Climate Predictor

Davide Bulotta

February 9, 2025

1 Introduction

The **Climate Predictor**[1] project is designed to explore the application of *Reinforcement Learning*[9] (RL) within a *Federated Learning* (FL)[3] framework, leveraging real-world meteorological data to predict temperature trends. The implementation relies on *Ray*[4], a distributed computing framework that facilitates large-scale machine learning tasks by enabling efficient parallel processing and robust model training.

By structuring the system in a federated manner, it is possible to train a predictive model across multiple decentralized data sources.

The main objective of this project is to develop an intelligent agent capable of learning patterns in weather variations and making temperature predictions based on meteorological attributes such as humidity, atmospheric pressure, and wind speed.

2 Dataset

To ensure the realism and reliability of the simulation, meteorological data was sourced from the **National Oceanic and Atmospheric Administration** (NOAA)[5], which provides access to comprehensive weather records from numerous stations across the United States.

For this project, **nine weather stations** located in Miami were selected, with data collected at hourly intervals over a period spanning from **January 1, 2016, to January 25, 2025**. The dataset was curated to remove inconsistencies, particularly missing values, which were significantly present in certain attributes. Some columns had over **90% missing data**, necessitating their exclusion to maintain data integrity.

The dataset includes a variety of meteorological features, such as:

- **HourlyVisibility**: 4% missing
- **HourlyStationPressure**: 4% missing
- **HourlyRelativeHumidity**: 4% missing
- **HourlyWindDirection**: 4% missing

- **HourlyWindSpeed**: 4% missing
- **HourlyAltimeterSetting**: 6% missing
- **HourlyWetBulbTemperature**: 5% missing
- **HourlyDewPointTemperature**: 4% missing

These variables serve as inputs to the predictive model.

3 Model Architecture

The system adopts an **agent-based approach**[7] in which an intelligent agent interacts with a simulated environment. The environment is represented by the climate dataset, where each time step corresponds to an hourly record from a specific weather station.

The chosen reinforcement learning strategy is based on **Proximal Policy Optimization (PPO)**[8], a widely used algorithm that ensures stable training by limiting excessive policy updates. PPO operates by iteratively refining its predictions while maintaining a balance between exploration (testing new strategies) and exploitation (leveraging learned patterns).

3.1 Training Process

The training process unfolds in sequential steps. Initially, the model is provided with climate data up to a certain date, allowing the agent to learn the relationships between different meteorological parameters. The reinforcement learning framework then iteratively adjusts the model's policy based on feedback received from its predictions.

The learning system is structured to advance one day at a time. At each iteration, the head node updates the dataset by revealing new hourly data points. This ensures that the model is continuously exposed to fresh information (like real world environment), enhancing its ability to generalize across different weather conditions.

Each node independently processes the data available for its assigned weather station, meaning that some nodes may have access to more complete datasets than others. This variability reflects real-world scenarios, where certain weather stations may experience missing data due to sensor malfunctions or communication failures.

3.2 Model results

The model was trained with a step size of 2000 and a batch train size of 500, using an entropy coefficient of 0.01 and a learning rate of 0.00001. The training process spanned multiple nodes in a distributed cluster, with iterations reaching up to 950,000 sampled and trained steps. The entropy values generally decreased over time, which is expected as the policy becomes more deterministic.

The rewards obtained during rollouts showed a general trend of improvement, with predicted values approaching true values. However, occasional deviations between predictions and actual values indicate areas for further fine-tuning.

The following table shows the prediction results:

True Value	Predicted Value	Error
84.00	87.65	+3.65
87.00	87.06	+0.06
68.00	70.09	+2.09
80.00	82.74	+2.74
75.00	73.76	-1.24

Table 1: Comparison between true values and predicted values (Note: the temperatures are in Fahrenheit[2])

4 System Architecture

4.1 Discovery Head

The *discovery-head* plays a central role in managing the distributed learning process. This node is responsible for initializing and supervising worker nodes, ensuring that each station operates efficiently. It continuously monitors the status of the nodes, collects their locally computed weights, and aggregates them to update the global model.

The system is designed to function autonomously, adapting dynamically to the availability of computing resources. If a node becomes unavailable due to hardware failure or network issues, the head node redistributes tasks among the remaining nodes, provided they have the necessary data to proceed with training.

Fault tolerance is a key aspect of the system. Since Ray operates in a distributed environment, occasional failures are expected. To mitigate potential disruptions, the *discovery-head* incorporates Ray's built-in exception handling mechanisms[6]. This prevents the entire system from crashing due to the failure of a single node, ensuring continuity in the learning process.

4.1.1 Federated Weight Aggregation

A crucial function of the *discovery-head* is the aggregation of model weights received from individual nodes. This is achieved using a **Federated Aggregator** component known as the , which collects the trained model parameters from each weather station, converts them into tensors, and computes an average to update the global model. Into specific: the aggregation process collects the *default_policy* weights from all nodes and computes a new global model by recursively averaging these values. Although the approach is inspired by Federated

Averaging, it currently performs a simple arithmetic mean without weighting each node's contribution by its local data volume.

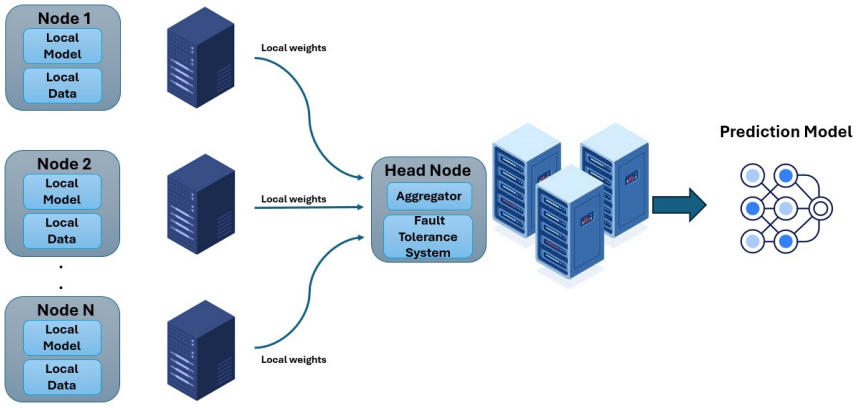

Figure 1: System Architecture design.

4.2 Worker Nodes

Each worker node corresponds to a weather station and operates independently from other nodes. Due to hardware constraints, the project deployment consists of three worker nodes per server, distributed across three servers, excluding the *discovery-head*.

A worker node is initialized with specific attributes, including access to its designated dataset and a storage directory for training checkpoints. Checkpoints are essential for maintaining training progress, as they allow the node to resume from the last successfully completed step in case of an unexpected failure. This mechanism prevents redundant computations and enhances the efficiency of the learning process.

When a node receives new data from the *discovery-head*, it processes the information locally, updates its model weights, and give the updated parameters back to the head node for aggregation. This decentralized training strategy significantly reduces the amount of data that needs to be transmitted across the network, improving scalability and computational efficiency.

4.3 Scalability and Performance Analysis

A key goal of this system is to remain operational and efficient as more nodes (meteorological stations) join the cluster. In the *Climate Predictor* project, each

node's tasks (*ping*, *train*, *get_weights*) are scheduled in parallel by the *discovery-head* using Ray.

A crucial aspect of this implementation is the way results are retrieved from worker nodes. Initially, using a simple blocking call like `ray.get()` resulted in inefficient execution because each task was retrieved one at a time, sequentially, failing to leverage the parallelism of the cluster. Instead, we implemented `ray.wait()`, which launches all tasks in parallel and then blocks execution while waiting for all nodes to either finish or time out. This approach ensures that all nodes complete execution in parallel, rather than waiting for each task sequentially.

Despite this improvement, performance can degrade when scaling up, especially when multiple nodes run on the same physical machine.

Configuration	Mean Training Time (s)	Total Training Time (s)	Aggregation Time (s)
Number of nodes 1	215.26	215.26	0.050
Number of nodes 3 (1 per server)	213.80	225.51	0.066
Number of nodes 3 (same server)	247.97	262.83	0.068
Number of nodes 9 (3 per server)	232.31	280.18	0.125

Table 2: Comparison of training times and aggregation time for different node configurations.

The first configuration achieves the lowest mean training time (213.80 s) and total training time (225.51 s), demonstrating the benefit of distributing nodes across different physical machines. When all three nodes are placed on a single server, as in the **Node 3 (same server)** configuration, both mean and total training times increase (247.97 s and 262.83 s, respectively), indicating resource contention due to shared CPU and memory.

With the **Node 9 (3 per server)** configuration, where three nodes run on each of three servers, the mean training time (232.31 s) remains comparable to the three-node configurations.

The main challenge is the aggregation time nearly doubles from 0.066 s in the three-node setup to 0.125 s on nine-node. As more nodes contribute model updates, the *discovery-head* must process a larger volume of weight tensors, leading to increased communication and computational overhead.

In real-world contexts with potentially hundreds or thousands of nodes, the aggregator's workload can become a bottleneck. Therefore, investing in hardware resources for the *discovery-head* or considering distributed aggregators and efficient data-handling techniques would be crucial for sustaining performance at scale.

References

[1] Davide Bulotta. Climatepredictor: A python-based tool for climate data analysis and prediction. `https://github.com/Degik/ClimatePredictor`, 2025.

[2] Hasok Chang. *Inventing Temperature: Measurement and Scientific Progress*. Oxford University Press, 2004.

[3] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence, 2016.

[4] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications, 2018.

[5] National Oceanic and Atmospheric Administration (NOAA). Climate data online (cdo). Online: (https://www.ncdc.noaa.gov/cdo-web/datasets), 2025.

[6] Ray Core Team. Fault tolerance. Ray Documentation, 2023. [Accessed 04-02-2025].

[7] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.