CSE 151B Project Milestone Report

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

1 Task Description and Exploratory Analysis

1.1 Task Description

Knowing the future climate change is important. Since the climate patterns under different emissions scenarios are affected by environmental policy and sustainable development. In this project, We are going to develop a deep learning model that emulates these simulations, predicting key climate variables with low Area-Weighted Root Mean Square Error (RMSE).

Our goal is to predict two main variables—surface temperature (tas) and precipitation (pr)—based on input forcings such as CO_2, CH_4, SO_2, black carbon (BC), and solar radiation (rsdt). The model is trained on historical scenarios (SSP126, SSP370, SSP585) and evaluated on an unseen scenario (SSP245). By learning the spatial and temporal climate responses to different emissions, our model provides a fast and computationally efficient way to estimate future climate trajectories and support climate risk analysis.

1.2 Exploratory Analysis

1.2.1 Data Size & Spacial Dimension

The provided climate dataset encompasses four different Shared Socioeconomic Pathway (SSP) scenarios (`ssp126`, `ssp245`, `ssp370`, `ssp585`), each consisting of **1,021 monthly time steps**. Among these, three scenarios (`ssp126`, `ssp370`, `ssp585`) are used for training and validation, while `ssp245` is designated for testing.

The data covers a global spatial grid of **48 latitude points** and **72 longitude points**, resulting in **3,456 spatial locations**. Additionally, the dataset includes an ensemble of **three members** (`member_id = 3`), which accounts for internal variability in climate simulations.

These dimensions provide a rich spatiotemporal structure suitable for learning relationships between climate forcings and output variables such as surface air temperature and precipitation rate.

1.2.2 Distribution of Target Variables (tas and pr)

The distributions of surface air temperature (`tas`) and precipitation rate (`pr`) were examined across all four Shared Socioeconomic Pathways (SSPs).

As shown in Figure 1, for `tas`, the distributions for `ssp126`, `ssp370`, and `ssp585` exhibit a multi-modal pattern, spanning a range from approximately **210K to 320K**. These peaks correspond to different geographical zones, such as polar, mid-latitude, and equatorial regions. As the SSP scenario progresses from lower to higher emissions (from `ssp126` to `ssp585`), a subtle rightward shift in the distribution is observed, indicating overall global warming trends. Notably, `ssp370` and `ssp585` display slightly higher frequencies in the upper temperature ranges, which aligns with their higher greenhouse gas emission assumptions.

For pr, the distributions across ssp126, ssp370, and ssp585 are heavily right-skewed, with the majority of grid points exhibiting low precipitation values (close to zero) and a long tail extending towards higher values up to **50–70 mm/day**. This skewness is characteristic of precipitation data, which is inherently sparse with occasional extreme rainfall events. Higher emission scenarios like ssp370 and ssp585 exhibit a slightly extended tail, suggesting an increase in heavy precipitation events under more extreme climate conditions.

In summary, the surface temperature (tas) distribution reflects realistic global climate variations, showing warming patterns consistent with emission intensities, while precipitation (pr) maintains its typical skewed profile with extreme values becoming more pronounced under higher emission scenarios.

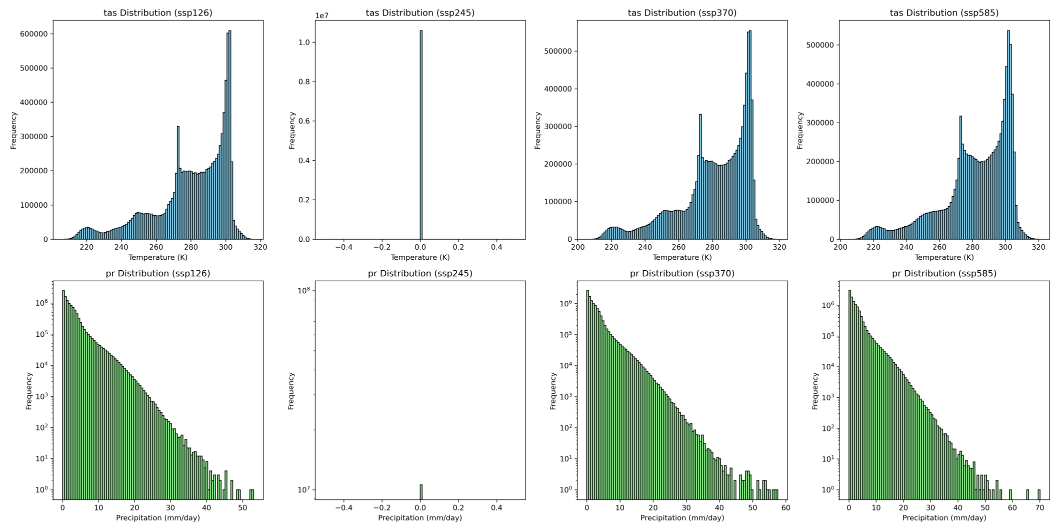


Figure 1: tas & pr Distribution.

1.2.3 Distribution of the Input Data

Figure 2 shows the distribution of each variable (CO_2, CH_4, BC, SO_2) in all SSP scenarios (ssp126, ssp370, ssp585). The input variables are first flatten to 1D vectors. The violin plots show the mean, standard deviation, and variance of each variable, with black bars indicating $+\sigma$ and $-\sigma$. Since black carbon (BC) and sodium dioxide (SO_2) have very low levels, a log-transform of the sum of its value with ϵ is performed to better visualize the distribution. All variables have a significantly large variance and multi-modality as a result of the multidimensionality. BC and SO_2 have more dimensionality, the distribution is more stratified. Methane (CH_4) and carbon dioxide (CO_2) have a less dimensionality, and the distribution is less stratified, for example, CH_4 shows two smooth peaks.

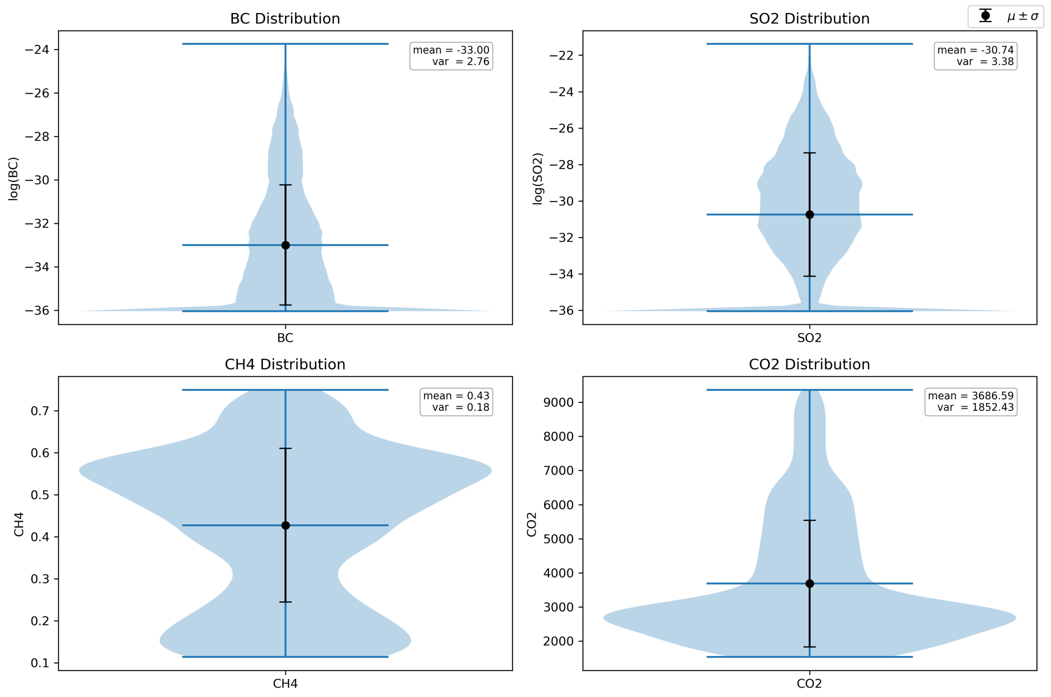


Figure 2: BC, CH_4, SO_2, CO_2 Distribution.

1.2.4 Distribution of the Input Data against Time

Figure 3 shows the trend of each variable (CO_2, CH_4, BC, SO_2) against time in all ssp (`ssp126, ssp370, ssp585`). Both BC and SO_2 show a general decline over time in all ssps. CO_2 shows a general increase in all ssps, where the cases with more emission (`ssp370 & ssp585`) have exponential growth in CO_2, which can be inferred intuitively, and, the case with less emission (`ssp126`) plateaus after around 2070. `ssp126` has decreasing CH_4 over time, while `ssp370` has a linear increasing trend and `ssp585` has an increasing trend at first but decreases after around 2060.

Figure 4 shows the change of the distribution of variables (CH_4, BC) over time in all ssps. Due to their high dimensionality, the standard deviation is plotted as the shaded region along with the global mean plotted as the solid line. While global means are decreasing for both variables, standard deviations are also decreasing for all ssps, indicating diminishing variances of the data over time. `ssp126` shows the strongest convergence of both variables, while `370` shows the weakest convergence.

1.3 Additional EDA

Figure 5 below illustrates clear global warming trends across all SSP scenarios. The left panel shows that the global mean surface temperature (`tas`) steadily increases over time, with higher emissions scenarios (`ssp370`, `ssp585`) exhibiting more pronounced warming than the low-emission scenario (`ssp126`). Similarly, the right panel indicates a gradual rise in global mean precipitation rates (`pr`), particularly under `ssp585`. The widening spread over time also suggests increasing variability and extremes, reflecting heightened climate sensitivity to greenhouse gas forcing under higher emissions scenarios. This visual evidence supports the expected outcomes of climate change models projecting stronger warming and intensified hydrological cycles under aggressive emission pathways.

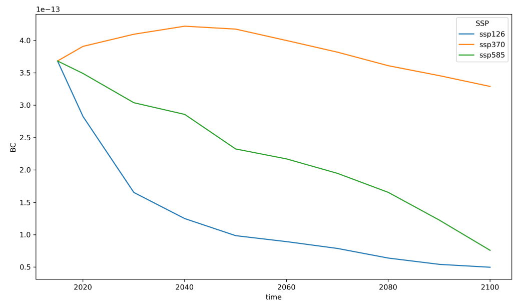


Figure 3a: BC vs. time

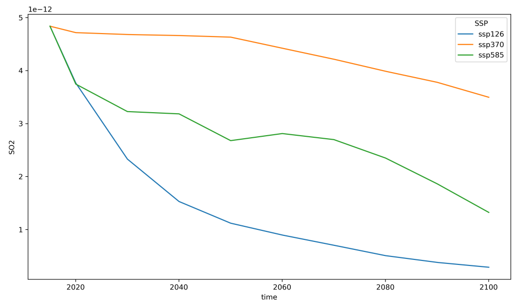


Figure 3b: SO_2 vs. time

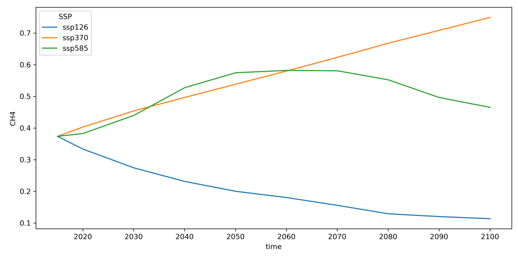


Figure 3c: CH_4 vs. time

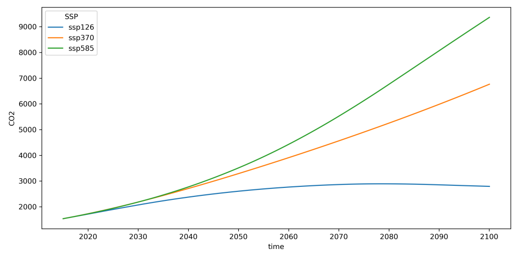


Figure 3d: CO_2 vs. time

Figure 3: Variables vs. time in `ssp126, ssp370, ssp585`

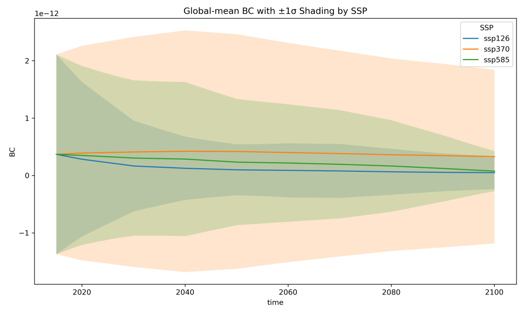


Figure 4a: BC std vs. time

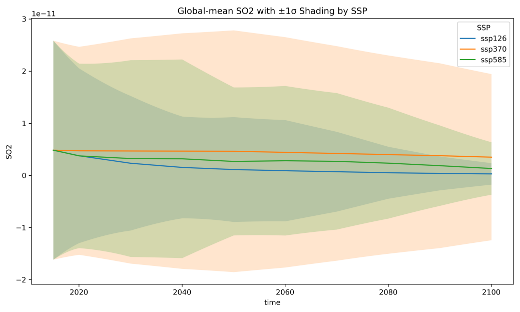


Figure 4b: SO_2 std vs. time

Figure 4: Variables std vs. time

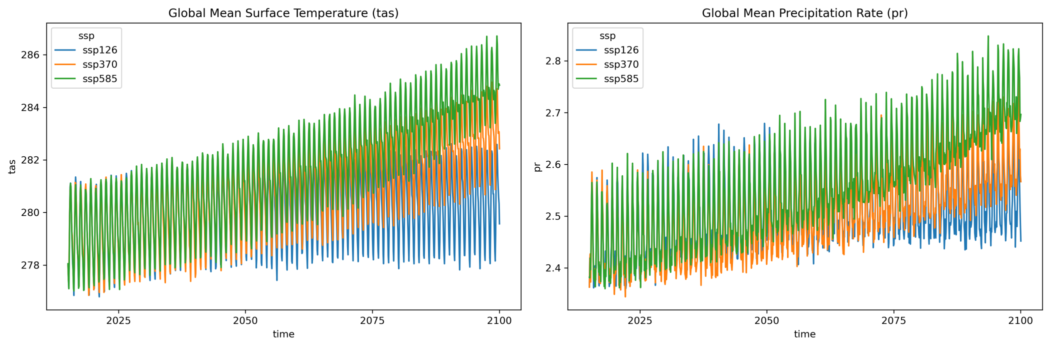


Figure 5: Global Mean `tas` & `pr`.

2 Deep Learning Model and Experiment Design

Complete detailed model implementation can be found in our git repo. See link in Appendix 4.

2.1 Baseline Model Piepline Set Up

To begin with, we used RTX 3070 Ti Laptop for training/validation. We used Adam optimizer, which is a commonly used and well-performing optimizer for deep learning tasks, especially when the dataset has complex spatial patterns as in climate data. We fixed the learning rate at 1e-3 based on common practices and baseline configurations. No advanced tuning like grid search or learning rate scheduling was performed due to time and hardware constraints. Weight decay was not explicitly applied, but we included a dropout rate of 0.1 to reduce overfitting risks. Other parameters are kept as default in starter code.

For the baseline SimpleCNN, we initially trained for 10 epochs, and later extended to 30 epochs to observe overfitting or performance saturation. Also, we fixed the batch size to 64. Each epoch takes for about 15 minutes and the result was in Table 1 & 2. Improvement occurs, but with huge time cost. So we decided to modify our CNN model.

The design choices were primarily influenced by hardware limitations (8GB GPU memory) and stability considerations, as the dataset is large and requires careful memory management. We initially tried `init_dim = 128`, but it required 16GB GPU memory, so we discarded it. dim are configurable via the YAML config file.

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

Later, we implemented a Vision Transformer (ViT) to further explore spatial modeling capacity. Our ViT model is defined in models.py and takes multi-channel climate variable maps with the dataset such as CO_2, SO_2, CH_4 ... as the input. The input is divided into non-overlapping patches using a Conv2d layer with kernel size and stride equal to the patch size. These patches are projected to 256-dim embeddings and positional embeddings are added.

We used PyTorch's built-in TransformerEncoderLayer to stack 6 attention layers, each with 8 heads and an MLP ratio of 4. The encoder output is then passed to a small head: two Linear layers with a ReLU and Dropout in between, projecting the token back to the original patch resolution. The output is reshaped to recover the full spatial resolution of the prediction map. We used the PyTorch Lightning framework for training, with mixed precision disabled by set the precision set to 32, and deterministic mode enabled to ensure reproducibility.

We started with a learning rate of 5×10^{-4}, which is commonly used for Vision Transformers. And also we monitored the validation loss and made minor adjustments during short runs by 10 and 30 epochs. As shown in Table 3 and Table 4. Since the model was not overfitting and performance improved steadily, we kept this learning rate for full training.

For weight decay and other hyperparameters, we used default values from the PyTorch Lightning and AdamW implementations, as I found they worked well in practice. I also used LearningRateMonitor to log learning rate changes during training and ModelCheckpoint to automatically save the best-performing model based on val/loss. We applied a weight decay of 0.01 and a dropout rate of 0.1, which are standard choices for Vision Transformer models. Other optimizer parameters followed the default settings in AdamW, including betas as $(0.9, 0.999)$ and epsilon as 1×10^{-8}. These values worked well in practice and did not require further tuning.

Therefore, we trained for a total of 100 epochs. After testing with 10 and 30 epochs, we found the steady improvements, so we decided to go with 100 for the best performance. Each epoch took around 3 to 5 seconds, so the training process was very fast.

We also created an ImprovedVisionTransformer subclass for potential future use. Since we will try Stochastic Depth or LayerScale later for the competition, but for this project, both models shared the same structure. All model hyperparameters such as depth, patch size, and embedding dimension are configurable to the YAML file.

Table 3: Validation Performance Metrics (10 Epochs)

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	2.2758	1.5163	-	0.6017
pr	2.0790	0.4006	-	0.8468

Table 4: Validation Performance Metrics (30 Epochs)

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	1.7249	0.9853	-	0.3891
pr	2.0169	0.3157	-	0.7220

2.2 Advanced Model Designation

2.2.1 Start With Simple CNN

To enhance the baseline SimpleCNN model's performance and efficiency, several modifications were made to the training pipeline and model configuration. First, the dropout rate was increased from 0.1 to 0.3 to introduce stronger regularization and reduce overfitting, while the batch size was reduced from 64 to 32 to accommodate the memory limitations of the RTX 3070 Ti Laptop GPU with 8 GB VRAM. Additionally, the learning rate was carefully lowered from 1×10^{-3} to 8×10^{-4} to stabilize training and promote smoother convergence over longer epochs. The precision was also adjusted from 32-bit to 16-bit mixed precision, effectively utilizing NVIDIA Tensor Cores for faster and more memory-efficient training. The total training epochs were increased from 30 to 90 to allow the model to fully benefit from these adjustments, without significant signs of overfitting, thanks to the improved regularization.

These modifications led to significant improvements in both model accuracy and speed (30 secs for each Epoch) during the training process, as shown in Table 5. The validation and test RMSE and MAE for both `tas` and `pr` showed considerable reduction compared to the baseline, indicating better generalization. Moreover, the use of mixed precision not only reduced GPU memory usage but also nearly halved the per-epoch training time, despite the smaller batch size. All design choices were based on a combination of empirical observations, best practices in CNN training, and practical considerations related to the available hardware platform and model performance bottlenecks identified during baseline evaluation.

Table 5: Modified CNN Validation Performance Metrics

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	1.5383	0.6688	0.3350	0.3350
pr	1.9620	0.3071	0.7789	0.7789

2.2.2 Advanced Model Designation

Our most complex model was a Vision Transformer (ViT), implemented from scratch in models.py. It takes multi-channel climate maps (e.g., CO_2, SO_2, CH_4) as input and splits them into 6×9 patches using a Conv2d layer with kernel and stride = 8. Each patch is embedded into a 256-dim vector, with positional encoding added. We stacked 6 layers of TransformerEncoderLayer with 8 heads each and an MLP ratio of 4. The output, see in Table 6, goes through a small prediction head with two `Linear` layers and then gets reshaped back to image format.

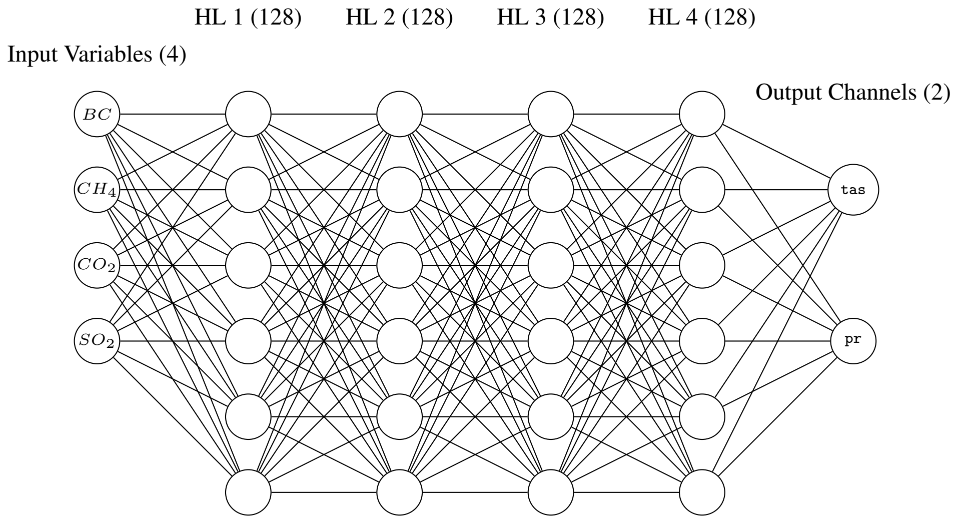
We also added an ImprovedVisionTransformer class that inherits from the base ViT for now, it's the same, but we plan to try Stochastic Depth and LayerScale later.

Table 6: Validation Performance Metrics

Variable	RMSE	Time-Mean RMSE	Time-Stddev	MAE
tas	1.4119	0.5908	-	0.2833
pr	1.9929	0.2585	-	0.7689

2.2.3 Final Model Design

The final model is a multi-layer perceptron (MLP), which flattens all variables (CO_2, CH_4, SO_2, BC) into 1D vectors. Then the vectors are passed through 4 fully connected hidden layers, each has 128 perceptrons. In between the layers, ReLU activation and 5% dropout are applied. Finally, the output will be projected to the spatial shape of `{latitude x longitude}`. The loss function is `MSEloss` and the optimizer used is `Adam`. The number of output channel is 2 (ie. `pr, tas`). The figure below shows the architecture of the MLP model, where the output represents a 3D tensor with the dimension of $outputsize \times longitude \times latitude$.



3 Experiment Results and Future Work

3.1

3.1.1 Training/Validation Metrics Visualization

As shown in Figure 6, both the training and validation loss curves show an initial rapid decrease, indicating effective early learning. The training loss stabilizes at a slightly lower level than the validation loss, which is expected, as the model sees the training data directly. The validation loss exhibits higher fluctuations and occasional spikes, suggesting the model might be encountering more variability or harder examples in the validation set. No clear overfitting trend is seen, since the

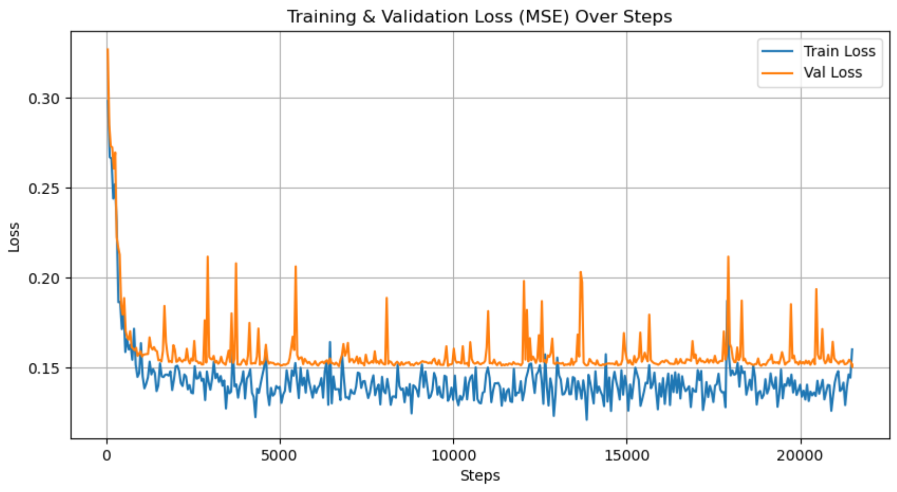


Figure 6: Training & Validation Loss (MSE) Over Steps

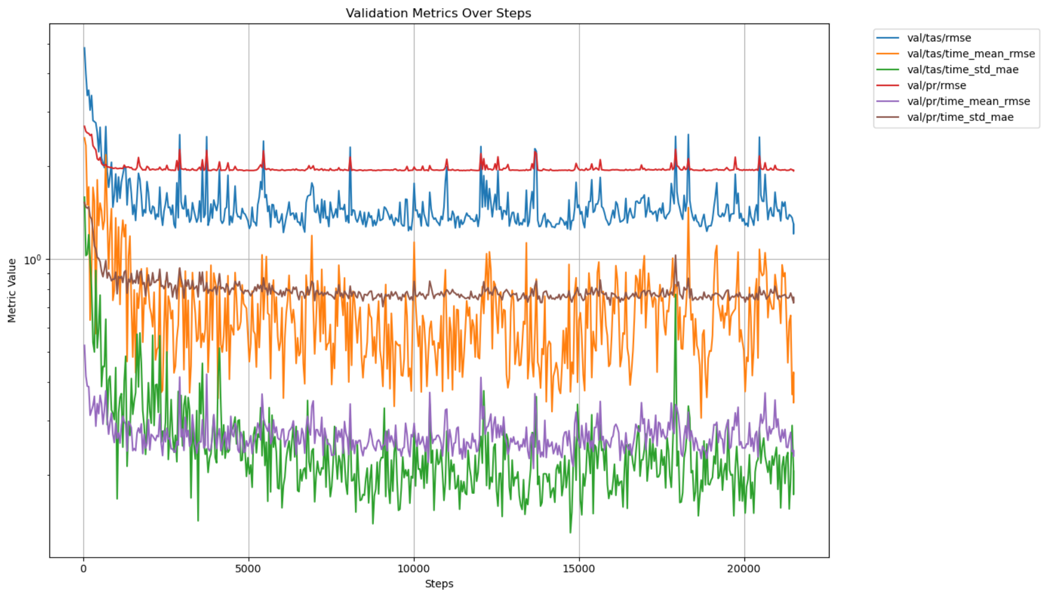


Figure 7: Validation Metrics Over Steps

validation loss does not significantly diverge upward from the training loss. In addition, Figure 7 shows the validation metrics over steps.

3.1.2 Visualization On Highest Error Samples

As shown in Figure 8 & 9 & 10 & 11 below, the largest prediction errors tend to occur in regions with sharp spatial gradients or high variability, such as the equatorial zones for precipitation (pr) and the polar regions for temperature (tas). These areas are often more dynamic and harder for the model to learn due to complex interactions and localized phenomena. The difference plots show that errors are concentrated in those high-variance zones, suggesting that the model struggles to generalize in these more extreme or fluctuating conditions.

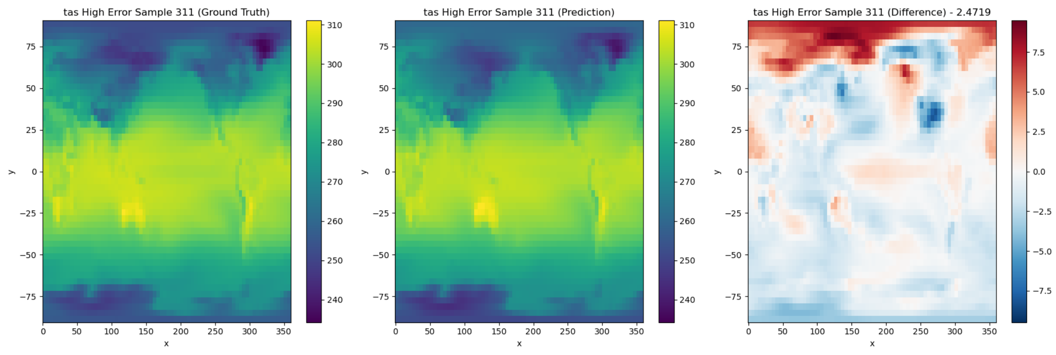

tas High Error Sample 311 (Ground Truth)
tas High Error Sample 311 (Prediction)
tas High Error Sample 311 (Difference) - 2.4719
x
y

Figure 8: Sample 311

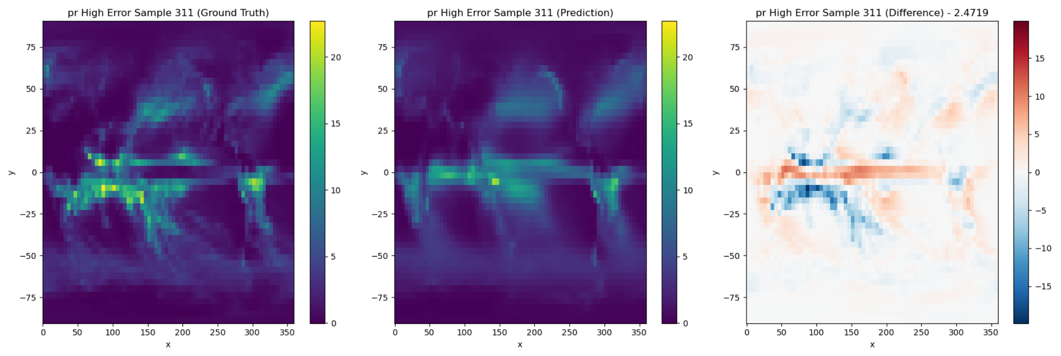

pr High Error Sample 311 (Ground Truth)
pr High Error Sample 311 (Prediction)
pr High Error Sample 311 (Difference) - 2.4719
x
y

Figure 9: Sample 311

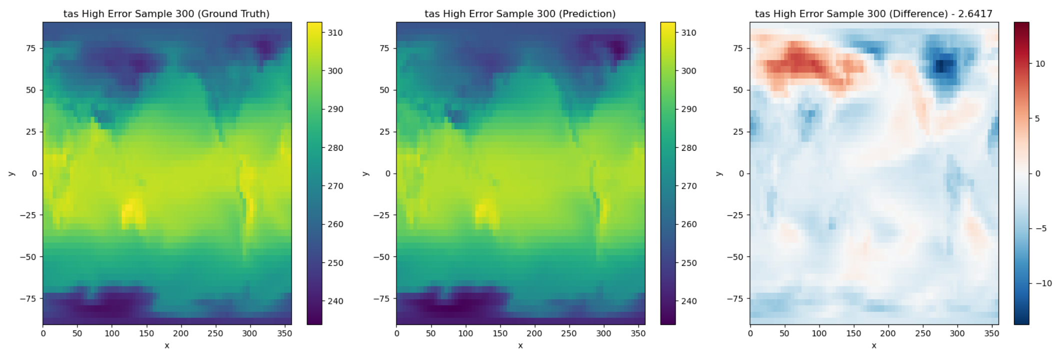

tas High Error Sample 300 (Ground Truth)
tas High Error Sample 300 (Prediction)
tas High Error Sample 300 (Difference) - 2.6417
x
y

Figure 10: Sample 300

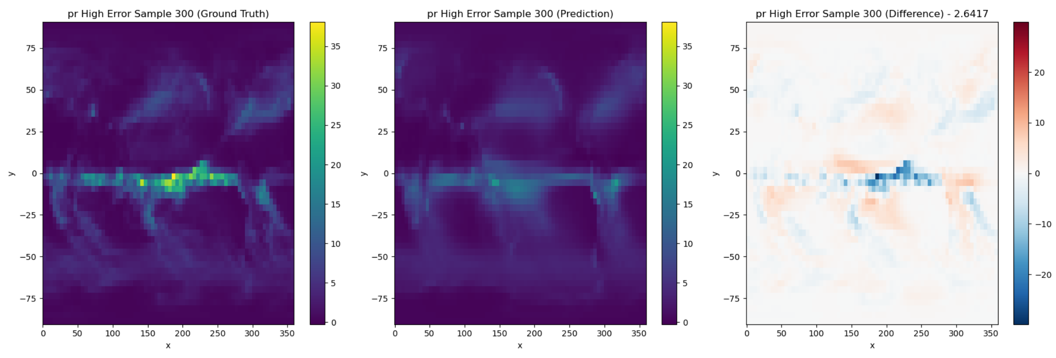


Figure 11: Sample 300

3.1.3 Public Leaderboard Test Score and Validation Scores

See in Table 7.

Table 7: 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**

3.2 Results Comparison Across Different Designs

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 #1	0.3165	2.6764	0.8260	1.4429	4.6929	3.4156	1.1408
CNN Baseline #2	0.1811	2.0799	0.4726	0.9136	2.6200	1.5575	0.7564
Modified CNN	0.1624	1.9889	0.3572	0.8115	1.9652	1.0126	0.5086
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
Vision Transformer	0.1752	2.0169	0.3157	0.8220	1.7249	0.9853	0.3891

4 Appendix: Code Repository

The complete codebase and implementation details are available at:

`https://github.com/marksui/Climate-Emulation-Model-Competition/tree/main`