated model.

A similar results can be seen in the “Brattain 2020” map. As shown in Fig. 9, the target grows slowly, and have a strong spread direction. The target shape is even more unstable with worse metrics (e.g., an average Jaccard Index of 78.4%) between repeated target runs. The uncalibrated simulation is significantly worse in visualization and metrics, the uncalibrated fire for under estimation case did not spread at all, with a Jaccard Index of 1%, and the over estimation case is burning without considering the low density of the vegetation. After calibration, the Jaccard Index has improved from 37% to 79%, and the Manhattan Distance has reduced by several orders of magnitude, falls into the range of target. Fig. 10 shows that the calibrated model has its dynamics of affected cell count aligns with the target, and its Jaccard Index is stable after several initial iterations, both showing great improvement than the uncalibrated one.

To further experiment with our parameter calibration method's effectiveness across different maps and ignition points, we conducted a comparative analysis of wildfire parameter calibration results using multiple simulated datasets. Each dataset was evaluated under two pre-generated fire condition scenarios and five distinct initial ignition points, with the parameters set as: $a = 0.15$, $c_1 = 0.15$, $c_2 = 0.15$. The parameter p_{continue} was randomly selected within the range $[0.4, 0.6]$,

and the parameter p_h was manually set for each map. Specifically, p_h values were 0.25 and 0.5 for Bear 2020, 0.5 and 0.6 for Brattain 2020, and 0.45 and 0.6 for Pier 2017.

The comparison of calibrated and uncalibrated fire simulations against the targets (reference simulations) is presented in Table 3. A noticeable improvement in both metrics can be observed across all datasets except for Bear 2020 with a small fire scale. This demonstrates that our method effectively calibrates parameters in most cases. Notably, in the Bear 2020 small-scale fire case, the target itself has a low Jaccard Index of 56%, compared to approximately 70% in other cases. This indicates that the small fire on this map is highly random, making it challenging to reliably predict the fire using the current CA model. This highlights a limitation of the current fire model equations (Alexandridis et al., 2008), suggesting that an improved fire model is needed to enhance performance in this scenario. Although the running time for parameter calibration varies across different datasets, it remains acceptable for real-time applications, with most cases achieving acceptable parameters after just 3 or even 1 epoch of calibration.

Experiments with real wildfire data

To evaluate the effectiveness of parameter calibration in real-world wildfire scenarios, we conducted an experiment analogous to our previous one. Specifically, we measured the relevant metrics both before and after calibration, utilizing real-world fire observation data obtained daily from satellite imagery of MODIS during the fire event (Cheng et al., 2022a). Additionally, dynamic wind data, updated daily, was incorporated to enhance the simulation environment.

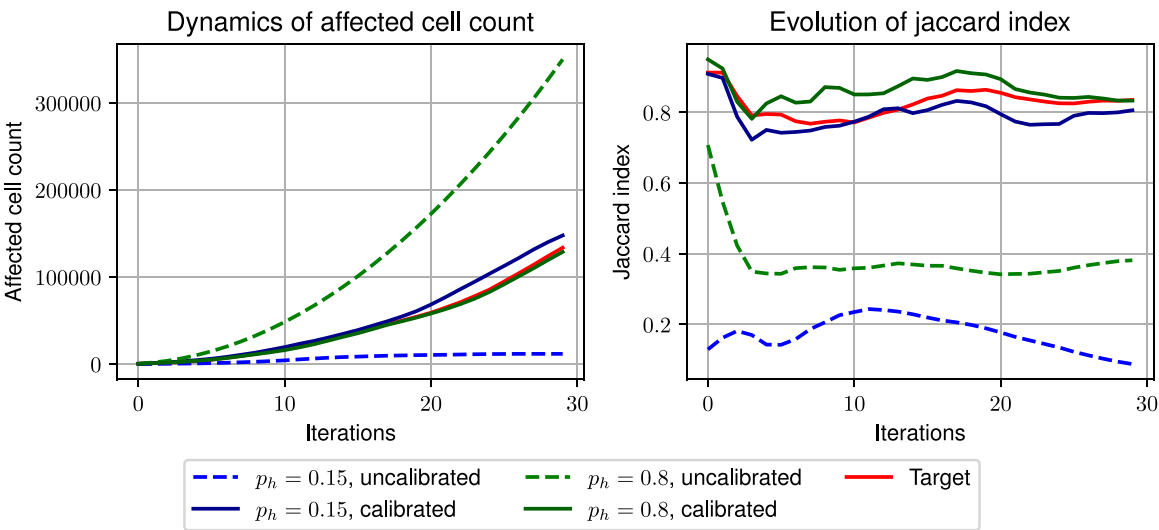


Fig. 8. Calibration results in the Bear 2020 fire ecoregion. The ‘Dynamics of Affected Cell Count’ chart shows alignment with targets across various iterations. The ‘Evolution of Jaccard Index’ chart reflects the accuracy stability.

	Under estimation $p_h = 0.15$ (initial)	Over estimation $p_h = 0.8$ (initial)	Under estimation $p_h = 0.15$ (calibrated)	Over estimation $p_h = 0.8$ (calibrated)	Target
Time step $t = 50$					
Time step $t = 100$					
Time step $t = 200$					
Time step $t = 300$					
Jaccard index	0.012	0.411	0.809	0.786	0.784 ± 0.061
Manhattan distance ($\times 10^4$)	95.79	155.57	5.11	12.95	11.02 ± 4.27

Fig. 9. Comparison of wildfire simulation results at various time steps in the Brattain 2020 mfire ecoregion. Two initial parameter sets (underestimation and overestimation) are used. Post-calibration, the simulations closely align with the target fire scale, demonstrating effective recovery despite the inherent randomness of CA.

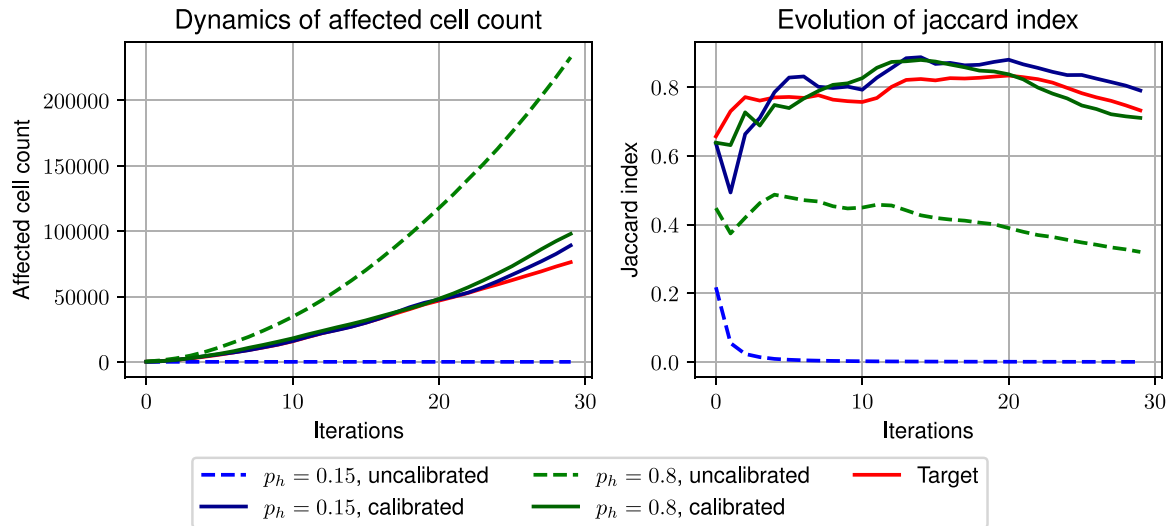


Fig. 10. Calibration results in the Brattain 2020 fire ecoregion. The ‘Dynamics of Affected Cell Count’ chart shows alignment with targets across various iterations. The ‘Evolution of Jaccard Index’ chart reflects the accuracy stability.

Table 3

Comparative analysis of wildfire parameter calibration results across multiple simulated datasets. Each dataset is evaluated under two pre-generated fire condition scenarios and five different initial ignition points.

Map name	Scale	Jaccard Index			Manhattan Distance ($\times 10^4$)			Running time (s)
		Uncalibrated	Calibrated	Target	Uncalibrated	Calibrated	Target	Per epoch
Bear 2020	Small	0.441 ± 0.072	0.452 ± 0.061	0.563 ± 0.113	20.34 ± 12.95	36.56 ± 20.17	12.73 ± 8.87	11.9
	Large	0.378 ± 0.144	0.907 ± 0.065	0.931 ± 0.018	113.02 ± 30.32	8.90 ± 11.64	2.98 ± 1.94	26.7
Brattain 2020	Small	0.021 ± 0.018	0.733 ± 0.092	0.767 ± 0.065	125.62 ± 28.62	24.67 ± 12.45	7.32 ± 4.20	60.0
	Large	0.013 ± 0.005	0.733 ± 0.039	0.687 ± 0.162	83.62 ± 31.09	15.04 ± 12.83	14.87 ± 12.95	63.3
Pier 2017	Small	0.065 ± 0.098	0.666 ± 0.104	0.693 ± 0.120	40.00 ± 17.36	12.25 ± 7.65	9.45 ± 5.07	42.4
	Large	0.071 ± 0.102	0.765 ± 0.146	0.774 ± 0.173	118.94 ± 27.89	18.47 ± 10.50	12.27 ± 14.92	45.8

This section presents the visualization of parameter calibration on real-world wildfire datasets ‘Bear 2020’ and ‘Pier 2017’. In the Bear 2020 map, one simulation time step t is equivalent to 0.8 h in the real world, and in Pier 2017, one simulation time step t is equivalent to 1.6 h in the real world.

Fig. 11 shows that, in the map ‘Bear 2020’, the target fire grows in a specific direction, gradually forming a shape. However, the growth speed changes from fast to slow over time. The uncalibrated simulation of underestimation case is close to the target in shape but does not match the same growth. The over-estimation fire differs significantly from the target fire. After calibration, the metrics of both cases improve. The underestimation case is closer to the target in fire scale, and the overestimation case now has a similar fire scale to the target fire. It is difficult to achieve the same shape due to the randomness of fire dynamics and the accuracy requirement of the dataset. Fig. 12 shows that the dynamics of affected cell count align with the target, but it is hard to imitate the gradual growth curve of real-world fire. This is a limitation of the current CA fire model, in that the fire tends to spread at a uniform speed during the simulation. The Jaccard Index remains stable and has greatly improved compared to the uncalibrated one.

Fig. 13 shows that, in the map ‘Pier 2017’, the target fire grows very slowly after forming a basic shape. The uncalibrated simulation of both cases differs from the target. After calibration, the metrics of both cases improve greatly in visualization. Both cases have a similar shape to the target fire and, most importantly, have their fire scale aligned with the target. The Manhattan Distance has improved by up to three times compared to before. Fig. 14 shows that the dynamics of the affected cell count become closer to the target. However, it is challenging to perfectly imitate the gradual growth curve of real-world fire due to the limitations of the CA model. The Jaccard Index remains stable and has greatly improved compared to the uncalibrated one.

4. Conclusions

In conclusion, PyTorchFire represents a significant advancement in wildfire simulation and parameter calibration, leveraging the computational power of GPUs and the flexibility of PyTorch. PyTorchFire utilizes geophysical and climate data as input to predict the spatio-temporal spread of wildfires. The software is fully scalable to accommodate different resolutions of input and observation data. By introducing the redesigned differentiable wildfire CA, PyTorchFire achieves remarkable computational efficiency, enabling millisecond-level simulations and real-world-scale wildfire modeling at high resolution, which are essential for timely wildfire management. The integration of automatic differentiation and gradient descent for parameter calibration ensures that the simulations closely align with real-world wildfire behavior, enhancing both the accuracy and reliability of predictions. PyTorchFire’s ability to maintain the stochastic nature of fire propagation while operating in real-world ecoregions makes it an invaluable tool for researchers and practitioners in the field of wildfire management. Its generalized and environment-data-sensitive design performs well in simulating early-stage fires, contributing to future wildfire early warning systems. As an open-access software, PyTorchFire not only advances the state-of-the-art in wildfire simulation but also democratizes access to powerful computational tools, fostering innovation and collaboration in combating uncontrolled wildfires. It has the potential to be integrated into automated fire monitoring system toolchains.

While current wildfire warnings primarily focus on the probability of ignition (Scott et al., 2013), they often overlook the spatial dynamics and potential consequences of the fire’s spread. This probabilistic approach tends to prioritize areas based on their likelihood of ignition rather than the potential severity or impact of a fire once it occurs. Incorporating spatial extent into risk assessments could provide a more

	Under estimation $p_h = 0.15$ (initial)	Over estimation $p_h = 0.8$ (initial)	Under estimation $p_h = 0.15$ (calibrated)	Over estimation $p_h = 0.8$ (calibrated)	Target
Iteration $it = 0$ Date 2020-08-20					
Iteration $it = 3$ Date 2020-08-23					
Iteration $it = 5$ Date 2020-08-25					
Iteration $it = 8$ Date 2020-08-28					
Jaccard index	0.311	0.197	0.408	0.397	
Manhattan distance ($\times 10^4$)	8.01	113.84	4.93	7.06	

Fig. 11. Comparison of wildfire simulation results with real-world satellite observations at various time steps on the Bear 2020 map. Post-calibration, the simulations align with the actual fire scale, demonstrating effective recovery in real-world scenarios.

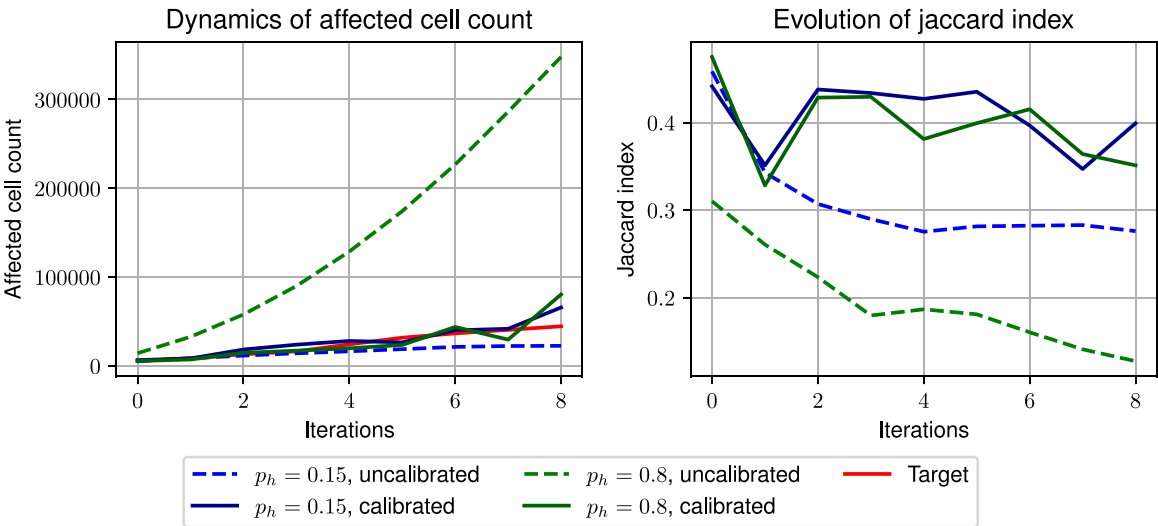


Fig. 12. Simulation results after calibration on the Bear 2020 map. The 'Dynamics of Affected Cell Count' chart demonstrates alignment with targets across various time steps. The 'Evolution of Jaccard Index' chart indicates the accuracy stability of fire scale recovery.

	Under estimation $p_h = 0.15$ (initial)	Over estimation $p_h = 0.8$ (initial)	Under estimation $p_h = 0.15$ (calibrated)	Over estimation $p_h = 0.8$ (calibrated)	Target
Iteration $it = 0$ Date 2017-08-30					
Iteration $it = 3$ Date 2017-09-02					
Iteration $it = 8$ Date 2017-09-07					
Iteration $it = 12$ Date 2017-09-11					
Jaccard index	0.253	0.468	0.389	0.525	
Manhattan distance ($\times 10^4$)	176.71	271.88	49.06	95.56	

Fig. 13. Comparison of wildfire simulation results with real-world satellite observations at various time steps on the Pier 2017 map. Post-calibration, the simulations align with the actual fire scale, demonstrating effective recovery in real-world scenarios.

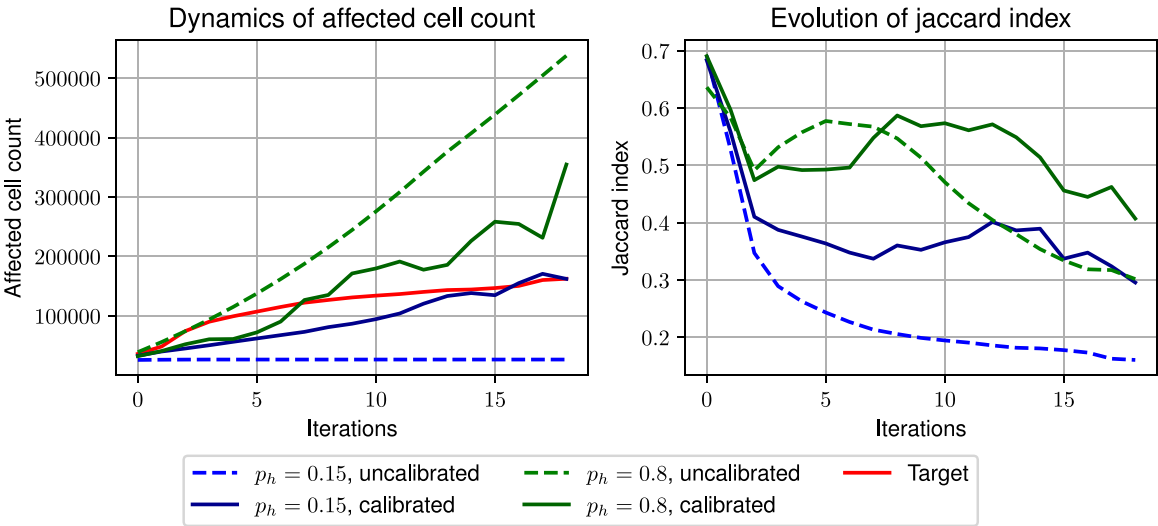


Fig. 14. Simulation results after calibration on the Pier 2017 map. The ‘Dynamics of Affected Cell Count’ chart demonstrates alignment with targets across various time steps. The ‘Evolution of Jaccard Index’ chart indicates the accuracy stability of fire scale recovery.

holistic view of wildfire threats. By considering factors such as fuel, topography, and weather patterns, we can better predict not only where a fire might start but also how it might evolve and spread across the landscape. This expanded perspective could enhance operational strategies, allowing for more targeted resource allocation and improved mitigation efforts. Such an approach acknowledges the complex interplay between environmental variables and human factors, ultimately leading to a more resilient response framework. By shifting our focus from mere probabilities to comprehensive risk evaluations, we can better safeguard communities and ecosystems from the multifaceted threats posed by wildfires.

Future work should focus on extending the application of PyTorchFire to global wildfire scenarios, incorporating various human interventions such as fire retardant placement and controlled burns. In this paper, we present a proof of concept for differentiable cellular automata with GPU acceleration, rather than an operational model. Our proposed framework is flexible and can be easily adapted for more advanced CA models, such as PROPAGATOR (Trucchia et al., 2020). For operational improvements, integrating these models and incorporating additional parameters, such as fine fuel moisture content, can enhance the realism of simulations. These parameters can be added by applying and calibrating appropriate scaling factors to Eq. (2). Reliable conversion of landscape data to scaling factors will significantly enhance the simulation. Additionally, due to the limitations of CA, methods need to be developed to achieve a non-linear spread speed. Obtaining high-resolution wind and fire data can also improve the accuracy of the simulation and parameter calibration. Once spotting effects are implemented, the simulation of fire dynamics will be more accurate. Modeling multiple ignition points is also pending.

CRediT authorship contribution statement

Zeyu Xia: Writing – original draft, Software, Investigation, Data curation. **Sibo Cheng:** Writing – review & editing, Validation, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors utilized OpenAI's gpt-4o-2024-05-13 API to enhance readability and language. Following the use of this service, the authors reviewed and edited the content as necessary and assume full responsibility for the content of the publication.

Acknowledgments

The authors would like to thank Juntao Lu, Yifan Shao, and Tianle Zhong (University of Virginia) for their valuable discussions on the research direction. We also extend our gratitude to Gang Xia for providing feedback on this paper. This paper is dedicated to the memory of the first author's grandfather, Junhong Xia. The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR) under reference ANR-22-CPJ2-0143-01. This research was supported by the academic research grant from NVIDIA.

Appendix A. Dataset

All data in this paper were projected and then cropped using the EPSG:3310 coordinate system for a consistent spatial reference. To avoid erroneous edge effects, a 4980-meter padded boundary was added to the original fire-affected region.

For landscape data, we utilized resources from LANDFIRE, downloaded via the LANDFIRE Product Service. Specifically, we used the Forest Canopy Bulk Density Version 2.3.0 (U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2023b) (unit: $\text{kg/m}^3 \times 100$) as our density input, the Forest Canopy Cover Version 2.3.0 (U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2023b) (unit: percentage) for canopy data, and the Slope 2020 dataset (U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2023a) (in degrees) for slope data.

Wind data were obtained from the Google Earth Engine dataset ECMWF/ERA5_LAND/DAILY_AGGR (Joaquín, 2019). The `u_component_of_wind_10m` and `v_component_of_wind_10m` bands were used to compute wind velocity and direction.

We used fire satellite observation data generated from Moderate Resolution Imaging Spectroradiometer (MODIS) (Giglio et al., 2016). MODIS provides thermal observations globally four times a day (Terra at 10:30 and 22:30; Aqua at 13:30 and 01:30 local time) at a resolution of approximately 1 km.

Appendix B. Environment

Most experiments were conducted on a shared server equipped with 64 vCPUs from an AMD EPYC 9654 @ 2.400 GHz (with AVX512 support), 240 GB of memory, and four NVIDIA RTX 4090 (24 GB) graphics cards. The operating system used was Ubuntu 22.04.3 LTS x86_64. The software environment included PyTorch 2.3.1 running on Python 3.11 (Ubuntu 22.04) with CUDA version 12.4.

To further evaluate the program's potential, additional tests were performed on a server with 14 vCPUs from an Intel®Xeon®Gold 6348 CPU @ 2.60 GHz, 100 GB of memory, and one NVIDIA A800 80 GB PCIe graphics card. The operating system for this setup was also Ubuntu 22.04.3 LTS x86_64. The software environment remained consistent, with PyTorch 2.3.1 on Python 3.11 (Ubuntu 22.04) and CUDA version 12.4.

Appendix C. Learning rate selection

To obtain optimal results, we experimented with multiple learning rates. Our findings indicate that the optimal learning rate varies depending on the specific map. However, they do not vary significantly in terms of results after sufficient epochs of calibration. We recommend that users experiment with different learning rates to determine the one that yields the best outcome.

For cases in Fig. 7, the learning rate is 5×10^{-3} for the under-estimation case and 1×10^{-2} for the over-estimation case. For cases in Fig. 9, the learning rate is 5×10^{-3} for the under-estimation case and 7×10^{-3} for the over-estimation case.

In Fig. 11, the learning rate is 1×10^{-2} for both under-estimation and over-estimation cases. In Fig. 13, the learning rate is 5×10^{-3} .

For the rest of the experiments, the learning rate is 5×10^{-3} .

Abbreviations

CA Cellular Automata

CFD Computational Fluid Dynamics

CUDA Compute Unified Device Architecture

DCA Differentiable Cellular Automata

GA Genetic Algorithm

GPU Graphic Processing Unit

IoU Intersection over Union

ML Machine Learning

MODIS Moderate Resolution Imaging Spectroradiometer

MPI Message Passing Interface

PyPI Python Package Index

MSE Mean Squared Error

BCE Binary Cross Entropy

Data availability

Data will be made available on request.

References

- Alessandri, A., Bagnerini, P., Gaggero, M., Mantelli, L., 2021. Parameter estimation of fire propagation models using level set methods. *Appl. Math. Model.* 92, 731–747. <http://dx.doi.org/10.1016/j.apm.2020.11.030>.
- Alexandridis, A., Vakalis, D., Siettos, C.I., Bafas, G.V., 2008. A cellular automata model for forest fire spread prediction: the case of the wildfire that swept through spetses island in 1990. *Appl. Math. Comput.* 204 (1), 191–201. <http://dx.doi.org/10.1016/j.amc.2008.06.046>.
- Ansel, J., Yang, E., He, H., Gimselshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C.K., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M.Y., Suk, H., Zhang, S., Suo, M., Tillet, P., Zhao, X., Wang, E., Zhou, K., Zou, R., Wang, X., Mathews, A., Wen, W., Chanan, G., Wu, P., Chintala, S., 2024. PyTorch 2: faster machine learning through dynamic python bytecode transformation and graph compilation. In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. In: ASPLOS '24, vol. 2, Association for Computing Machinery, New York, NY, USA, pp. 929–947. <http://dx.doi.org/10.1145/3620665.3640366>.
- Auer, M.R., Hexamer, B.E., 2022. Income and insurability as factors in wildfire risk. *Forests* 13 (7), 1130. <http://dx.doi.org/10.3390/f13071130>.
- Bak, P., Chen, K., Creutz, M., 1989. Self-Organized Criticality in the 'game of life. *Nature* 342 (6251), 780–782. <http://dx.doi.org/10.1038/342780a0>.
- Chen, G., Guo, Y., Yue, X., Tong, S., Gasparrini, A., Bell, M.L., Armstrong, B., Schwartz, J., Jaakkola, J.J.K., Zanobetti, A., Lavigne, E., Saldiva, P.H.N., Kan, H., Royé, D., Milojevic, A., Overcenco, A., Urban, A., Schneider, A., Entezari, A., Vicedo-Cabrera, A.M., Zeka, A., Tobias, A., Nunes, B., Alahmad, B., Forsberg, B., Pan, S.-C., Iñiguez, C., Ameling, C., Valencia, C.D.I.C., Åström, C., Houthuijs, D., Dung, D.V., Samoli, E., Mayvaneh, F., Sera, F., Carrasco-Escobar, G., Lei, Y., Orru, H., Kim, H., Holobaca, I.-H., Kysely, J., Teixeira, J.P., Madureira, J., Katsouyanni, K., Hurtado-Díaz, M., Maasikmets, M., Ragetti, M.S., Hashizume, M., Stafoggia, M., Pascal, M., Scottichini, M., Coelho, M.d.Z.S., Ortega, N.V., Rytí, N.R.I., Scovronick, N., Matus, P., Goodman, P., Garland, R.M., Abrutsky, R., Garcia, S.O., Rao, S., Fratianni, S., Dang, T.N., Colistro, V., Huber, V., Lee, W., Seposo, X., Honda, Y., Guo, Y.L., Ye, T., Yu, W., Abramson, M.J., Samet, J.M., Li, S., 2021. Mortality risk attributable to wildfire-related PM2.5 pollution: A global time series study in 749 locations. *Lancet Planet. Heal.* 5 (9), e579–e587. [http://dx.doi.org/10.1016/S2542-5196\(21\)00200-X](http://dx.doi.org/10.1016/S2542-5196(21)00200-X).
- Cheng, S., Jin, Y., Harrison, S.P., Quilodrán-Casas, C., Prentice, I.C., Guo, Y.-K., Arcucci, R., 2022a. Parameter flexible wildfire prediction using machine learning techniques: forward and inverse modelling. *Remote. Sens.* 14 (13), 3228. <http://dx.doi.org/10.3390/rs14133228>.
- Cheng, S., Prentice, I.C., Huang, Y., Jin, Y., Guo, Y.-K., Arcucci, R., 2022b. Data-driven surrogate model with latent data assimilation: application to wildfire forecasting. *J. Comput. Phys.* 464, 111302. <http://dx.doi.org/10.1016/j.jcp.2022.111302>.
- Collette, A., Kluyver, T., Caswell, T.A., Tocknell, J., Kieffer, J., Jelenak, A., Sco-patz, A., Dale, D., Chen, Vincent, T., Einhorn, M., payno, Juliagarra, Sciarelli, P., Valls, V., Ghosh, S., Pedersen, U.K., Kittisopikul, M., jakirkham, Raspaud, M., Danilevski, C., Abbasi, H., Readey, J., Mühlbauer, K., Paramonov, A., Chan, L., Schepper, R.D., Solé, V.A., jialin, Guest, D.H., 2024. H5py/H5py. <http://dx.doi.org/10.5281/zenodo.594310>, (Accessed: 7 July 2024).
- Denham, M., Laneri, K., 2018. Using efficient parallelization in graphic processing units to parameterize stochastic fire propagation models. *J. Comput. Sci.* 25, 76–88. <http://dx.doi.org/10.1016/j.jocs.2018.02.007>, arXiv:1701.03549.
- Doyle, M., Threet, M., Dotter, M., Kempis, C., Tapley, A., Welsh, T., 2024. SimFire. <https://github.com/mitrefireline/simfire>.
- Finney, M.A., Brittain, S., Seli, R.C., McHugh, C.W., Gangi, L., 2023. FlamMap: fire mapping and analysis system (version 6.2). <https://www.firelab.org/project/flammap>.
- Giglio, L., Schroeder, W., Justice, C.O., 2016. The collection 6 MODIS active fire detection algorithm and fire products. *Remote Sens. Environ.* 178, 31–41. <http://dx.doi.org/10.1016/j.rse.2016.02.054>.
- Gui, W., Xia, Z., Gong, R., Wang, G., Chen, B., Zhang, D., 2023. Singing voice detection based on a deeper convolutional neural network. In: *Proceedings of the 3rd International Symposium on Automation, Information and Computing*, vol. 1. ISAIC, SciTePress, INSTICC, Online, pp. 336–341. <http://dx.doi.org/10.5220/0011924600003612>.
- Guo, X., Zhang, C., Lu, J., Wang, Y., Duan, Y., Yang, T., Zhu, Z., Chen, L., 2024. OpenStereo: A comprehensive benchmark for stereo matching and strong baseline. <http://dx.doi.org/10.48550/arXiv.2312.00343>.
- Hanson, H.P., Bradley, M.M., Bossert, J.E., Linn, R.R., Younker, L.W., 2000. The potential and promise of physics-based wildfire simulation. *Environ. Sci. & Policy* 3 (4), 161–172. [http://dx.doi.org/10.1016/S1462-9011\(00\)00083-6](http://dx.doi.org/10.1016/S1462-9011(00)00083-6).
- Hernández Encinas, A., Hernández Encinas, L., Hoya White, S., Martín del Rey, A., Rodríguez Sánchez, G., 2007. Simulation of Forest Fire Fronts Using Cellular Automata. *Adv. Eng. Softw.* 38 (6), 372–378. <http://dx.doi.org/10.1016/j.advengsoft.2006.09.002>.
- Hoang, R.V., 2008. *Wildfire Simulation on the GPU* (Ph.D. thesis). University of Nevada, Reno.
- Jain, P., Coogan, S.C., Subramanian, S.G., Crowley, M., Taylor, S., Flannigan, M.D., 2020. A review of machine learning applications in wildfire science and management. *Environ. Rev.* 28 (4), 478–505. <http://dx.doi.org/10.1139/er-2020-0019>.
- Joaquín, M.S., 2019. ERA5-land monthly averaged data from 1981 to present. In: *Copernicus Climate Change Service (C3S) Climate Data Store (CDS)*. <http://dx.doi.org/10.24381/cds.68d2bb30>.
- Li, X.G., 2018. XC-Li/Parallel_CellularAutomaton_Wildfire. https://github.com/XC-Li/Parallel_CellularAutomaton_Wildfire. (Accessed: 19 April 2024).
- López-De-Castro, M., Trucchia, A., Morra Di Cella, U., Fiorucci, P., Cardillo, A., Pagnini, G., 2024. Fire-spotting modelling in operational wildfire simulators based on cellular automata: a comparison study. *Agricult. Forest. Meteorol.* 350, 109989. <http://dx.doi.org/10.1016/j.agrformet.2024.109989>.
- Maffei, C., Menenti, M., 2019. Predicting forest Fires Burned Area and rate of spread from pre-fire multispectral satellite measurements. *ISPRS J. Photogramm. Remote Sens.* 158, 263–278. <http://dx.doi.org/10.1016/j.isprsjprs.2019.10.013>.
- Martin, C., 2017. Differentiable Cellular Automata. <http://dx.doi.org/10.48550/arXiv.1708.09546>.
- Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M., 2020. Growing neural cellular automata. *Distill* 5 (2), e23. <http://dx.doi.org/10.23915/distill.00023>.
- National Interagency Fire Center, 2024. Wildfires and acres. <https://www.nifc.gov/fire-information/statistics/wildfires>. (Accessed: 28 July 2024).
- Ntinas, V.G., Moutafis, B.E., Trunfio, G.A., Sirakoulis, G.C., 2017. Parallel fuzzy cellular automata for data-driven simulation of wildfire spreading. *J. Comput. Sci.* 21, 469–485. <http://dx.doi.org/10.1016/j.jocs.2016.08.003>.
- Papadopoulos, G.D., Pavlidou, F.-N., 2011. A comparative review on wildfire simulators. *IEEE Syst. J.* 5 (2), 233–243. <http://dx.doi.org/10.1109/JSYST.2011.2125230>.
- Rogozhnikov, A., 2021. Einops: clear and reliable tensor manipulations with Einstein-like notation. In: *International Conference on Learning Representations*. <https://openreview.net/forum?id=oapK5VM2b3j>.
- Scott, J.H., Thompson, M.P., Calkin, D.E., 2013. A Wildfire Risk Assessment Framework for Land and Resource Management. Technical Report RMRS-GTR-315, U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station, Ft. Collins, CO. <http://dx.doi.org/10.2737/RMRS-GTR-315>.
- Trucchia, A., D'Andrea, M., Baghino, F., Fiorucci, P., Ferraris, L., Negro, D., Gollini, A., Severino, M., 2020. PROPAGATOR: an operational cellular-automata based wildfire simulator. *Fire* 3 (3), 26. <http://dx.doi.org/10.3390/fire3030026>.
- U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2023a. LANDFIRE 2.2.0. <https://lfrps.usgs.gov/arcgis/rest/services/LandfireProductService/GPServer>. (Accessed: 12 May 2024).
- U.S. Department of the Interior, Geological Survey, and U.S. Department of Agriculture, 2023b. LANDFIRE 2.3.0. <https://lfrps.usgs.gov/arcgis/rest/services/LandfireProductService/GPServer>. (Accessed: 12 May 2024).
- Valero, M.M., Jofre, L., Torres, R., 2021. Multifidelity prediction in wildfire spread simulation: Modeling, uncertainty quantification and sensitivity analysis. *Environ. Model. Softw.* 141, 105050. <http://dx.doi.org/10.1016/j.envsoft.2021.105050>.
- Xia, Z., Ma, K., Cheng, S., Blackburn, T., Peng, Z., Zhu, K., Zhang, W., Xiao, D., Knowles, A.J., Arcucci, R., 2023. Accurate identification and measurement of the precipitate area by two-stage deep neural networks in novel chromium-based alloys. *Phys. Chem. Chem. Phys.* 25 (23), 15970–15987. <http://dx.doi.org/10.1039/D3CP00402C>.

- Xu, R., Yu, P., Abramson, M.J., Johnston, F.H., Samet, J.M., Bell, M.L., Haines, A., Ebi, K.L., Li, S., Guo, Y., 2020. Wildfires, global climate change, and human health. *N. Engl. J. Med.* 383 (22), 2173–2181. <http://dx.doi.org/10.1056/NEJMs2028985>.
- Yassemi, S., Dragičević, S., Schmidt, M., 2008. Design and implementation of an integrated GIS-based cellular automata model to characterize forest fire behaviour. *Ecol. Model.* 210 (1–2), 71–84. <http://dx.doi.org/10.1016/j.ecolmodel.2007.07.020>.
- Zhang, C., Collin, A., Moireau, P., Trouvé, A., Rochoux, M.C., 2019. State-parameter estimation approach for data-driven wildland fire spread modeling: Application to the 2012 rxcadre S5 field-scale experiment. *Fire Saf. J.* 105, 286–299. <http://dx.doi.org/10.1016/j.firesaf.2019.03.009>.
- Zhong, T., Velázquez, I.M., Haneda, Y., 2021. Involution based speech autoencoder: Investigating the advanced vision operator performance on speech feature extraction. In: 2021 IEEE 10th Global Conference on Consumer Electronics. GCCE, pp. 179–180. <http://dx.doi.org/10.1109/GCCE53005.2021.9621826>.