CSE 151B Project Final Report

<https://github.com/yuz227/CSE-151B-Project>

Mark Sui
University of California San Diego
tisui@ucsd.edu

Yunqi Zhang
University of California San Diego
yuz227@ucsd.edu

Charles Cai
University of California San Diego
dicai@ucsd.edu

Abstract

Deep learning models such as Multi-layer perceptrons (MLPs) have recently emerged as lightweight yet effective tools for emulating complex physical simulations, including climate forecasting. However, accurately predicting high-resolution spatiotemporal climate variables under varying emission scenarios remains a key challenge due to nonlinear dynamics and limited computational resources. In this project, we develop and evaluate a range of deep learning models—including SimpleCNN, Vision Transformer, and a final advanced MLP—to predict surface temperature (tas) and precipitation (pr) across Shared Socioeconomic Pathways (SSPs). Trained on SSP126, SSP370, and SSP585 and tested on the unseen SSP245, our models demonstrate steady performance gains with architectural refinement. The advanced MLP achieves the best results, with a validation loss of 0.1515 and RMSEs of 1.2978 for tas and 1.9386 for pr. Our submission ranks 19th out of 83 teams on the Kaggle public leaderboard (score: 0.9152), showcasing the MLP's potential as a fast, generalizable climate emulator.

1 Introduction

Accurately predicting future climate conditions is a critical deep learning task with far-reaching real-world implications. As the consequences of climate change become increasingly urgent, the ability to efficiently emulate high-resolution physical climate models can help inform decisions in environmental policy, agriculture, infrastructure planning, and disaster mitigation. Traditional Earth system models are computationally intensive and time-consuming, whereas deep learning models offer a promising alternative by enabling fast, scalable, and approximate simulations based on learned patterns.

In this project, our goal is to develop deep learning-based emulators that predict spatial maps of **surface temperature (tas)** and **precipitation rate (pr)** from a set of key climate forcing variables (e.g., CO_2, CH_4, SO_2, BC, rsdt). These predictions are made across different **latitudes, longitudes, and time steps**, making this a spatiotemporal regression task. The models are trained on historical and projected climate scenarios under different Shared Socioeconomic Pathways (SSPs), and are evaluated on unseen SSPs (e.g., `ssp245`) to assess generalization to new climate trajectories.

The starter code provided a baseline convolutional neural network (SimpleCNN) to predict surface temperature (tas) and precipitation (pr) from climate forcing variables such as CO_2, CH_4, SO_2, black carbon (BC), and rsdt, across several Shared Socioeconomic Pathways (SSPs). While the baseline was a functional starting point, it suffered from two key limitations: **slow training speed** and **limited predictive accuracy**. Even after 30 epochs (taking $\sim$ 15 minutes each), the best validation RMSEs remained relatively high (tas: **2.62**, pr: **2.08**) with a loss of **0.1811**, indicating underfitting and saturation.

To address these issues, we explored multiple model architectures. We first improved the CNN by applying higher dropout, mixed-precision training, and reducing batch size, which yielded performance gains but remained limited by GPU memory and diminishing returns. Next, we implemented a Vision Transformer (ViT) to capture long-range spatial dependencies more effectively. The ViT demonstrated **competitive generalization** on precipitation metrics (e.g., `val/pr/time_mean_rmse = 0.3157`), outperforming CNNs in some areas. However, its overall performance(e.g. `tas RMSE: 1.72`) lagged behind our final model, likely due to insufficient training time and high resource requirements.

Our best-performing solution was a multi-layer perceptron (MLP) architecture that flattened input variables and passed them through four dense layers with dropout and ReLU activation. This design eliminated spatial overhead while preserving predictive power. Training was extremely faster (under 5 seconds per epoch), and the MLP achieved the **lowest validation loss (0.1515)** and RMSEs (tas: **1.2978**, pr: **1.9386**). Our final submission on Kaggle scored **0.9152**, ranking **#19 out of 83 teams**, confirming the effectiveness of our model under realistic hardware constraints.

2 Problem Statement

2.1 Climate Emulation Task

In this project, we treat the task as a supervised learning problem over space and time, using by the geophysical grids. The model takes in monthly forcing inputs from different SSP scenarios, including CO_2, CH_4, SO_2, BC, and incoming rsdt. Based on these inputs given, our goal is going to predict two important climate variables: near-surface temperature (tas) and rainfall (pr). Genrally speaking, The input x is in the form of a 4-dimensional tensor with T = number of time-steps, P = number of grid-points along latitude, Q = number of longitude, and D = number of input channels. More specifically, each training sample with index i is denoted as variable d at time t and coordinates (p, q):

$$x^i_{t,p,q,d} \in \mathbf{R}^{T\times P\times Q\times D_{in}}$$

The output y with index i is denoted as channel d at time t and coordinates (p, q):

$$y^i_{t,p,q,d} \in \mathbf{R}^{T\times P\times Q\times D_{out}}$$

2.2 Data, inputs, and outputs

- Scenarios are SSP126, SSP370, SSP585 for training and validation; SSP245 for test only.
- Timeline: January 2015 - December 2100
- Each scenario has $T = 1{,}021$ monthly steps on a 48×72 grid. Since the dataset contains monthly samples which from January 2015 to the December 2100.

$$X_t \in \mathbb{R}^{5\times 48\times 72}, \qquad Y_t \in \mathbb{R}^{2\times 48\times 72}.$$

- Split: SSP126, 370, 585 is for train and validation; SSP245 is held-out test. Within each training trajectory we keep the first 90 % months for training and the last 10 % for validation and time order preserved.
- Normalise: Standard z-score per channel, $\hat{x} = (x - \mu)/\sigma$, plus $\hat{\text{pr}} = \log(1 + \text{pr})$ to tame heavy tails.

2.3 Normalization & Pre-processing

- **Cosine-weighted z-score.** For each input channel c

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

where φ_h is the latitude of row h.

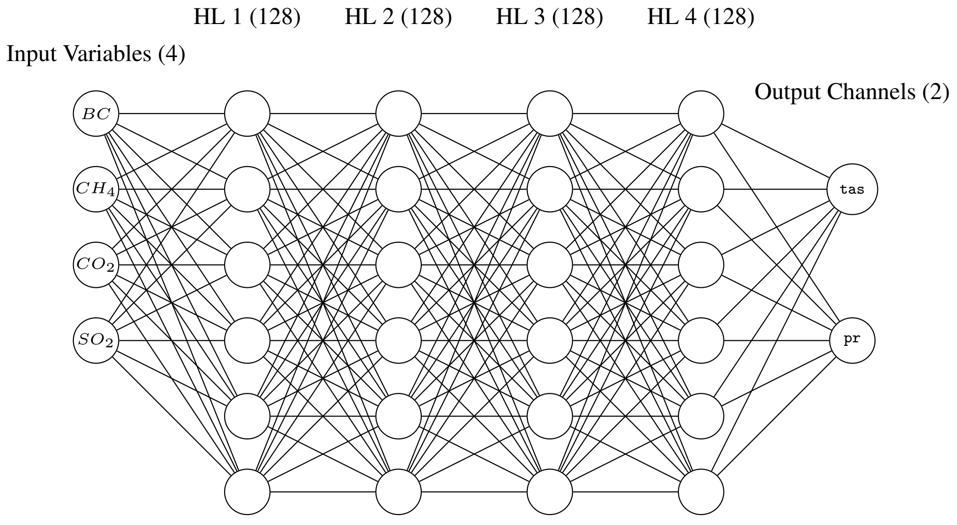
- Rain skew fix: $\hat{x}_{\text{pr}} \leftarrow \log(1 + \hat{x}_{\text{pr}})$.
- Reverse at the end: Predictions are de-normalised before computing RMSE.
- Evaluation metric: We followed the cosine-weighted RMSE as $\sqrt{\sum_{h,w} \cos\varphi_h \left(\hat{y}_{h,w} - y_{h,w}\right)^2}$, averaged over all grid cells and timesteps. Because grid cells cover different surface areas, we weight errors by $\cos(\varphi_h)$, the latitude of row h. The weighted RMSE at timestep t is $\text{RMSE}_t = \sqrt{\sum_{h=1}^{48} \sum_{w=1}^{72} \cos(\varphi_h) \left(\hat{y}_{h,w} - y_{h,w}\right)^2}$. Here $y_{h,w}$ is the true target at grid cell (h, w) and $\hat{y}_{h,w}$ is the prediction.

Model	Core blocks	Loss / Optimiser	Epoch time
Tuned CNN	6 conv layers, dropout 0.3	MSE; Adam, lr 8×10^{-4}	$\sim$15 min
ViT	8×8 patches, 6 encoders	MSE; AdamW, lr 5×10^{-4}, wd 0.01	$\sim$4 s
Advanced MLP	Flatten $\rightarrow$ 4 FC(128)	MSE; Adam, lr 1×10^{-3}	$\sim$2 s

3 Methods

Our goal is to design a model that maps climate forcing variables (e.g., CO_2, CH_4, SO_2, BC, rsdt) to spatial predictions of surface temperature (tas) and precipitation (pr). This is formulated as a supervised regression task where the objective is to minimize the Mean Squared Error (MSE) between the predicted and ground truth maps. The model takes temporally indexed climate forcing variables as input and outputs corresponding spatial fields for tas and pr.

To explore the design space, we implemented three distinct neural network architectures: Convolutional Neural Networks (CNNs), Vision Transformers (ViTs), and Multi-Layer Perceptrons (MLPs). The CNNs leverage 2D convolutional layers to exploit spatial locality and translation invariance, which makes them suitable for spatial data. The Vision Transformer introduces a patch-based encoding and multi-head self-attention, allowing the model to capture long-range dependencies across the map. Finally, the MLP architecture flattens input features and relies on dense layers, offering computational simplicity and fast training while still achieving strong performance.



All models are trained using the Adam optimizer with a fixed learning rate and dropout for regularization. We compare these architectures in subsequent sections and analyze their trade-offs in terms of performance, scalability, and generalization.

4 Experiments

4.1 Baselines

To establish a performance benchmark for climate emulation, we adopted the SimpleCNN provided in the starter code as our baseline model. The architecture comprises a stack of 2D convolutional layers, each followed by ReLU activations and batch normalization, enabling the model to learn spatial features from the multi-channel climate input maps. We trained the model using the Mean Squared Error (MSE) loss function. These processed feature maps are subsequently projected into the target prediction maps of surface temperature (`tas`) and precipitation rate (`pr`).

We trained the model using the Adam optimizer with a fixed learning rate of 1×10^{-3}, a batch size of 64, and a dropout rate of 0.1 to mitigate overfitting. We evaluated the model with 10 and 30 epochs to observe the effect of extended training on performance saturation. Each training epoch took approximately 15 minutes on an RTX 3070 Ti GPU with 8GB of VRAM, which defined our hardware and memory constraints. These settings were chosen to match the default configuration in the starter code while ensuring reproducibility and comparability with more advanced models.

This baseline architecture served as a stable foundation, but its limited depth and capacity led to early saturation in validation performance. This behavior indicated underfitting, which we later addressed through deeper CNN variants, transformer-based architectures, and MLPs. Detailed validation results of both the 10-epoch and 30-epoch variants are shown in Table 1 and Table 2.

Although the model demonstrated a moderate ability to fit the data, it quickly saturated in performance, with performance shown in Table 1 & 2.

Table 1: CNN Baseline Validation Performance Metrics (10 Epochs)

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	4.6929	3.4156	-	1.1408
pr	2.6764	0.8260	-	1.4429

Table 2: CNN Baseline Validation Performance Metrics (30 Epochs)

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	2.6200	1.5575	-	0.7563
pr	2.0799	0.4725	-	0.9136

4.2 Experiment With ViT

We implemented a Vision Transformer (ViT) to model spatial dependencies in climate data. The model takes multi-channel input maps such as the CO_2, SO_2, CH_4), splits them into 6×9 then patches using a Conv2d layer with kernel and stride of 8, and projects each patch into a 256-dim embedding with added positional encodings.

The core of the model consists of 6 TransformerEncoderLayers, each with 8 attention heads and an MLP ratio of 4. The final output is passed through two linear layers with ReLU and Dropout, then reshaped to the original spatial resolution. Training used AdamW with a learning rate of 5×10^{-4}, weight decay 0.01, dropout 0.1, and 32-bit precision over 100 epochs.

As shown in Table 3 below, ViT performance improved steadily with more training, achieving RMSE of 1.41 for `tas` and 1.99 for `pr` at 100 epochs, which is better than the CNN baseline. Each epoch took around 3–5 seconds on our hardware.

We also defined an ImprovedVisionTransformer class to support future enhancements, with all hyperparameters configurable via YAML.

Table 3: ViT Validation Performance Metrics Across Epochs

Variable	Metric	10 Epochs	30 Epochs	100 Epochs
`tas`	RMSE	2.2758	1.7249	1.4119
	Time-Mean RMSE	1.5163	0.9853	0.5908
	Time-Stddev	-	-	-
	MAE	0.6017	0.3891	0.2833
`pr`	RMSE	2.0790	2.0169	1.9929
	Time-Mean RMSE	0.4006	0.3157	0.2585
	Time-Stddev	-	-	-
	MAE	0.8468	0.7220	0.7689

4.3 Experiment With MLP

While convolutional models such as CNNs demonstrated reasonable predictive capacity, they incurred significant training time, with each epoch requiring approximately 15 minutes on an RTX 3070 Ti GPU. Given our resource constraints and the need to iterate efficiently, we turned to Multi-Layer Perceptrons (MLPs) for their faster training dynamics. MLPs offered competitive performance with reduced compute demands and better scalability. Moreover, transformer-based models such as ViT were constrained by GPU memory during training, which further motivated the use of MLPs as a practical yet effective alternative.

We developed and evaluated two fully connected MLP architectures to predict global surface temperature (tas) and precipitation (pr) fields from gridded climate forcing variables. Both models accept flattened global input vectors composed of 5 variables (CO_2, SO_2, CH_4, BC, rsdt) over a 48×72 grid (i.e., 17,280 input dimensions), and output spatial predictions across the same grid for two target variables. We trained the model using the Mean Squared Error (MSE) loss function.

The **baseline MLP** consists of 4 hidden layers with 128 units each, ReLU activations, and a dropout rate of 0.1. It was trained using the Adam optimizer with a learning rate of 1×10^{-3}, batch size 64, and 32-bit precision for 1000 epochs. This model already showed strong performance and training stability while requiring relatively low training time (few secs per epoch), see in Table 4.

To further improve generalization and training speed, we implemented an **advanced MLP** with refined hyperparameters. We reduced the dropout rate to 0.05 to avoid underfitting, used 64-bit precision to increase numerical stability, and limited training to 500 epochs to reduce overfitting. All other architectural components remained the same.

The advanced MLP achieved the best validation performance among all tested models, with performance shown in Table 5. It also required fewer GPU resources compared to ViT and trained significantly faster, making it our final model choice.

Table 4: MLP Validation Performance Metrics

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	1.4065	0.7449	-	0.1627
pr	1.9434	0.2775	-	0.7648

Table 5: Validation Metrics and Public Leaderboard Score

Metric	**Score**
Validation Loss (MSE)	0.1515
Validation pr RMSE	1.9386
Validation pr Time-Mean RMSE	0.2398
Validation pr Time-Stddev MAE	0.7233
Validation tas RMSE	1.2978
Validation tas Time-Mean RMSE	0.4289
Validation tas Time-Stddev MAE	0.2055
Public Leaderboard Test Score	**0.7739**

4.4 Evaluation

For performance evaluation, we employed multiple metrics to assess model accuracy and robustness:

- **Root Mean Squared Error (RMSE):** Measures the overall deviation between predicted and ground truth values across all spatial points and time steps. Lower RMSE indicates better accuracy.
- **Time-Mean RMSE:** RMSE computed after averaging predictions and targets over the temporal axis, highlighting spatial prediction quality.
- **Time-Stddev RMSE / MAE:** Measures the model's ability to capture temporal variability (standard deviation across time) at each spatial location.
- **Mean Absolute Error (MAE):** Complements RMSE by assessing the average absolute deviation, less sensitive to large errors.

4.5 Implementation details

Table 6: Reproducible Hyperparameter Settings for CNN Baselines

Model	Ep.	BS	LR	Drop.	Init	Dpth	K	Prec.	Dev.	T/Ep.
CNN (10 ep)	10	64	1e-3	0.1	64	4	3	32-bit	RTX 3070ti lp	∼15min
CNN (30 ep)	30	64	1e-3	0.1	64	4	3	32-bit	RTX 3070ti lp	∼15min

Table 7: Reproducible Hyperparameter Settings for MLP Models

Model	Ep.	BS	LR	Drop.	Dim	Dpth	Act.	Prec.	Dev.	T/Ep.
MLP (baseline)	1000	64	1e-3	0.1	128	4	ReLU	32-bit	RTX 3070 Ti lp	∼2s
MLP (advanced)	500	64	1e-3	0.05	128	4	ReLU	64-bit	RTX 3070 Ti lp	∼2s

The rationale behind the parameter tuning of different models is provided in the corresponding sections above.

4.6 Results

See in Table 8.

Across various design iterations, we observed steady performance improvements as we refined our model architectures and training strategies. Our initial baselines using `SimpleCNN` architectures demonstrated moderate performance, but they quickly saturated in learning capacity, especially as we increased the number of epochs. Transitioning to a Modified CNN and then to a fully connected `MLP` led to noticeable gains in both validation loss and RMSE/MAE across `tas` and `pr`. The best-performing model was the Advanced MLP, which leveraged deeper layers and regularization techniques like dropout, yielding the lowest validation loss (0.1515) and significantly improved time-mean and time-std metrics. As a comparison, our experiment with Vision Transformer (ViT) offered promising performance with better generalization on `pr` metrics and more stable time-mean RMSE (e.g., `val/pr/time_mean_rmse = 0.3157`), but it did not outperform the MLP in overall validation loss or `tas` metrics—potentially due to limited training time and resource constraints.

One of the main challenges we faced was the limitation of local hardware, particularly our 8GB GPU, which constrained batch size, model complexity, and training speed. This limitation prevented us from fully exploring larger architectures like Vision Transformers or more advanced CNN variants. To address this, we plan to migrate our training pipeline to cloud-based platforms such as AWS, which would allow us to scale up both compute and memory resources. Leveraging cloud GPUs would enable us to perform longer training runs, larger batch experiments, and hyperparameter tuning more efficiently, ultimately pushing model performance further in future iterations.

Table 8: Validation Metrics for Different Models

Model	val loss	val/pr rmse	val/pr time_mean rmse	val/pr time_std mae	val/tas rmse	val/tas time_mean rmse	val/tas time_std mae
CNN Baseline 10 Ep	0.3165	2.6764	0.8260	1.4429	4.6929	3.4156	1.1408
CNN Baseline 30 Ep	0.1811	2.0799	0.4726	0.9136	2.6200	1.5575	0.7564
MLP	0.1526	1.9434	0.2775	0.7648	1.4056	0.7449	0.1627
Advanced MLP (Final Prediction Model)	0.1515	1.9386	0.2397	0.7233	1.2978	0.4289	0.2055
ViT (10 Ep)	-	2.0790	0.4006	0.8468	2.2758	1.5163	0.6017
ViT (30 Ep)	-	2.0169	0.3157	0.7220	1.7249	0.9853	0.3891
ViT (100 Ep)	-	1.9929	0.2585	0.7689	1.4119	0.5908	0.2833

4.7 Ablations

To understand the contribution of individual design choices to our final performance, we conducted several controlled ablation studies. Starting from the baseline CNN, we modified the dropout rate (from 0.1 to 0.3) and batch size (from 64 to 32), which led to measurable improvements in RMSE and MAE—highlighting the role of regularization and gradient noise in training stability. In the MLP series, we evaluated the impact of precision (32-bit vs. 64-bit) and dropout (0.1 vs. 0.05) under fixed architecture and data conditions. While dropout tuning yielded marginal gains, the change in precision primarily affected training efficiency, not accuracy. Finally, we compared the best-performing MLP to ViT with comparable training settings. Although ViT generalized well, especially for precipitation, its performance was constrained by GPU memory limits, which prevented deeper tuning. These ablations confirm that careful regularization and efficient architectural choices were central to our final model's success.

5 Discussion

Our experiments demonstrate the effectiveness of deep learning architectures in approximating computationally intensive climate simulations. Through careful model comparison and tuning, we observed that while CNNs provide strong spatial modeling capacity, MLPs ultimately yielded the best performance in both accuracy and efficiency. In particular, the advanced MLP architecture achieved the lowest RMSE across both temperature and precipitation variables, benefiting from its simplicity, dropout regularization, and compatibility with constrained hardware. Interestingly, despite its theoretical advantage in capturing spatial dependencies, the Vision Transformer (ViT) did not outperform the MLP. We attribute this to GPU memory bottlenecks that limited its full training potential.

We found that dropout tuning and learning rate selection significantly affected generalization, particularly for CNNs. Smaller batch sizes and increased dropout improved CNN performance, but not sufficiently to surpass the MLP. These ablations guided us toward the more scalable and well-regularized MLP pipeline. Notably, the final MLP achieved a validation RMSE of 1.2978 on `tas` and 1.9386 on `pr`—ranking us #19 out of 83 teams overall. We had initially ranked #8 on the public leaderboard, suggesting some level of overfitting or limited generalization to unseen SSPs.

One key insight was the importance of computational efficiency. CNNs required approximately 15 minutes per epoch, while MLPs trained considerably faster with fewer GPU resources. This was crucial given our limited access to high-performance hardware. Additionally, our modular codebase enabled rapid prototyping and reproducibility, allowing us to iterate across multiple architectures with YAML-configured parameters and early stopping mechanisms.

Limitations. First, due to hardware constraints, we could not explore larger or deeper transformer models, which may have unlocked better performance. Second, our models were primarily spatial, and lacked temporal encoding—potentially ignoring important dynamics across time steps. Lastly, our preprocessing pipeline focused on coarse-grained climate variables and did not fully capture spatiotemporal correlations or multi-modal signals such as wind or pressure.

Future work. Looking ahead, we plan to migrate our training pipeline to AWS for greater scalability and to experiment with time-series architectures like RNNs or TCNs. These could enhance temporal consistency and better model long-term climate patterns. Additionally, we hope to enrich the input space with other physical variables and experiment with ensemble learning to further improve robustness. Overall, this project not only helped us explore powerful emulation models but also taught us the value of careful model selection, tuning, and engineering under practical constraints.