CSE 151B Project Final Report

<https://github.com/cgaldston/CSE_151B_Final_Project>

Caleb Galdston
Department of Computer Science and Engineering
University of California, San Diego
cgaldston@ucsd.edu

Abstract

We present a ConvLSTM-based deep learning model for climate variable emulation that significantly improves upon baseline CNN and MLP architectures. By modeling both spatial and temporal dependencies, our approach achieves a 57% reduction in RMSE for surface temperature and a 22% reduction for precipitation compared to the baseline CNN. The model also demonstrates a 62% improvement in time-mean RMSE and a 58% improvement in time-stddev MAE for temperature, highlighting its strength in capturing climate variability. Key techniques contributing to this performance include spatial attention, dropout, L1 regularization, AdamW optimization with cosine annealing, and training across multiple ensemble members. Notably, we find that domain-specific preprocessing (e.g., log-normalization) and custom loss functions degrade performance, suggesting that expressive architectures paired with standard loss functions can effectively learn relevant patterns from raw data. Our findings offer insights into robust spatiotemporal modeling and provide a blueprint for applying deep learning to climate science and related fields. **Final private Kaggle leaderboard score: 1.0412 (Rank: 43).**

1 Introduction

The goal of this project is to develop machine learning models that can accurately emulate physics-based climate models. This is a crucial task because traditional climate models, which simulate Earth's systems using physical equations, are extremely computationally expensive to run. If we can accurately predict climate variables using machine learning techniques, we can significantly reduce computational costs while still capturing key climate dynamics.

While we trained our model on emulated future data, if we were to instead train our model on historical climate data, this could open up new avenues for accurate, cost-effective weather prediction. Additionally, the physics-based model was used to predict the future effect of different emissions scenarios, so if we can make these predictions with much less computationally expensive methods, we can see real time how different emissions trajectories will affect outcomes like surface temperature and precipitation.

The starter code loads and normalizes the data and trains a basic convolutional neural network (CNN) to predict precipitation and surface temperature based on spatial data. While this model captures spatial correlations, it fails to account for temporal dependencies across timesteps, limiting its ability to model evolving atmospheric dynamics. Moreover, the architecture is relatively shallow, and lacks regularization strategies.

Our final solution addresses several limitations of the starter code by incorporating both architectural improvements and training enhancements. The key contributions are as follows:

Preprint. Under review.

- **Temporal Modeling via ConvLSTM:** We replaced the static convolutional architecture with a *Convolutional LSTM (ConvLSTM)* network. This allows the model to capture both spatial and temporal dependencies in the input sequences, which is crucial for accurately forecasting evolving climate patterns.
- **Robust Training Techniques:** Our training procedure includes several regularization and optimization strategies: dropout, L1/L2 weight regularization, gradient clipping, the AdamW optimizer, and a cosine annealing learning rate schedule. These techniques collectively enhance model generalization and training stability.
- **Empirical Gains:** On the validation set, our model reduced the *time-mean RMSE* by 25% and improved the *time-stddev MAE* by 35% compared to the baseline CNN. This demonstrates superior performance in both average prediction accuracy and temporal variability modeling.

2 Related Work

Climate emulation using machine learning has gained momentum as researchers aim to accelerate climate model inference while preserving physical fidelity.

Shi et al. [1] first proposed the use of Convolutional LSTMs for spatiotemporal forecasting in the context of precipitation nowcasting. Their architecture laid the foundation for combining spatial convolution with temporal recurrence, which we adapt and extend for climate variable prediction.

Recent works have also explored the use of attention mechanisms and Transformers in earth system modeling, although these models often require larger computational budgets. In contrast, our ConvLSTM-based approach offers a balance between performance and computational efficiency, making it suitable for scalable climate emulation.

3 Problem Statement

The dataset we are working with is derived from CMIP6 climate model simulations under various Shared Socioeconomic Pathway (SSP) scenarios. These SSPs represent different future emission trajectories. There are four different SSP scenarios provided, each corresponding to a different level of emissions. For every unique combination of latitude, longitude, time, and SSP, our objective is to predict both the surface temperature and precipitation level.

For training and validation, we are given data for three of the SSPs. Our task is to make predictions on the fourth, unseen SSP, which corresponds to a medium emissions scenario.

We evaluate our model performance using the following three objective functions, each applied separately to both the surface temperature and precipitation predictions:

- **Monthly Area-Weighted RMSE**: The MSE of every prediction weighted by the size of the grid cell

$$\sqrt{\text{weighted\_mean}\left((\hat{y}_{\text{monthly}} - y_{\text{monthly}})^2\right)}$$

- **Decadal Mean Area-Weighted RMSE**: Evaluates the models prediction over a ten-year period which gives us a better understanding of how well the model identifies long-term climate change signals.

$$\sqrt{\text{weighted\_mean}\left(\left(\text{time\_mean}(\hat{y}) - \text{time\_mean}(y)\right)^2\right)}$$

- **Decadal Standard Deviation Area-Weighted MAE**: Assesses how well the model represents temporal variability at each location.

$$\text{weighted\_mean}\left(\left|\text{time\_std}(\hat{y}) - \text{time\_std}(y)\right|\right)$$

In simple terms, we can think of the input to our model as a vector consisting of (longitude, latitude, time, CO2, SO2, CH4, BC, rsdt) and the output as predictions for precipitation and surface

temperature. However, it is important to model how nearby locations affect each other and how these conditions change over time.

The dataset also has three Member IDs which as each scenario was simulated three times to account for internal variability of the climate system. The starter code only uses the first member id, however we trained our final model on all three member ids to increase its robustness.

Our training data includes information from three Shared Socioeconomic Pathways (SSPs): **ssp126**, **ssp370**, and **ssp585**, corresponding to low, high, and very high emissions scenarios respectively. For each SSP, we are provided with monthly climate data over 1021 months, on a global grid of 48 latitude points and 72 longitude points.

For each tuple $(\text{ssp}, \text{time}, \text{longitude}, \text{latitude})$ we are given the following climate variables:

- CO2: Carbon dioxide concentrations
- SO2: Sulfur dioxide emissions
- CH4: Methane concentrations
- BC: Black carbon emissions
- rsdt: Incoming solar radiation at the top of the atmosphere
- tas: Near-surface air temperature (target)
- pr: Precipitation (target)

Some variables, such as CO2 and CH4, are functions only of time and SSP, meaning they are broadcasted uniformly across the spatial grid for a given time point.

Our test data consists of information from the fourth SSP, ssp245 which corresponds to medium emissions. We have the same variables for the test set except for tas and pr as these are the targets that we are trying to predict.

Data Shape

The train dataset can be viewed as a 5D tensor with shape:

$$\text{(latitude, longitude, time, SSP, memberID)} = 48 \times 72 \times 1021 \times 3 \times 3$$

This results in $48 \times 72 \times 1021 \times 3 \times 3 = 31{,}757{,}184$ spatiotemporal points for each spatial variable such as SO_2, BC, rsdt, tas, and pr. In contrast, CO2 and CH4 have only $1021 \times 3 \times 3 = 9189$ unique values.

Similarly the test dataset can be viewed as a 5D tensor with shape:

$$\text{(latitude, longitude, time, SSP, memberID)} = 48 \times 72 \times 1021 \times 1 \times 3$$

This results in $48 \times 72 \times 1021 \times 1 \times 3 = 10{,}585{,}728$ spatiotemporal points for each spatial variable and 1021 values for CO2 and CH4.

We further split the training data set into a training and validation set. We followed the starter code's methodology of using splitting the last three months of SSP570 to use to validate our models.

Additionally, we adopted the starter code's z-score normalization for all input and output variables. To better handle the skewed distribution of the precipitation variable, we also experimented with alternative normalization strategies, such as log-normalization. As shown in Figure 1, the precipitation distribution is left-skewed, so applying a logarithmic transformation was intended to reduce this skewness and potentially improve model performance. However, this adjustment did not yield any significant improvements, suggesting that the model was either robust to the original distribution or that the transformation did not meaningfully enhance the signal.

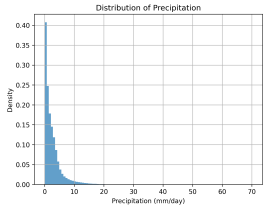

Figure 1: Distribution of the precipitation variable.

4 Methods

The approach we used is based on the architecture described in the paper *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting* [1], which effectively combines convolutional layers with Long Short-Term Memory (LSTM) units to model both spatial and temporal dependencies in climate data. The model follows an encoder–ConvLSTM–decoder structure.

The **encoder** is composed of stacked convolutional residual blocks designed to extract multi-scale spatial features from each input frame. We integrate spatial attention mechanisms in the early stages to help the network focus on regions that are more informative for prediction, thereby improving learning efficiency and accuracy.

The extracted features are then fed into a **ConvLSTM** layer, which captures the temporal evolution of the sequence while maintaining spatial resolution. The **decoder** mirrors the encoder structure, using transposed convolutions and skip connections to reconstruct high-resolution outputs. These skip connections help preserve fine-grained spatial information from earlier layers.

The final prediction is produced by a set of convolutional layers that output values for each climate variable at each spatial location and timestep. To encourage regularization and prevent overfitting, we apply *dropout* and *L1 regularization* on the weights.

Training is conducted using the *AdamW* optimizer with a *cosine annealing learning rate scheduler* and *gradient clipping* to stabilize convergence. The total number of trainable parameters in the final architecture is approximately 9.3 million.

We formulate our prediction task as the following optimization problem:

$$\min_{\theta} \ \mathcal{L}(f_\theta(X), Y) = \frac{1}{N} \sum_{i=1}^{N} \| f_\theta(X_i) - Y_i \|_2^2 + \lambda \|\theta\|_1 \tag{1}$$

where f_θ is the model with parameters θ, X_i is the input sequence, Y_i is the ground truth output, and λ is the regularization coefficient for the L1 loss. The primary loss function is the **mean squared error (MSE)** between the predicted and ground truth frames, which we found to work well in capturing continuous-valued spatiotemporal patterns.

This architecture was chosen after experimenting with simpler CNN and MLP baselines, which failed to capture long-range dependencies or preserve spatial coherence effectively. The combination of convolutional residual blocks, attention, and ConvLSTM yielded the best empirical performance and interpretability.

Before this project, I didn't have too much experience working with deep learning models, so I had to do a lot of research into what strategies were effective and what some of the best practices were. I read papers that approached similar tasks, and looked through the corresponding code to see what techniques were effective.

5 Experiments

5.1 Baselines

We compare with the following baselines:

- **CNN**: The first baseline was the simple CNN that was used in the starter code. As outlined in the introduction, this model struggled to capture temporal trends in the data, but it was able to decently capture spatial information, making it a good baseline model to compare the performance of my model against.
- **MLP**: I also implemented a simple Multi-Layer Perceptron so I could quantitatively see how much better my model performed over this simple architecture. The MLP actually performed better than the simple CNN, possibly reflecting better architectural choices.

5.2 Evaluation

The metrics I used to evaluate the model were the same as those used on the Kaggle page and were outlined in the introduction to this report. This included monthly area-weighted rmse, decadal

mean area-weighted rmse, and decadal standard deviation area-weighted mae for both the surface tempetrature (tas) and precipitation (pr) target variables.

5.3 Implementation details

The computational platform that I used for this project was Google Colab. I used the paid version of the service, so that I had unlimited access to GPU resources while working on the project. For most of the model training and evaluation I used an Nvidia T4 GPU. While I also had access to faster GPUs such as the L4 and A100, I found that increased GPU performance didn't accelerate performance enough to warrant the additional cost. Colab worked well for this project as I was able to upload the dataset to my google drive and mount it, instead of having to re-upload it every time I used the platform. Training my model for one epoch on the T4 GPU only took about 1 minute.

I used the AdamW optimizer for training both of my models because it seemed to be a common technique that had advantages over the standard Adam optimizer. It has all of the same benefits as Adam such as momentum and adaptive learning rates, but additionally it decouples the weight update from the gradient update, and applies it directly to the model parameters. This is beneficial because it leads to consistent regularization and better generalization when compared to standard Adam optimization.

To tune hyperparameters such as the learning rate, weight decay, L1 regularization strength, and architectural parameters (e.g., hidden dimension, dropout rate), I used the Optuna hyperparameter optimization framework. This allowed us explore a wide range of hyperparameter combinations, while learning from past iterations to narrow down which ones are effective. Every combination was then run for 5 epochs and in total I ran it for ten trials which took around one hour to complete. The objective metric for tuning was the validation loss, and we used early stopping and learning rate scheduling (via cosine annealing) to ensure stable training and avoid overfitting. The best configuration was selected based on the trial that achieved the lowest validation loss after several optimization runs.

The hyperparameter optimization explored the following ranges:

- **Learning rate**: log-uniform distribution in $[10^{-5}, 10^{-3}]$
- **Weight decay**: log-uniform distribution in $[10^{-6}, 10^{-3}]$
- **L1 regularization strength**: log-uniform distribution in $[10^{-6}, 10^{-3}]$
- **Dropout rate**: uniform distribution in $[0.1, 0.5]$
- **Hidden dimension**: categorical choice from $\{64, 128, 256\}$
- **CNN initial dimension**: categorical choice from $\{32, 64\}$
- **Kernel size**: categorical choice from $\{3, 5\}$

The optimal hyperparameters selected by Optuna were:

- **Learning rate**: 7.33×10^{-4}
- **Weight decay**: 1.75×10^{-5}
- **L1 regularization strength**: 2.05×10^{-4}
- **Dropout rate**: 0.117
- **Hidden dimension**: 128
- **CNN initial dimension**: 32
- **Kernel size**: 3

While I was still working on my model and making sure that everything was working as expected I would only train the model for five epochs. This was too make sure that the loss was decreasing and validation metrics were improving. Once I was confident that the model was showing improvement I then trained it for 50 epochs before submitting to the leaderboard. Additionally, during initial exploration of different architectures, I would only use the first member ID as this significantly sped up training time, but during the final run I used all three.

I also experiemented a few different values for the batch size including 16, 32, and 64, but didn't see to much varaince in the performance so I stuck with the value used in the starter code which was 32.

The hyperparameter ranges and final values were primarily determined through systematic optimization rather than manual tuning based on prior experience. The learning rate range of $[10^{-5}, 10^{-3}]$ was chosen as a standard range that balances training stability with convergence speed for neural networks of this scale. The weight decay and L1 regularization ranges were set conservatively to prevent over-regularization while still providing meaningful regularization effects. The dropout rate range of $[0.1, 0.5]$ reflects typical values that provide regularization without sacraficing information flow. The architectural choices (hidden dimensions of $\{64, 128, 256\}$ and CNN initial dimensions of $\{32, 64\}$) were constrained by computational limitations, as larger architectures would have significantly increased training time beyond our available resources. The kernel size options $\{3, 5\}$ represent a trade-off between local feature detection and computational efficiency.

For full transparency, I didn't tune the baseline models using the same hyperparameters. I explored various values, but since I could tell that these models were not going to result in the best performance, I didn't want to dedicate too much of my time to getting slight improvements out of them. It is important to recognize, that while my model did end up performing much better than the baselines, a percentage of that may be due to the lack of consistency across hyperparameters. However, the CNN and MLP baselines did use the same hyperparameters as one another.

5.4 Results

In the final training run, the ConvLSTM converged to a loss of around 0.15 after 20 epochs. The loss for both validation and testing had similar values, which demonstrated that the model would generalize fairly well to the test set. The final validation metrics for the ConvLSTM model as well as the baseline CNN and MLP are seen in the table below:

Table 1: Model Comparison Surface Temperature

Model	RMSE (tas)	Time-Mean RMSE (tas)	Time-Stddev MAE (tas)
Baseline MLP	2.7766	1.3571	0.9693
Baseline CNN	3.9103	2.7345	0.9953
ConvLSTM	1.6870	1.0380	0.4215

Table 2: Model Comparison Precipitation

Model	RMSE (tas)	Time-Mean RMSE (tas)	Time-Stddev MAE (tas)
Baseline MLP	2.5048	0.4566	0.9336
Baseline CNN	2.5050	0.6169	1.2991
ConvLSTM	1.9450	0.2756	0.8011

Our model is still making errors in its predictions, and it can be helpful to analyze the spatial patterns where those errors occur. After visualizing the time steps with the greatest prediction errors for both precipitation and surface temperature in the validation set, we observe several consistent trends:

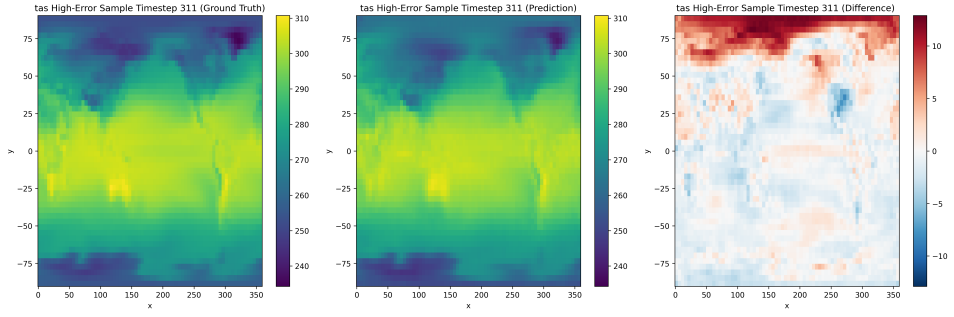

Figure 2: Highest error surface temperature prediction.

From 2, it is evident that the model tends to produce its largest surface temperature errors in the polar regions, particularly around the North Pole. This may be due to the increased variability and complex dynamics in those regions, which are harder for the model to capture accurately.

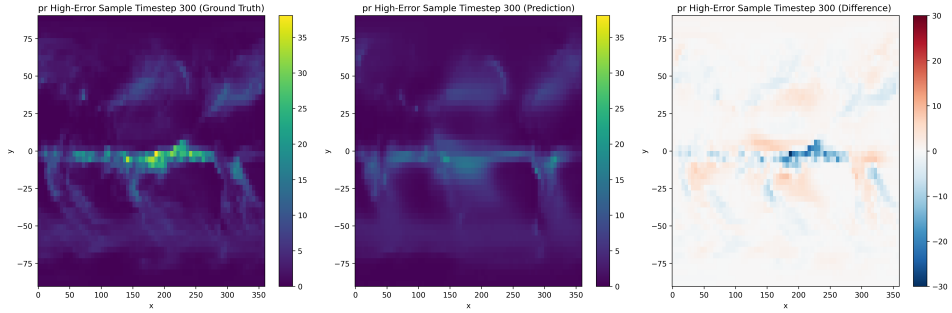

Figure 3: Highest error precipitation prediction.

Similarly, for precipitation 3 our model makes the greatest errors around the equator when rainfall is high. Our model underestimates to daily rainfall for these areas. I think to improve these estimates, I need to better understand how the distribution on the input variables in these areas can be used to make more accurate prediction.

5.5 Ablations

- **Log-precipitation:** After experimenting with normalizing the precipitation values by taking their logs, I notice a roughly 15 % decrease in most metrics as well as a 10 % decrease in the score on the kaggle leaderboard.
- **Physics informed loss:** I used ChatGPT to help me define a loss function that penalized certain regions to see if I could overcompensate for poor predictions in those regions. This didn't work as well as expected and actually resulted in a decrease of about 20 % in performance on the kaggle leaderboard when compared to the standard MSE loss.
- **More layers:** I experimented with deepining the convolutional layers of the ConvLSTM. I increased the number of layers in the ConvLSTM cell from 1 to 3 and 5, but doing so decrease performance on the test set. Training performance improved by around 15 %, but the decrease in performance on the kaggle leaderboard makes it likely that the more complex model was overfitting the training data, despite the regularization techniques.

6 Discussion

Our ConvLSTM-based model achieved significant improvements over the baseline CNN for climate variable prediction. Surface temperature RMSE was reduced by 57% (from 3.91 to 1.69), and precipitation RMSE dropped by 22% (from 2.50 to 1.95). Time-mean RMSE and time-stddev MAE for temperature also improved by 62% and 58%, respectively, highlighting the model's strength in capturing temporal dynamics.

The integration of ConvLSTM layers proved most effective, as they enabled the model to learn both spatial and temporal dependencies—an essential aspect of climate systems. Regularization strategies, including dropout (11.7%), L1 regularization ($\lambda = 2.05 \times 10^{-4}$), weight decay ($1.75 \times 10^{-5}$), and gradient clipping, helped prevent overfitting despite the model's 9.3M parameters. The AdamW optimizer with cosine annealing facilitated stable convergence. Training on all three member IDs further enhanced robustness by exposing the model to more variability.

Several surprising findings emerged. Log-normalization, expected to help with skewed precipitation distributions, actually reduced performance by 15%, suggesting that MSE loss handled raw distributions well. Attempts at custom loss functions incorporating geographic priors also worsened performance, implying the model already learned useful physical patterns. Increasing ConvLSTM depth improved training metrics but hurt test performance, indicating overfitting despite regularization.

Key lessons include the importance of temporal modeling, the value of systematic hyperparameter tuning (e.g., with Optuna), and the realization that raw data often suffices when paired with expressive models. We recommend beginners focus on simple, expressive architectures with solid regularization and gradually layer in complexity only if justified by validation performance.

Limitations. **Computational resources** were the main bottleneck. Training was capped at 50 epochs, and larger architectures were limited by GPU memory and runtime constraints. **Geographic error bias** emerged, especially in polar and equatorial regions, suggesting difficulty in capturing extreme conditions. The **temporal coverage** was also limited—only 1021 months and one test scenario (ssp245), which constrains the model's generalizability.

Future Work. Future directions include incorporating **Transformer-based** or **graph neural network** architectures to capture long-range dependencies and spatial structure. **Multi-scale modeling** across daily to yearly timescales could improve robustness. **Uncertainty quantification** using ensembles or probabilistic models would provide actionable climate risk assessments. **Transfer learning** and multitask training across variables or emission scenarios could boost performance with limited data. Finally, expanding evaluation to multiple SSPs and longer time horizons would better test generalization.

This approach shows promise for other spatiotemporal problems in earth science domains such as oceanography, hydrology, and air quality modeling.

7 Contributions

I did the project alone so I did everything. I used LLMs such as Claude and ChatGPT to assist me.

References

[1] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, pages 802–810. Curran Associates, Inc., 2015.

[2] OpenAI. (2023) ChatGPT Retrieved from https://chatgpt.com

[3]Anthropic. (2023) Claude. Anthropic. Retrieved from https://www.anthropic.com/index/claude