
CSE 151B Project Final Report

<https://github.com/jessicabat/climate-emulation.git>

Jessica Batbayar
UC San Diego
La Jolla, CA, USA
jbatbayar@ucsd.edu

Abstract

Emulating complex physics-based climate models offers a computationally efficient alternative for projecting future climate patterns. This project presents a novel approach for climate emulation, leveraging a hybrid deep learning architecture combining a U-Net for multi-scale spatial feature learning with an FNO integrated at its bottleneck for efficient global spectral processing. Through an iterative development process, which explored pure U-Net and Fourier Neural Operator (FNO) models before moving to hybrid designs and ultimately deepening the U-Net to three downsampling steps, the model significantly improved predictive accuracy. My final 3-step hybrid model achieved strong validation performance with RMSEs of 1.3100 for surface air temperature (tas) and 1.9845 for precipitation (pr). This approach demonstrates that combining complementary deep learning paradigms, supported by meticulous engineering optimizations, can effectively capture complex climate dynamics. This solution achieved a final Kaggle private leaderboard score of 0.9362 and ranked 22nd out of 83 teams.

1 Introduction

Accurately projecting future climate is a critical global challenge. Traditional physics-based climate models, though detailed, are computationally intensive, limiting research and timely predictions. Deep learning offers a powerful alternative that allows fast, cost-effective and accurate climate emulation to inform policy, resource management, and disaster preparation [8].

The starter code provided for this project offered a fundamental framework for climate emulation, including essential data loading, pre-processing, and a simple convolutional neural network (CNN) baseline. Although functional, its straightforward architecture struggled to extract the hierarchical features necessary for high-fidelity climate predictions, thereby yielding suboptimal performance.

As shown in Table 1, my final solution directly addresses these limitations. It develops a hybrid deep learning architecture that effectively captures the complex spatiotemporal patterns essential for accurate climate emulation, leading to significantly improved predictive accuracy compared to the starter code.

My main contributions are:

- Development of a novel Hybrid U-Net [4] + FNO [1] architecture, effectively integrating multi-scale spatial feature capture with global spectral processing for state-of-the-art climate emulation.
- A crucial deepening of the U-Net component to three downsampling steps, which empirically demonstrated a significant improvement in performance beyond shallower hybrid models.

- Implementation of advanced training methodologies (e.g., OneCycleLR learning rate scheduler, gradient clipping) and resource management strategies (e.g., mixed-precision training, optimized batching) essential for efficiently training a complex, high-performing model.

Table 1: Performance Comparison: Baseline vs. Final Model (Validation Metrics)

Metric	Variable	Baseline SimpleCNN	Final Hybrid Model
Monthly Area-Weighted RMSE	tas	3.5728	1.3100
	pr	2.6090	1.9845
Decadal Mean Area-Weighted RMSE	tas	2.4045	0.4473
	pr	0.5510	0.2265
Decadal Stddev Area-Weighted MAE	tas	0.8493	0.2500
	pr	1.3607	0.7560

2 Problem Statement

This project aims to develop a deep learning model for climate emulation, learning to predict temperature and precipitation from climate forcings. The goal is to provide fast and accurate climate projections at a significantly reduced computational cost. Such high-fidelity emulations are crucial for climate research and societal preparedness.

The dataset utilized for this task originates from the Coupled Model Intercomparison Project Phase 6 (CMIP6) climate model simulations, specifically selected Shared Socioeconomic Pathways (SSPs). The data is structured across multiple dimensions: time (monthly resolution), spatial (global grid of 48 latitude and 72 longitude points), and member ID (multiple ensemble runs for each SSP), as visualized in Figure 1.

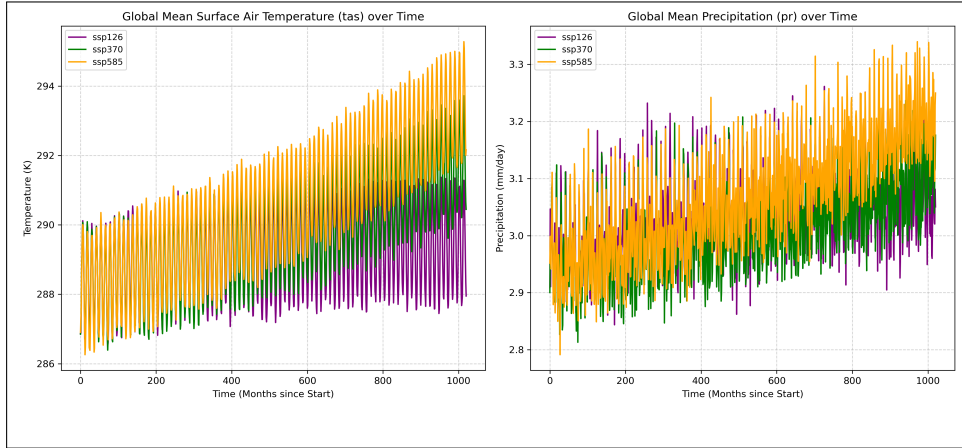


Figure 1: Time series of global mean (left) surface air temperature (tas) and (right) precipitation (pr) for the SSPs used in training (SSP126, SSP370, SSP585). These plots show how temperature and rainfall change over time in each scenario, informing the model’s learning task.

The model’s inputs, denoted as a tensor \mathbf{X} , comprise five climate forcing variables: Carbon Dioxide (CO_2), Sulfur Dioxide (SO_2), Methane (CH_4), Black Carbon (BC), and incoming solar radiation at the top of the atmosphere (rsdt). The outputs, denoted as a tensor \mathbf{Y} , consist of two key climate response variables: surface air temperature (tas, in Kelvin) and precipitation rate (pr, in mm/day). Mathematically, the input and output tensors for a given timestep are structured as $\mathbf{X} \in \mathbb{R}^{C_{in} \times H \times W}$ and $\mathbf{Y} \in \mathbb{R}^{C_{out} \times H \times W}$, where $C_{in} = 5$, $C_{out} = 2$, $H = 48$ (latitude), and $W = 72$ (longitude). Non-spatial inputs like CO_2 and CH_4 are broadcast across the spatial dimensions to align with the grid-based climate variables, ensuring consistent tensor shapes for the model.

The initial data preprocessing pipeline, including data loading, splitting strategies, and normalization techniques, was largely inherited from the provided starter code, forming a robust foundation for model development.

The dataset was systematically split to ensure robust evaluation of the model’s generalization capabilities:

- **Training Data:** Comprised data from SSP126 (low emissions), SSP585 (very high emissions), and the initial portion of SSP370 (high emissions). A single ensemble member (member ID 0) was consistently used for target variables during training.
- **Validation Data:** Consisted of the last 120 months (10 years) of SSP370. This in-domain, future-facing segment allowed for real-time monitoring of generalization without data leakage from the test set.
- **Test Data:** Composed of the last 120 months of SSP245 (intermediate emissions). This scenario was entirely unseen during training or validation, serving as a critical out-of-distribution test for the model’s ability to emulate novel future climate trajectories.

Data preprocessing involved Z-score normalization (standardization) for both input and output variables, ensuring a mean of 0 and a standard deviation of 1. Crucially, the mean and standard deviation statistics required for this normalization were computed exclusively from the training dataset to prevent any form of data leakage. These statistics were then consistently applied to normalize the validation and test datasets. During evaluation, model predictions were inverse-transformed back to their original physical units for meaningful metric calculation.

3 Methods

My goal in climate emulation is to train a model that learns to predict climate responses, like temperature and precipitation (\mathbf{Y}), directly from climate forcing inputs (\mathbf{X}). The model learns by minimizing its prediction errors during training, making its predictions closer to the true climate data.

I trained the model using the Mean Squared Error (MSE) loss function, which aims to minimize the average squared difference between predictions and the actual climate data. This loss function was provided as part of the starter code. For a given batch, the MSE loss $\mathcal{L}(\theta)$ is calculated as:

$$\mathcal{L}(\theta) = \frac{1}{B \cdot C_{out} \cdot H \cdot W} \sum_{b=1}^B \sum_{v=1}^{C_{out}} \sum_{h=1}^H \sum_{w=1}^W (\mathbf{Y}_{b,v,h,w} - \hat{\mathbf{Y}}_{b,v,h,w})^2$$

where B is the batch size. For training, I used the AdamW optimizer along with a dynamic learning rate schedule to ensure efficient and stable model convergence.

My core approach uses a novel Hybrid U-Net + FNO architecture. This model effectively combines the U-Net’s strength in handling local patterns [2] with the FNO’s ability to learn global climate relationships [7]. The final model employs a 3-step U-Net with the FNO placed at its deepest point.

3.1 U-Net Encoder

The encoder part of my hybrid model is a U-Net that captures features at different scales. It has three stages that progressively shrink the data’s size:

- Each stage uses two convolutional layers with Batch Normalization and ReLU. These layers pull out important spatial patterns.
- After each block, a Max Pooling step halves the data’s spatial size, making the features more compressed (down to $H/8 \times W/8$).
- Skip connections are established from the output of each convolutional block (prior to pooling) to their corresponding layers in the decoder, preserving fine-grained spatial information.

The channel depths progress from C_{in} to a base channel count (`base_channels=32`), then to intermediate (`mid_channels=64`), and finally to the bottleneck channel count (`bottle_channels=128`).

3.2 Fourier Neural Operator (FNO) Bottleneck

The FNO sits at the deepest point of the U-Net, serving as the model’s core. It’s powerful for understanding complex systems and works especially well for global climate patterns.

- Input features are first projected to an internal width (128 channels) using a linear layer (`nn.Linear`).
- The main FNO block uses eight layers. Each layer combines a Spectral Convolution (which processes data in the frequency domain using FFTs for global patterns) with a Pointwise Convolution (which works on local spatial details).
- These blocks include residual connections, GELU activation, and Dropout to help the model learn effectively and generalize well.
- The FFT steps within the FNO run in 32-bit precision to maintain high accuracy.

Specific FFT operations within the `SpectralConv2d` module are executed in 32-bit precision (`autocast(enabled=False)`) to maintain numerical stability.

3.3 U-Net Decoder

The decoder mirrors the encoder, taking the FNO-processed features and rebuilding the full-size climate predictions.

- It has three stages that gradually increase the data’s spatial size using Transpose Convolution layers.
- At each stage, features are combined with the high-resolution skip connections from the encoder. This brings back precise spatial details.
- After combining, two more convolutional layers, Batch Normalization, and ReLU activations further refine the features.

The channel depths revert from the bottleneck (`bottle_channels=128`) back to the intermediate (`mid_channels=64`) and base (`base_channels=32`) channel counts. A final 1x1 convolutional layer maps the features to the two output variables (tas and pr).

4 Experiments

4.1 Baselines

The experimental evaluation begins by establishing several baselines, starting from a simple convolutional model and progressively increasing architectural complexity with established deep learning paradigms. These baselines serve as crucial points of comparison to assess the effectiveness of my proposed hybrid approach.

We compare with the following baselines:

- **SimpleCNN (Starter Code):** This was a lightweight residual convolutional network provided as the competition’s starter code, designed to capture basic spatial patterns.
 - *Architecture:* It employs an initial convolution block followed by a series of 4 residual blocks [5] with increasing feature dimensions (up to 512), and a final convolutional layer to produce predictions.
 - *Key Parameters:* Configured with `init_dim=64` channels and a `depth=4` residual blocks.
- **Pure U-Net:** A classic encoder-decoder architecture, adapted to predict global climate fields. This model was recommended & chosen for its strong ability to capture features through its characteristic skip connections [3].
 - *Architecture:* Features a symmetric 4-step encoder-decoder structure, where the encoder progressively downsamples inputs (to $H/16 \times W/16$), and the decoder upsamples, leveraging skip connections to reintegrate fine-grained details.

- *Key Parameters*: The channel depths were set by features=(64, 128, 256, 512) across its four levels.
- **Pure Fourier Neural Operator (FNO)**: An advanced model designed to learn solution operators for partial differential equations, making it highly effective [6].
 - *Architecture*: It uses a linear layer to lift input features into a higher-dimensional space, followed by 8 stacked layers each performing spectral convolutions (in the Fourier domain) combined with pointwise convolutions.
 - *Key Parameters*: The model’s internal width was set to 128 channels, and modes (number of low-frequency Fourier modes kept) was 32.

4.2 Evaluation

My models and baselines were evaluated primarily based on the competition’s predefined metrics: Monthly Area-Weighted RMSE, Decadal Mean Area-Weighted RMSE, and Decadal Standard Deviation Area-Weighted MAE, calculated for both surface air temperature (tas) and precipitation (pr). These metrics assess the model’s ability to capture overall accuracy, long-term spatial patterns, and temporal variability, respectively.

The primary criterion for model selection and comparison during development was the comprehensive set of validation metrics obtained from the final epoch of each model’s training run. These validation metrics provided a reliable indicator of generalization performance on unseen data within the training SSP scenarios.

A significant challenge during the testing phase was the known corruption of test targets within the public Kaggle dataset. This made the raw test metric scores (e.g., RMSE values in the hundreds for tas) misleading and uninterpretable for direct model assessment. To ensure the integrity of predictions before submission, I implemented diagnostic print statements during the test step, inspecting normalized and raw sample predictions (`y_hat_norm`, `y_hat_raw`) and true targets (`y_true_raw`), alongside overall normalization statistics (`mean_out`, `std_out`). This allowed me to verify that the model was producing numerically sensible predictions, even though the computed test metrics were skewed by the corrupted targets. Final model performance was ultimately validated by its true score on the Kaggle private leaderboard.

4.3 Implementation details

All models were trained and evaluated on GPU-accelerated hardware using PyTorch Lightning. The final Hybrid U-Net FNO model typically trained in under 2 minutes per epoch.

Hyperparameter tuning was an iterative process, largely driven by empirical observation and the constraints of the computational platform. My primary goal was to optimize validation performance while avoiding out-of-memory (OOM) errors. I systematically adjusted several key hyperparameters across different model iterations, although a comprehensive record of all unsuccessful trials was not maintained. The differences in hyperparameters and regularization techniques across the various models explored are summarized below, highlighting the final values used for each reported experiment:

- **Optimizer and Learning Rate Schedule:**
 - *SimpleCNN and Pure U-Net*: Utilized the basic Adam optimizer with a fixed learning rate of 1×10^{-3} .
 - *Pure FNO and Hybrid Models (2-step and 3-step)*: Employed the AdamW optimizer with a weight decay of 1×10^{-4} . A OneCycleLR scheduler was used, with `max_lr` set to 1×10^{-3} (for Pure FNO) and 7×10^{-4} (for Hybrid models), `pct_start` at 0.1, and `div_factor` at 10 (for Pure FNO) and 15 (for Hybrid models). This dynamic scheduler was chosen for its proven ability to achieve faster and more stable convergence.
- **Epochs and Batch Size:**
 - *SimpleCNN and Pure U-Net*: Trained for 10 epochs with a batch size of 64.
 - *Pure FNO*: Trained for 30 epochs with a batch size of 64.

- *Hybrid (2-step)*: Trained for 30 epochs with a reduced batch size of 32. This reduction was motivated by early memory limitations encountered when combining U-Net and FNO.
- *Hybrid (3-step, Final Model)*: Trained for 40 epochs with a further reduced batch size of 16. This significant reduction was crucial to accommodate the increased architectural depth (3-step U-Net) and to prevent repeated Out-of-Memory errors.
- **Precision:**
 - *SimpleCNN, Pure U-Net, and Pure FNO*: Trained using full 32-bit precision.
 - *Hybrid Models (2-step and 3-step)*: Trained using 16-bit mixed-precision (`precision: 16`). This was a vital design choice, directly addressing GPU memory limitations that arose with more complex hybrid architectures and deeper U-Net components.
- **Regularization (Dropout and Gradient Clipping):**
 - *SimpleCNN*: Used a dropout rate of 0.2 within its `SimpleCNN` class.
 - *Pure U-Net*: No explicit dropout was applied in the provided model structure.
 - *Pure FNO*: Utilized a dropout rate of 0.1 within its FNO component.
 - *Hybrid Models (2-step and 3-step)*: Employed a dropout rate of 0.3 within their FNO components. Gradient clipping with a value of 1.0 was also applied to the trainer for these models to ensure training stability.

4.4 Results

The quantitative results from this final model, compared to the initial SimpleCNN baseline, demonstrate significant improvements across all evaluated metrics, as previously summarized in Table 1.

Result 1. The final Hybrid U-Net + FNO model achieved a validation RMSE of 1.3100 for surface air temperature (tas) and 1.9845 for precipitation (pr). Critically, the model also demonstrated strong capabilities in capturing long-term spatial patterns, with Decadal Mean Area-Weighted RMSEs of 0.4473 for tas and 0.2265 for pr. Furthermore, its ability to represent temporal variability was strong, with Decadal Stddev Area-Weighted MAEs of 0.2500 for tas and 0.7560 for pr.

Result 2. Beyond quantitative metrics, I used the code provided in the starter notebook to perform qualitative assessments by visualizing model predictions against ground truth for various climate variables and temporal characteristics. These visualizations, exemplified in Figures 2(a) and 2(b), reveal the model’s ability to capture complex spatial patterns and temporal variability.

- **Time-Mean Maps:** Visualizations of 30-year average spatial patterns for both predictions and ground truth revealed that the model successfully captured the overall climate state and large-scale geographical features, showing minimal long-term biases.
- **Time-Standard Deviation Maps:** Plots of standard deviation across time for each grid cell indicated that the model effectively reproduced the temporal variability inherent in the climate data, suggesting a good grasp of climate fluctuations.
- **Random Timestep Samples:** Direct visual comparisons of predictions against ground truth for individual months demonstrated the model’s capacity to represent fine-grained anomalies and short-term climate events with reasonable accuracy.

Result 3. Despite the overall strong performance, a persistent challenge was observed in consistently reducing the validation RMSE for precipitation (pr). While the model achieved a respectable pr RMSE of 1.9845, its improvement across various experimental iterations was less pronounced and more incremental compared to the substantial gains seen for tas. This suggests that predicting precipitation, a highly complex and noisy variable, remains a more challenging aspect of climate emulation for this architecture.

4.5 Ablations

In this section, we analyze the contribution of key architectural design choices to the final model’s performance. My ablation study focuses on the impact of increasing the U-Net’s depth within its hybrid architecture.

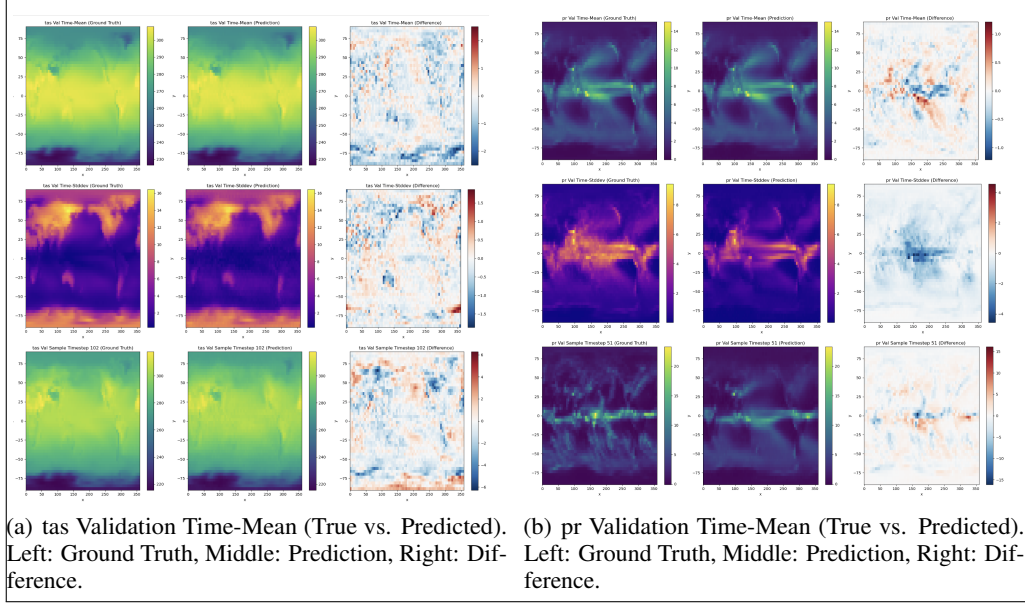


Figure 2: Qualitative Comparison.

My final model incorporates a 3-step U-Net encoder-decoder within its hybrid architecture, an increase from the 2-step U-Net used in the initial hybrid iteration. This architectural modification was a critical decision, made to enhance the model’s capacity for deeper multi-scale feature learning and capture even more complex hierarchical climate patterns.

To isolate the contribution of this depth increase, we directly compare the performance of the initial 2-step hybrid model with the final 3-step hybrid model. Both models utilized the same core hybrid U-Net FNO design and were trained with optimized hyperparameters (as detailed in Section 4.3) to ensure a fair comparison.

Table 2: Ablation Study: Impact of U-Net Depth on Validation Performance

Metric	Variable	2-Step	3-Step, Final Model	Difference(RMSE/MAE)
RMSE	tas	1.4513	1.3100	-0.1413
	pr	1.9878	1.9845	-0.0033
Time-Mean RMSE	tas	0.5114	0.4473	-0.0641
	pr	0.2242	0.2265	+0.0023
Time-Stddev MAE	tas	0.2635	0.2500	-0.0135
	pr	0.7655	0.7560	-0.0095

Through the above Table 2, this ablation confirms that architectural depth plays a crucial role in enhancing the model’s ability to emulate temperature patterns, likely due to a more comprehensive capture of global climate representations.

5 Discussion

My project successfully developed a high-performing hybrid deep learning model for climate emulation, significantly outperforming the provided SimpleCNN baseline and establishing a competitive solution on the Kaggle leaderboard. The iterative development process, detailed in Section 4, revealed key insights into designing effective climate emulators. The core achievement lies in combining the U-Net’s strength in handling spatial features with the FNO’s efficiency in modeling global dynamics, leading to a remarkable reduction in prediction error.

Throughout this journey, several techniques proved particularly effective. The strategic deepening of the U-Net architecture from two to three downsampling steps emerged as a critical design choice, delivering substantial performance gains, particularly for surface air temperature (tas) prediction. This underscored the importance of ensuring sufficient model capacity to capture the complex, hierarchical nature of climate patterns. Furthermore, efficient hyperparameter tuning, including the adoption of the OneCycleLR scheduler and careful selection of learning rates, played a vital role in achieving stable and rapid model convergence. A surprising finding was the significant improvement in tas RMSE when architectural depth was increased, initially with the goal of improving precipitation (pr) predictions. The entire development process was notably augmented by insights and architectural recommendations from LLMs, which helped explore diverse model paradigms and deepen understanding.

For deep learning beginners tackling similar prediction tasks, my experience suggests prioritizing early and thorough data exploration to grasp its characteristics before extensive coding.

Limitations Despite the achieved success, several limitations were encountered that constrained further exploration and optimal performance. A recurring challenge was persistent Out-of-Memory (OOM) errors, which severely restricted the ability to test larger model architectures (e.g., increasing channels or FNO modes beyond current configurations) or explore more complex designs without significant sacrifices in batch size. Coupled with this, the project was frequently hampered by disk quota limitations on the computational platform, necessitating the repeated deletion of training logs and intermediate test results, impacting comprehensive record-keeping. A notable performance limitation was the lack of significant improvement in precipitation (pr) prediction, as discussed in detail in Section 4.4. Furthermore, a late project start limited the initial fundamental exploration of data and methods, potentially hindering more creative and thoughtful customizations (e.g., alternative normalization schemes or custom loss functions) beyond the provided starter code’s framework.

Future work Building upon the insights gained, several promising avenues for future work can be explored to further enhance climate emulation capabilities, assuming access to more robust computational resources:

- **Targeted Precipitation (pr) Modeling:** Investigate specialized techniques for pr prediction, such as incorporating quantile loss functions to better handle its skewed distribution, or designing multi-headed architectures with branches specifically optimized for pr’s unique characteristics.
- **Explicit Temporal Modeling:** Integrate explicit temporal components, such as recurrent neural networks (RNNs) or Transformers along the time dimension, into the hybrid architecture. This could allow the model to learn time-dependencies and potentially improve predictions for longer sequences.
- **Automated Hyperparameter Optimization:** Utilize advanced tools for automated hyperparameter search (e.g., Optuna, Weights & Biases Sweeps) to explore larger parameter spaces more efficiently than manual tuning allowed.

6 Contributions

This project was developed and completed as a sole individual effort. All aspects of data preprocessing, model architecture design, implementation, training, evaluation, analysis, and report writing were performed independently.

References

- [1] CAMLab ETHZ. Tutorial 05 - Operator Learning - Fourier Neural Operator. https://github.com/camlab-ethz/AI_Science_Engineering/blob/main/Tutorial%2005%20-%20Operator%20Learning%20-%20Fourier%20Neural%20Operator.ipynb, 2024.
- [2] Qimin Deng, Peirong Lu, Shuyun Zhao, and Naiming Yuan. U-net: A deep-learning method for improving summer precipitation forecasts in china. *Atmospheric and Oceanic Science Letters*, 16(4):100322, 2023. ISSN 1674-2834. doi:<https://doi.org/10.1016/j.aosl.2022.100322>. URL <https://www.sciencedirect.com/science/article/pii/S1674283422002100>. Special Issue: Machine Learning Applications for Atmospheric and Oceanic Sciences.
- [3] Esri. How u-net works. <https://developers.arcgis.com/python/latest/guide/how-unet-works/>, n.d.
- [4] GeeksforGeeks. U-Net Architecture Explained. <https://www.geeksforgeeks.org/machine-learning/u-net-architecture-explained/>, 2025.
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL <https://arxiv.org/abs/1512.03385>.
- [6] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.
- [7] Jaideep Pathak, Karthik Kashinath, Sanat K. Prasad, and Thorsten et al. Kurth. Modeling Earth’s Atmosphere with Spherical Fourier Neural Operators. NVIDIA Developer Blog, November 2023. URL <https://developer.nvidia.com/blog/modeling-earths-atmosphere-with-spherical-fourier-neural-operators/>.
- [8] Minn Lin Wong and Manmeet Singh. Developing an urban temperature emulator using physics-based climate models and machine learning. In *ICLR 2025 Workshop on Tackling Climate Change with Machine Learning*, 2025. URL <https://www.climatechange.ai/papers/iclr2025/79>.