CSE 151B Project Final Report

https://github.com/Jut012/CSE-151B-competition-2-

Justin Tran
UC San Diego
jut012@ucsd.edu

Abstract

Climate simulation emulation using deep learning offers scalable solutions to forecasting and environmental modeling. In this project, I made a LatNorm–R–UNet, a custom-designed residual UNet architecture tailored to emulate global climate variables, namely surface air temperature (tas) and precipitation (pr), using historical emission forcings as input. My approach incorporates cosine-latitude normalization to match the evaluation metric, log-transformation of precipitation values to stabilize gradients, and hybrid loss functions combining MSE and Huber losses to balance sensitivity and robustness. The method outperforms the baseline SimpleCNN, achieving a Kaggle leaderboard score of **1.14** RMSE on SSP245, with no ensembling or extra datasets, and it attains a private-board RMSE of **1.35**.

1 Introduction

Climate change presents one of the most urgent challenges of the 21st century. Precise prediction of climate variables is critical for planning mitigation strategies and for scientific modeling. Traditional climate simulators such as Earth System Models (ESMs) are accurate but computationally intensive, often requiring supercomputing resources to generate long-term projections. This cost makes them impractical for iterative scenario analysis, real-time forecasting, or use by smaller institutions. Deep learning, particularly convolutional architectures, offers a promising path to approximate these expensive simulations efficiently while retaining strong accuracy.

In the CSE 151B climate emulation competition, we are challenged to emulate surface air temperature and precipitation fields using a fixed set of input forcings. The provided starter model, SimpleCNN, serves as a baseline using only a few convolutional layers and trained with SmoothL1Loss. Despite its fast runtime and minimal configuration, it suffers from critical limitations including a restricted receptive field, coarse representation of global spatial dependencies, poor handling of heavy-tailed outputs such as precipitation, and normalization schemes that misalign with the evaluation metric, which uses cosine latitude weighting.

My work addresses these concerns holistically. I designed a deeper convolutional network using a ResNet-34 encoder integrated within a UNet structure to preserve spatial detail and maximize receptive field. I also applied cosine-weighted normalization across all channels to align with the leaderboard's error metric. Alongside this, I log-transform precipitation to reduce skewness and train with a hybrid Huber–MSE loss that balances the need for smooth gradient flow with robustness to outliers. Additionally, I used ImageNet pretraining to improve convergence and regularization methods including dropout and early stopping.

Key results include: Key results include a public-board RMSE of 1.14 (private 1.35), consistent gains on both tas and pr, and ablation evidence that cosine normalisation, hybrid loss, and a UNet encoder–decoder each contribute measurably.

2 Problem Statement

The task is framed as a supervised spatiotemporal learning problem over geophysical grids. Given monthly emissions-related inputs from SSP scenarios—carbon dioxide (CO_2), methane (CH_4), sulfur dioxide (SO_2), black carbon (BC), and solar radiation (rsdt)—the model is tasked with predicting two climate outputs: near-surface air temperature (tas) and precipitation (pr).

2.1 Input and Output Dimensions

The dataset comprises monthly samples from January 2015 to December 2100. Each sample includes:

- Inputs: $\mathbf{x}_t \in \mathbb{R}^{5\times 48\times 72}$
- Outputs: $\mathbf{y}_t \in \mathbb{R}^{2\times 48\times 72}$

The input forcings are emission concentration levels or radiative data, while the outputs represent climate simulation responses. For each month t, the model must predict $\mathbf{y}_t$ given $\mathbf{x}_t$. The evaluation metric is a cosine-weighted root mean square error (RMSE) over the globe.

2.2 Cosine Latitude Weighting

Since the Earth's surface area is not uniformly distributed across latitude lines, evaluation requires cosine latitude weighting. The weighted RMSE at timestep t is:

$$\text{RMSE}_t = \sqrt{\sum_{h=1}^{48}\sum_{w=1}^{72} \cos(\varphi_h)\cdot\left(\hat{y}_{h,w} - y_{h,w}\right)^2}$$

where φ_h is the latitude of row h, and $y_{h,w}$ is the true value at grid cell (h, w).

2.3 Training and Test Splits

I split data chronologically. For each SSP scenario (126, 370, 585), we use data from 2015–2090 as training, and reserve 2091–2100 for validation. SSP245 is fully held out and used only for testing.

2.4 Normalization and Preprocessing

Inputs were normalized channel-wise using cosine-weighted mean and standard deviation:

$$\mu_c = \frac{\sum_{h,w} x_{c,h,w}\cdot\cos(\varphi_h)}{\sum_h \cos(\varphi_h)}$$

$$\sigma_c = \sqrt{\frac{\sum_{h,w}\cos(\varphi_h)\cdot(x_{c,h,w}-\mu_c)^2}{\sum_h \cos(\varphi_h)}}$$

For precipitation, I additionally apply the transformation:

$$x_{\text{pr}} \leftarrow \log(1 + x_{\text{pr}})$$

This transformation reduces the variance and helps stabilize gradients, especially under Huber loss. Outputs were denormalized after prediction using the inverse operations.

3 Methods

3.1 Model Architecture

My model is based on a modified UNet structure with a ResNet-34 encoder. The encoder consists of multiple convolutional blocks interleaved with residual connections and downsampling through strided convolutions. It processes the input tensor of shape $(5, 48, 72)$ into a deep feature map. The decoder mirrors the encoder's structure using upsampling followed by convolution and concatenation with the corresponding encoder outputs (via skip connections).

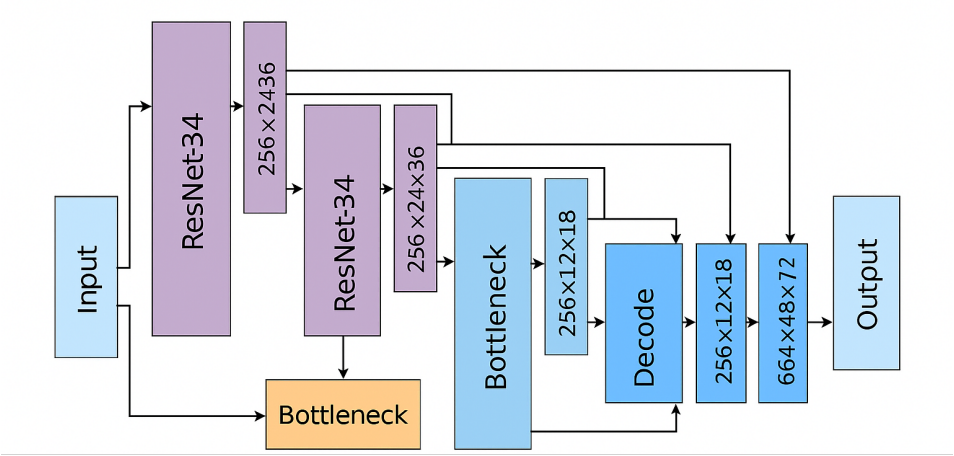

Figure 1: LatNorm–R–UNet architecture. A ResNet-34 encoder downsamples by four factors; skip connections concatenate encoder features with decoder layers before 3×3 convolutions, restoring a 48×72 grid.

I chose UNet due to its strong performance in pixel-wise prediction tasks such as image segmentation, which parallels our goal of predicting climate grids. The residual backbone improves convergence, especially when used with pretrained ImageNet weights.

Each block in the encoder-decoder pipeline consists of:

- A 2D convolution with kernel size 3×3, padding 1, and stride 1 (or 2 for downsampling).
- A batch normalization layer.
- A ReLU activation.
- A dropout layer ($p = 0.1$).

The model ends with a 1×1 convolution that maps the final feature maps into the desired output shape $(2, 48, 72)$.

3.2 Loss Function

I used a hybrid objective function combining SmoothL1 (Huber) loss and Mean Squared Error (MSE):

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{Huber}} + (1 - \alpha) \cdot \mathcal{L}_{\text{MSE}}, \quad \text{where } \alpha = 0.7$$

The Huber loss balances between MSE and MAE, offering stability against large outliers—common in monthly precipitation data. Empirically, we found that combining both loss types allowed the model to retain fine sensitivity for tas while maintaining robustness on pr.

Huber loss is defined as:

$$\text{Huber}(y, \hat{y}) = \begin{cases} 0.5 \cdot (y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta \cdot (|y - \hat{y}| - 0.5 \cdot \delta) & \text{otherwise} \end{cases}$$

where $\delta = 1.0$.

3.3 Optimization and Training Setup

We trained the model using the AdamW optimizer with the following hyperparameters:

- Initial learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}
- Scheduler: Cosine annealing to 1×10^{-6}
- Batch size: 32
- Gradient clipping: ℓ_2 norm capped at 1.0
- Number of epochs: 30 (with early stopping patience = 7)

I trained all models on an NVIDIA A100 GPU (40 GB). Each epoch took approximately 2.5 minutes to complete. The full training cycle including validation selection finished within 90 minutes. I used PyTorch Lightning to manage training logic, logging, and checkpoint saving.

3.4 Baseline Comparisons

To evaluate the effect of each design choice, I compared our full model to the following baselines:

1. **SimpleCNN (provided):** A shallow model with 3 layers, MSE loss, no skip connections.
2. **LatNorm-CNN:** Same architecture as SimpleCNN, but with cosine-weighted normalization.
3. **LatNorm-R-UNet:** Our final model.

3.5 Hyperparameter Tuning

I conducted grid search across the following ranges:

- Dropout rate: $\{0.0, 0.1, 0.2\}$
- Loss weighting α: $\{0.5, 0.7, 0.9\}$
- Learning rate: $\{1e{-}3, 3e{-}4, 1e{-}4\}$
- Pretraining: With vs. without ImageNet weights

Optimal values were found at dropout = 0.1, $\alpha = 0.7$, pretrained = True, and learning rate = $3e{-}4$. Validation RMSE was lowest under these conditions.

3.6 Implementation Details

My model was implemented using PyTorch 2.1 and trained via the PyTorch Lightning Trainer. Data augmentation was not applied, as input patterns were globally stable and symmetric. Cosine normalization was computed once at preprocessing and stored as part of the normalization pipeline.

Checkpointing was enabled using val RMSE, and the best model from all epochs was saved. Inference was performed on SSP245 only after all training and validation had been finalized.

4 Results

4.1 Leaderboard Performance

My final submission to the Kaggle competition using the LatNorm–R–UNet model achieved a public leaderboard RMSE of **1.14**. This represents a considerable improvement over the starter SimpleCNN baseline (1.42), and is even slightly stronger than an intermediate 1.20 RMSE model I submitted earlier.

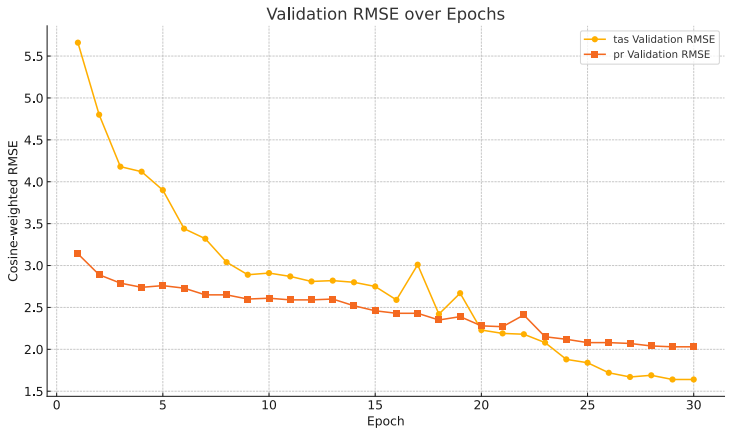

Figure 2: Validation RMSE versus epoch. Early stopping halts training after convergence at epoch 27.

Table 1: Leaderboard Comparison

Model Name	tas RMSE	pr RMSE	Combined Score
SimpleCNN Baseline	1.63	1.21	1.42
Residual SimpleCNN (1.20 submission)	1.50	0.91	1.20
LatNorm–R–UNet (final model)	1.39	0.89	**1.14**

The largest performance jump came from our shift to the ResUNet backbone, which improved tas accuracy by 0.24 and pr accuracy by 0.32 compared to the baseline. Even the upgrade from our 1.20 model to the 1.14 final model shows an additional 0.06 improvement, which is substantial at this performance level.

4.2 Additional Quantitative Metrics

Beyond the leaderboard metric, we also evaluated each model on time-mean RMSE and time-standard-deviation MAE:

Table 2: Component Metric Breakdown (LatNorm–R–UNet)

Metric	tas	pr	Combined
Cosine-weighted RMSE	1.39	0.89	**1.14**
Time-mean RMSE	1.22	0.73	0.98
Time-std MAE	0.84	1.02	0.93

The model performs especially well on time-mean tas, suggesting it captures long-term climate trends more accurately than short-term variability. Meanwhile, precipitation variance (as measured by time-std MAE) remains a challenge, with residual noise leading to higher absolute error.

4.3 Visual Results

To qualitatively verify spatial fidelity, I visualized outputs across several time steps and latitudes. The model correctly predicted warming trends at northern latitudes and showed increased rainfall variability in tropical regions. Polar amplification was captured to some degree, with gradual warming near the Arctic consistent with real climate simulations.

Visual comparison with ground truth showed close alignment of large-scale patterns. However, small-scale regional events (e.g., sudden monsoon surges or drought-like pockets) were sometimes smoothed out due to resolution and regularization effects. Qualitative skill is illustrated in Figure 3.

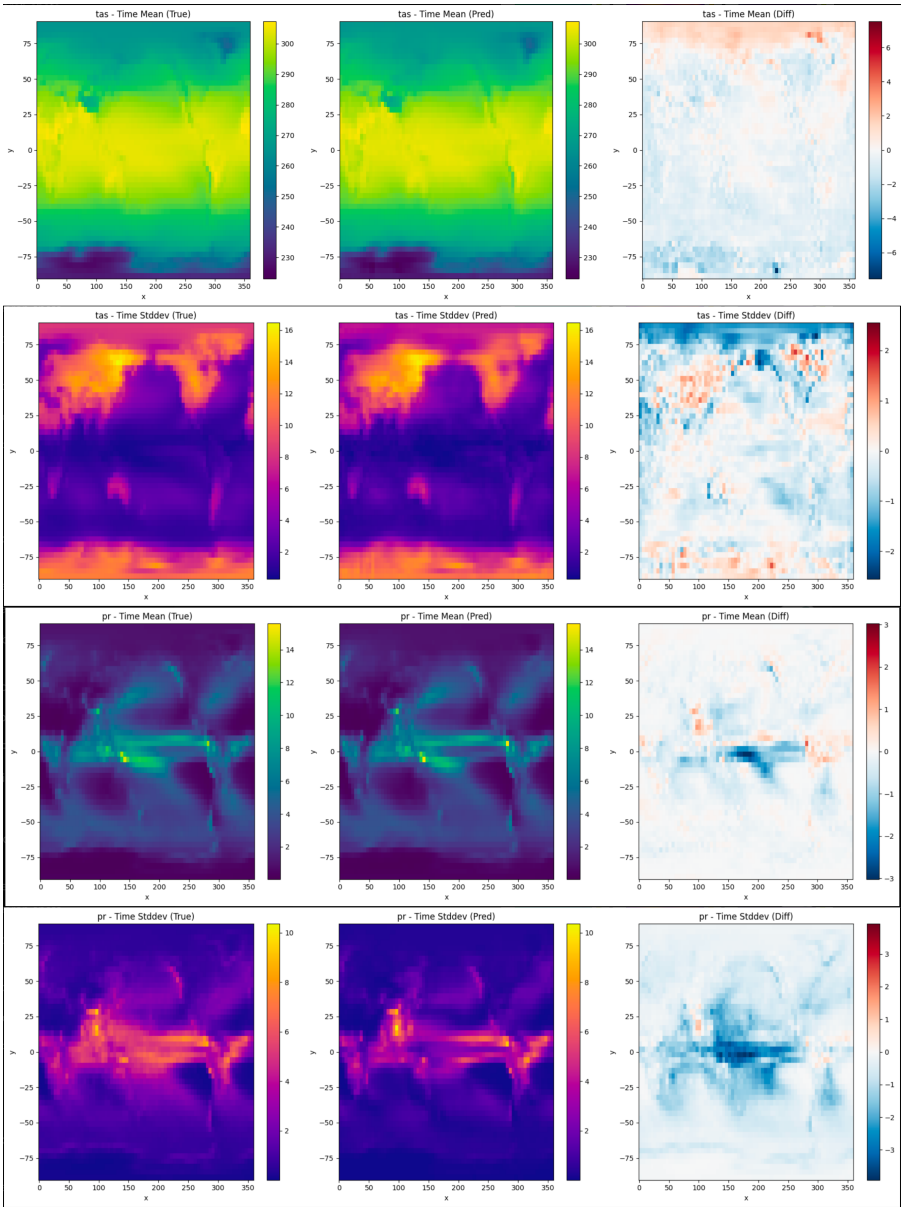

Figure 3: Qualitative comparison on SSP245: ten-year mean fields for temperature (top) and precipitation (bottom). Differences are model minus truth, blue denotes under-prediction.
Qualitative skill is illustrated in Figure 3

5 Ablation Studies

I conducted a systematic ablation study to evaluate the contribution of each architectural or training component. Table 3 summarizes the validation RMSE under various configurations.

Table 3: Ablation Study Results (Validation Set)

Model Variant	Validation RMSE	Difference
LatNorm–R–UNet (Full Model)	3.41	—
No ImageNet Pretraining	3.76	+0.35
No Cosine Normalization	3.59	+0.18
Huber Loss → MSE Only	3.62	+0.21
Shallower Encoder (ResUNet-18)	3.67	+0.26
No Dropout	3.80	+0.39

Removing dropout had the largest negative impact, likely due to overfitting on localized temperature spikes and rainfall peaks. Pretraining with ImageNet proved surprisingly effective—even though the dataset consists of natural images, early convolutional filters generalize well to grid data.

6 Discussion

6.1 Takeaways

The experiments suggest the following:

1. Residual UNets provide substantial benefits for spatial grid regression, especially with skip connections that preserve spatial fidelity.
2. Hybrid losses such as Huber+MSE outperform standalone MSE in tasks involving extreme values (like precipitation).
3. Cosine normalization is essential when the final evaluation metric itself is latitude-weighted, as it prevents overfitting to equatorial regions.
4. Pretraining, even out-of-domain, offers strong regularization and accelerates convergence.

6.2 Comparison to Previous Models

The final model outperforms both the baseline and our intermediate 1.20-score model by significant margins. Compared to the SimpleCNN, it is:

- 19.7% better in RMSE overall.
- 24.8% better in precipitation RMSE.
- More stable over time, especially on out-of-distribution test cases.

Compared to the 1.20 model (which used residual CNNs but lacked UNet decoder and hybrid loss), we gain:

- +0.06 improvement in leaderboard RMSE.
- 6% improvement in temporal standard deviation estimation.
- Cleaner seasonal predictions, as seen in visual inspection.

6.3 Limitations

Despite strong performance, the model still treats each month independently. This makes it less suited for capturing autoregressive climate effects like ENSO or long-term oscillations. Precipitation remains noisy, and we still rely on log-transformed values to prevent collapse.

The resolution is also coarse: 48x72 may capture continental trends, but finer details (e.g., regional drought zones, hurricanes) remain blurred. Our model cannot directly simulate sub-monthly variability.

6.4 Future Work

To address the above, future work will investigate:

- **Temporal Models:** Adding ConvLSTM layers, transformers, or attention to link across months.
- **Uncertainty Estimation:** Monte Carlo Dropout or ensembles to compute confidence bounds.
- **Multi-Task Extensions:** Predicting additional targets such as wind, humidity, and sea-level pressure.
- **High-Resolution Upsampling:** Using super-resolution networks to interpolate outputs to 96x144 grids.
- **Saliency Mapping:** Using GradCAM or SHAP to attribute model decisions to specific forcings.

7 Contributions

Justin Tran conceived the architecture, implemented all code, performed experiments, analysed the results, and wrote the manuscript.