Benchmarking Spherical Fourier Neural Operators for Regional Weather Forecasting

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfilment of the requirements for the BS-MS Dual Degree Programme

by

Vishnu Hanumant Kadam

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

May, 2026

Supervisor: Dr. Bedartha Gowsami

Certificate

This is to certify that this dissertation entitled "Benchmarking Spherical Fourier Neural Operators for Regional Weather Forecasting" towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents work carried out by **Vishnu Hanumant Kadam** at Indian Institute of Science Education and Research under the supervision of Dr. Bedartha Goswami, Department of Data Science, during the academic year 2025-2026.

Dr. Bedartha Goswami

Committee:
Dr. Bedartha Goswami
Dr. Arnab Mukherjee

Declaration

I hereby declare that the matter embodied in the report entitled **Benchmarking Spherical Fourier Neural Operators for Regional Weather Forecasting** are the results of the work carried out by me at the Department of Data Science, Indian Institute of Science Education & Research (IISER) Pune, under the supervision of Dr. Bedartha Goswami, and the same has not been submitted elsewhere for any other degree. Wherever others contribute, every effort is made to indicate this clearly, with due reference to the literature and acknowledgement of collaborative research and discussions.

Vishnu Kadam

Vishnu Hanumant Kadam
20211222

Abstract

Modern deep learning weather prediction models excel in prediction speed as well as accuracy. Among these Spherical Fourier Neural Operators have emerged as a promising architecture that implicitly models underlying atmospheric physics using Spherical Harmonics. However, implementations of such architecture are primarily evaluated on global benchmarks, while region specific analysis is left unexplored.

The thesis focuses on investigating the reproducibility and performance of SFNO based ForeCastNet model on both Global as well as Indian Subcontinent data. Experiments are conducted on two spatial resolutions, and the resulting model dynamics, stability and accuracy are analyzed. A complete training and evaluation pipeline is developed, which includes data preprocessing, dataset construction, model training, rollout forecasting and evaluations across multiple variables.

Model performance is evaluated using standard meteorological verification metrics such as Root Mean Square Error (RMSE) and Anomaly Correlation Coefficient (ACC) across multiple forecast lead times.

The results provide insights into the strengths and limitations of neural operator-based forecasting models and contribute toward improving reproducibility and regional evaluation in scientific machine learning for weather prediction.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Bedartha Goswami for his continuous guidance, support, and encouragement throughout the course of this work. Their insights and suggestions have been invaluable in shaping this thesis.

I would like to acknowledge my friends and peers for their constant support, helpful discussions, and motivation during this journey.

Finally, I am deeply grateful to my family for their unwavering support, understanding, and encouragement, without which this work would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Importance of Weather Prediction

Accurate weather prediction plays an important role in modern society, influencing decision making across sectors such as agriculture, disaster management, aviation, energy production, and water resource planning. Reliable forecasts allow governments and organizations to anticipate extreme weather events such as cyclones, floods, droughts, and heatwaves, thereby enabling timely mitigation strategies and reducing potential economic and human losses. Sectors such as agriculture and fisheries employ a large population of people in India. Such sectors are largely climate dependent and the impacts of adverse weather can cause a massive ripple effect on the entire Indian subcontinent.

Indian monsoon system is responsible for the majority of rainfall patterns in south asia. Variations in yearly monsoon patterns such as its onset, duration and severity can affect crop yields, water availability and food security. Accurate forecasts are therefore important for planning, scheduling and managing agricultural and various other tasks. Improved cyclone tracking also helps in preparations and evacuations along coastal regions.

Industries such as aviation need accurate forecasts for avoiding turbulences and safely routing traffic. Renewable energy infrastructure such as wind and solar need ample wind and sunlight for optimal power generations. Water management is essential for hydrological reservoir operation and control.

The importance of weather prediction motivates the evaluation of modern deep learning forecasting systems. Deep learning has started to provide a faster and in some cases accurate or comparable results to traditional numerical weather predictions systems. Understanding the performance of such modern systems on specific regions such as Indian subcontinent forms the core motivation of this thesis.

1.2 Machine Learning for Weather Forecasting

Numerical weather prediction (NWP), formulates weather forecasting as a physics-driven problem, Machine learning (ML) on the other hand adopts a data-centric approach. Instead of explicitly solving the defining equations for various weather systems, deep learning weather prediction (DLWP) models learn statistical mappings from historical states to their future states. In this paradigm, equal importance is given to both model architecture and dataset quality, quantity and diversity.

The availability of large-scale, high quality data in the form of reanalysis datasets like ERA5[ERA, 2018], along with advances in deep learning architectures including attention mechanisms, transformers, graph neural networks, and neural operators has been the leading cause of the success of modern DLWP models. These models are able to learn complex spatiotemporal dependencies across multiple atmospheric variables and vertical levels, capturing large scale as well as small scale interactions without explicitly solving physical equations.

A prime motivation for using deep learning for weather prediction is computation efficiency during inference. Although training DLWP models can require considerable computational resources, but once trained, forecasting generally involves only a simple forward pass through the network. This makes the prediction faster and less compute intensive compared to traditional NWP systems, which require repeated numerical integration of partial differential equations (PDEs) over large grids. DLWP forecasts can be easily generated on a single A100 gpu without need for a specialized supercomputer.

Many deep learning models are released as open source implementations, enabling reproducibility and further research. Even in the cases where only architectural details are available, independent research groups have successfully implemented such models. The access to the codes of these models is essential for innovation in the field.

However, an major limitation of deep learning approaches is the unavailability of explicit uncertainty quantification. Deterministic deep learning models usually produce a single output without any confidence intervals. Ensemble based methods are one way to overcome this obstacle, another way to measure model reliability is to use metrics like Root Mean Square Error (RMSE), Anomaly Correlation Coefficient (ACC), and Power Spectrum, etc. Long autoregressive rollout are also use assess error accumulation. Prediction quality is therefore mostly judged by performance on such quantitative benchmarks, which are being used as a proxy for real world forecasting skill.

As deep learning architectures continue to improve in efficiency and scalability, their computational requirements are expected to decrease further. With increasing hardware accessibility and algorithmic optimization, machine learning-based weather forecasting models are likely to become more widely deployable, complementing and potentially augmenting traditional physics-based forecasting systems.

1.3 Motivation and Research Questions

The metric driven nature of deep learning has combined with the dominance of large western and eastern technology companies in the development of DLWP models. This has lead to an emphasis of globally averaged evaluation metrics as a standard for comparison. While such benchmarks allow for standardized comparisons, they may overshow region specific performance comparisons. Weather dynamics over Indian subcontinent are governed by complex and distinct processes. In global performance evaluations these local dynamics contribute incrementally. A poor performance of a specific model may go unnoticed. Despite this, many large-scale deep learning models have not been evaluated over isolated regions.

This thesis aim to implement, train and evaluated Spherical Fourier Neural Operator based FourCastNet model over Indian subcontinent to assess how well the model generalizes over specific regions. To further investigate the importance of spatial resolution on model predictive performance, experiments are conducted on two different resolutions. This enables us to examine the training dynamics, and the relations between hyperparameter ranges across different resolutions and trade-offs present.

Another important motivation is the need for reproducible and accessible data. Many state of the art DLWP models are developed in a proprietary or semi-proprietary research ecosystems. Sometimes architectural details are published, but the training pipeline is kept secret. The FourCastNet model was trained using the Nvidia makani library [nvi, 2025]. Such frameworks, are powerful but mask away the core model code across different abstraction layers and dependencies. This limits the ability of researchers to perform modification in the models thereby stifling creativity. To address this issue, this thesis aims to follow a modular approach, all the necessary training components are implemented in standard pytorch. The objective is to keep only core model components from the original codebase and remove all non-essential infrastructure. This stripped down implementation increases transparency, and simplifies and facilitates future extensions.

1.4 Contributions of this Thesis

The main contributions of this thesis are as follows:

- A systematic evaluation of the FourCastNet [Bonev et al., 2023] weather foundation model on the Indian subcontinent, a region whose forecasting characteristics are strongly influenced by complex atmospheric processes.
- An empirical study of model performance across two different spatial resolutions, enabling an analysis of how resolution influences forecasting accuracy, error accumulation, and computational requirements.

- The development of a simplified and modular PyTorch-based implementation of the FourCastNet training pipeline, removing dependencies on specialized or company-specific frameworks while preserving the essential architectural components required for training and inference.
- The design of an efficient data loading and preprocessing pipeline for large ERA5 datasets, enabling scalable training and evaluation on high–dimensional climate data.
- A detailed evaluation of forecasting performance using standard meteorological metrics such as Root Mean Square Error (RMSE) and Anomaly Correlation Coefficient (ACC), along with multi-step rollout experiments to assess long-horizon error propagation.
- An analysis of the computational trade-offs between resolution, model complexity, and training cost, providing insight into the practical deployment of deep learning weather forecasting models.

1.5 Organization of the Thesis

This thesis is organized into five chapters.

Chapter 1 introduces the background and motivation for the study. It discusses the limitations of traditional numerical weather prediction methods, the emergence of machine learning approaches for weather forecasting, and the research questions addressed in this work.

Chapter 2 provides a review of related work in numerical weather prediction and deep learning based forecasting methods. It also discusses foundational models for weather prediction, including architectures based on transformers, graph neural networks, and neural operators.

Chapter 3 describes the methodology used in this thesis. It details the datasets, preprocessing procedures, model architecture, training procedures, and evaluation metrics used for the experiments.

Chapter 4 presents the experimental results and analysis. The forecasting performance of the model is evaluated across different resolutions, and the impact of resolution scaling, training strategies, and computational cost is analyzed.

Finally, Chapter 5 summarizes the key findings of the thesis and discusses potential directions for future research in deep learning-based weather forecasting.

Chapter 2

Literature Review

2.1 Deep Learning Approaches to Weather Forecasting

Recent advances in deep learning have significantly improved data-driven weather forecasting. Early approaches relied on convolutional architectures operating on gridded reanalysis data such as ERA5. More recently, graph-based architectures such as *GraphCast* and neural operator models such as *FourCastNet* have demonstrated state-of-the-art performance. This section reviews feed-forward networks, convolutional neural networks (CNNs), graph neural networks (GNNs), and neural operators, culminating in models such as GraphCast and FourCastNet, which are central to modern data-driven forecasting.

2.1.1 Feed Forward Networks

Feed-forward neural networks (FFNs), also known as multilayer perceptrons (MLPs), are the simplest class of neural networks. A feed-forward network is constructed by stacking linear transformations followed by nonlinear activation functions such as the Rectified Linear Unit (ReLU) .

Given an input vector $x \in \mathbb{R}^d$, a single hidden layer network computes

$$h = \sigma(W_1 x + b_1) \tag{2.1}$$

$$\hat{y} = W_2 h + b_2 \tag{2.2}$$

where W_1, W_2 are weight matrices, b_1, b_2 are bias vectors, and $\sigma(\cdot)$ is a nonlinear activation function such as ReLU

$$\sigma(z) = \max(0, z). \tag{2.3}$$

For regression tasks, the final layer is typically linear, while for classification tasks a softmax activation is used

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}, \tag{2.4}$$

which produces a probability distribution over classes.

Although FFNs can approximate arbitrary continuous functions (Universal Approximation Theorem), they do not explicitly encode spatial or temporal structure. In weather forecasting, they have been used for localized prediction tasks where spatial structure is either limited or pre-processed into feature vectors. However, their lack of spatial inductive bias makes them inefficient for large-scale atmospheric modeling.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were designed to operate on structured grid-like data such as images. A convolution layer applies a learnable kernel $K \in \mathbb{R}^{n \times n}$ across the spatial domain of an input tensor.

For a 2D input field X, the convolution operation is defined as

$$Y_{i,j} = \sum_{m=1}^{n} \sum_{n=1}^{n} K_{m,n} \, X_{i+m,j+n}. \tag{2.5}$$

The same kernel parameters are shared across all spatial locations, giving CNNs translation equivariance and significantly reducing parameter count compared to fully connected networks.

As depth increases, CNNs hierarchically extract features ranging from low-level gradients and edges to higher-level semantic structures. In atmospheric modeling, meteorological variables (e.g., geopotential height, temperature, wind components) are arranged on latitude–longitude grids. Each variable is treated as a channel, similar to RGB channels in images.

Formally, if the atmospheric state at time t is represented as

$$X_t \in \mathbb{R}^{C \times H \times W}, \tag{2.6}$$

where C denotes variables and vertical levels, CNNs learn spatial mappings

$$X_{t+1} = f_\theta(X_t). \tag{2.7}$$

To incorporate temporal dependencies, CNNs are often combined with recurrent architectures such as Long Short-Term Memory (LSTM) networks. In CNN-LSTM models,

the CNN extracts spatial features

$$F_t = \text{CNN}(X_t) \tag{2.8}$$

which are then passed into an LSTM to model temporal evolution

$$h_t = \text{LSTM}(F_t, h_{t-1}) \tag{2.9}$$

$$\hat{X}_{t+1} = W h_t + b. \tag{2.10}$$

Here, the CNN captures spatial correlations while the LSTM models temporal dynamics. Such hybrid models have been used for short-term precipitation forecasting and regional weather prediction.

However, CNNs are inherently limited by their reliance on fixed-resolution grids. Changing resolution requires retraining or interpolation, and spherical geometry introduces distortions at high latitudes.

2.1.3 Graph Neural Networks

Graph Neural Networks (GNNs) were developed to process data represented as graphs, where inputs consist of nodes and edges rather than fixed grids. A graph is defined as

$$G = (V, E) \tag{2.11}$$

where V is a set of nodes and E is a set of edges. Each node i has features h_i.

Traditional neural architectures assume fixed-size structured inputs. While images can be resized without significant structural change, graphs may have variable numbers of nodes and irregular connectivity. CNNs cannot directly operate on such irregular domains.

GNNs address this by performing message passing over graph connectivity. A general graph convolution layer is expressed as

$$h_i^{(l+1)} = \sigma\left(W^{(l)} h_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} U^{(l)} h_j^{(l)}\right). \tag{2.12}$$

where $h_i^{(l)}$ is the representation of node i at layer l, $\mathcal{N}(i)$ denotes neighboring nodes, $W^{(l)}, U^{(l)}$ are learnable weight matrices, c_{ij} is a normalization factor, and σ is a nonlinear activation.

In matrix form, using adjacency matrix A and degree matrix D, a common formulation (GCN layer) is

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right) \tag{2.13}$$

where $\tilde{A} = A + I$ adds self-loops.

This formulation allows the same model to process graphs of different sizes because computation is local and defined per node. The adjacency structure encodes geometry instead of relying on a fixed grid.

GraphCast

GraphCast is a medium-range global weather forecasting model based on graph neural networks [Lam et al., 2023]. It represents the atmosphere on an icosahedral mesh to avoid polar distortions.

GraphCast uses an encoder–processor–decoder architecture. Given two previous atmospheric states X_{t-6h}, X_t the model predicts

$$\hat{X}_{t+6h} = f_\theta(X_{t-6h}, X_t). \tag{2.14}$$

The encoder maps gridded ERA5 data to graph node embeddings. The processor applies multiple message-passing layers to propagate information globally. The decoder maps node embeddings back to physical variables on the grid.

GraphCast demonstrated skill comparable to operational numerical weather prediction systems on many metrics, including lower RMSE at 500 hPa geopotential height and improved anomaly correlation coefficients across multiple lead times.

2.2 Neural Operator Models and Fourier Neural Operators

Traditional neural networks learn a mapping between finite dimensional vectors,

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^m. \tag{2.15}$$

However, in context of neural operators these problems are assumed to be **function-to-function mappings**. The model attempts to learn the evolution operator

$$\mathcal{G} : u(t, \mathbf{x}) \to u(t + \Delta t, \mathbf{x}) \tag{2.16}$$

where $u(t, \mathbf{x}) \in \mathbb{R}^C$ represents a set of atmospheric variables (such as wind velocity, temperature, or geopotential height) at spatial location $\mathbf{x}$ and time t.

Instead of learning mappings between vectors, **neural operators** learn mappings between functions:

$$\mathcal{G}_\theta : \mathcal{U} \to \mathcal{V} \tag{2.17}$$

where $\mathcal{U}$ and $\mathcal{V}$ are infinite dimensional function spaces. This formulation allows the model to approximate the solution operator of partial differential equations governing atmospheric dynamics.

2.2.1 Neural Operator Formulation

Let the input field be

$$u(x) : D \to \mathbb{R}^d \tag{2.18}$$

defined on a spatial domain D.

A neural operator is typically expressed as

$$(\mathcal{K}u)(x) = \int_D \kappa(x, y)u(y)\, dy \tag{2.19}$$

where $\kappa(x, y)$ is a learnable kernel.

The full neural operator layer becomes

$$u_{l+1}(x) = \sigma\left(W u_l(x) + \int_D \kappa(x, y)u_l(y)\, dy\right) \tag{2.20}$$

where W is a pointwise linear transformation and σ is a nonlinearity.

The integral term allows long range spatial interactions between locations x and y, which is essential for modeling global atmospheric dynamics. However, computing this integral directly is expensive for large spatial grids. This motivates the use of Fourier transforms.

2.2.2 Fourier Neural Operator (FNO)

The key idea of the Fourier Neural Operator is that convolution in spatial space corresponds to multiplication in Fourier space.

Let

$$\hat{u}(k) = \mathcal{F}(u(x)) \tag{2.21}$$

denote the Fourier transform of the input field.

The kernel convolution

$$\int_D \kappa(x, y)u(y)\, dy \tag{2.22}$$

can be expressed in Fourier space as

$$\hat{v}(k) = R(k)\hat{u}(k) \tag{2.23}$$

where $R(k)$ is a learnable complex valued matrix applied to each Fourier mode.

The algorithm for one Fourier layer is

$$u_{l+1} = \sigma\left(Wu_l + \mathcal{F}^{-1}\left(R \cdot \mathcal{F}(u_l)\right)\right) \tag{2.24}$$

where R is applied only to the lowest K frequency modes. Higher frequency modes are truncated in order to reduce computational cost.

For a grid with N spatial points, the computational complexity becomes

$$O(N \log N) \tag{2.25}$$

due to the Fast Fourier Transform. This makes FNO significantly more efficient than explicit kernel integration.

2.2.3 FourCastNet

FourCastNet is a global weather forecasting model based on the Fourier Neural Operator architecture.

Adaptive Fourier Neural Operator (AFNO)

The original FourCastNet model used the **Adaptive Fourier Neural Operator**. The Adaptive Fourier Neural Operator modifies the standard FNO by introducing **learnable sparsity in Fourier space**.

Let, $\hat{u}(k)$ be the Fourier transformed feature map.

Instead of applying a dense matrix $R(k)$ to all modes, AFNO performs a block-wise transformation

$$\hat{v}(k) = M(k)\hat{u}(k) \tag{2.26}$$

where $M(k)$ is a block diagonal matrix.

Additionally, soft thresholding is applied

$$\hat{v}(k) = \text{softshrink}(M(k)\hat{u}(k), \lambda) \tag{2.27}$$

where λ is a learnable threshold parameter.

The AFNO layer therefore becomes

$$u_{l+1} = \sigma\left(Wu_l + \mathcal{F}^{-1}\Big(\text{AFNO}(\mathcal{F}(u_l))\Big)\right). \tag{2.28}$$

2.2.4 Spherical Fourier Neural Operator (SFNO)

Standard Fourier transforms assume a Euclidean grid. However, the Earth is a sphere. Applying a standard two dimensional Fourier transform on latitude–longitude grids introduces distortion near the poles.

The **Spherical Fourier Neural Operator (SFNO)** addresses this by performing spectral operations using spherical harmonics.

Any function on the sphere can be represented as

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} a_{l,m} Y_l^m(\theta, \phi) \tag{2.29}$$

where Y_l^m are spherical harmonic basis functions.

The coefficients are computed using the spherical harmonic transform

$$a_{l,m} = \int_{S^2} f(\theta, \phi) \overline{Y_l^m(\theta, \phi)} d\Omega. \tag{2.30}$$

The spectral operator becomes

$$\hat{v}_{l,m} = R_{l,m} \hat{u}_{l,m} \tag{2.31}$$

where $R_{l,m}$ is a learnable transformation applied to each spherical harmonic mode. The spatial field is then reconstructed using the inverse transform $u(x) = \mathcal{S}^{-1}(\hat{v})$.

Neural operator models represent a new class of architectures that learn operators acting on continuous functions. The progression of models can be summarized as

$$\text{FNO} \rightarrow \text{AFNO} \rightarrow \text{SFNO}. \tag{2.32}$$

Each modification addresses a limitation of the previous approach:

- FNO introduces efficient global convolution using Fourier transforms.
- AFNO introduces sparsity and block-wise spectral learning.
- SFNO adapts the operator to spherical geometry of the Earth.

These architectures form the foundation of modern deep learning based weather forecasting systems.

2.3 Reproducibility in Scientific Machine Learning

A significant number of recent machine learning publications do not release the complete resources required to reproduce their results. While papers may describe model architectures and high-level training procedures, critical components such as training scripts,

preprocessing pipelines, trained model weights, or exact dataset configurations are often unavailable. In some cases, datasets themselves may be partially proprietary or require extensive preprocessing steps that are not fully documented.

Another major barrier to reproducibility arises from the computational scale at which many modern models are trained. Large foundation models and state-of-the- art forecasting systems are frequently trained on massive distributed clusters containing dozens or hundreds of high-end GPUs. For example, the original FourCastNet model was trained using multiple NVIDIA DGX systems, each containing several GPUs. Such infrastructure is beyond the reach of many academic institutions and individual researchers. This is why reproducing exact results from original work becomes infeasible.

Even with these challenges reproducibility remains an important factor for validation an experiment. In the cases when exact replication is not possible due to large trainig costs, a small scale experiment is necessary that implements the core methodology and demonstrates similar predictive skills.

This thesis partly focuses on contributing to improving reproducibility in data-driven weather forecasting. All training scripts, preprocessing pipelines, and model hyperparameters used in this work are publicly available. The trained model weights and experimental setup are documented in detail, enabling reproduction of the experiments presented here using commonly available computational resources. While the original FourCastNet model was trained on large multi-node GPU clusters, this work demonstrates that similar architectures can be trained and evaluated within more modest computational budgets while still providing meaningful insight into model behavior.

2.4 Research Gap

Recent advances in deep learning have demonstrated that data-driven models can achieve competitive or even superior performance compared to traditional numerical weather prediction systems. Architectures such as GraphCast and FourCastNet have shown strong forecasting skill across multiple atmospheric variables and lead times.

First, many state-of-the-art models rely on extremely large computational resources for training. This creates a barrier for independent verification of published results and limits the accessibility of these methods to a small number of institutions with large-scale GPU infrastructure. Consequently, there is a need for reproducible implementations that demonstrate how such models can be trained and evaluated using more modest computational resources.

Second, while neural operator models such as Fourier Neural Operators have been shown to capture large-scale atmospheric dynamics effectively, the behavior of these models under different regions and training configurations is not yet fully understood.

Third, many studies focus primarily on achieving state-of-the-art forecasting skill,

while comparatively less attention is given to systematic evaluation of training stability, computational efficiency, and reproducibility of the training pipeline. Understanding these practical aspects is essential for transitioning deep learning forecasting models from research prototypes to reliable scientific tools.

Chapter 3

Methodology

3.1 Problem Formulation

Weather forecasting can be formulated as a supervised learning problem in which the objective is to predict the future atmospheric state from its current state. Let the atmospheric state at time t be represented by

$$u_t \in \mathbb{R}^{C \times H \times W}, \tag{3.1}$$

where C denotes the number of meteorological variables (channels), and $H \times W$ represents the spatial grid resolution. Each tensor u_t therefore contains the values of all selected atmospheric variables at every grid point on the Earth at time t.

Given a fixed forecasting horizon Δt, the goal is to learn a function

$$\mathcal{F}_\theta : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^{C \times H \times W}, \tag{3.2}$$

parameterized by θ, such that

$$\hat{u}_{t+\Delta t} = \mathcal{F}_\theta(u_t), \tag{3.3}$$

where $\hat{u}_{t+\Delta t}$ is the predicted atmospheric state at time $t + \Delta t$.

The model parameters θ are learned from historical data by minimizing a loss function that measures the discrepancy between the predicted and true future states. A commonly used objective is the mean squared error (MSE):

$$\mathcal{L}(\theta) = \frac{1}{CHW} \sum_{CHW} (\mathcal{F}_\theta(u_t) - u_{t+\Delta t})^2. \tag{3.4}$$

In practice, the dataset consists of a sequence of atmospheric states

$$\{u_1, u_2, \ldots, u_T\},$$

from which training pairs $(u_t, u_{t+\Delta t})$ are constructed.

For multi-step forecasting, the model can be applied autoregressively:

$$\hat{u}_{t+k\Delta t} = \mathcal{F}_\theta^{(k)}(u_t), \tag{3.5}$$

where the prediction at each step is fed back as input to generate the next forecast. This introduces error accumulation, making stability and long-term consistency important considerations.

3.2 Dataset Description (ERA5)

To address the problem defined in the previous section using deep learning, we require a dataset that captures the Earth's climatology over multiple decades. Such dataset must provide consistent and continuous values of major atmospheric and surface variables at every location across the globe. However, most weather variables are observed through ground-based monitoring stations or remote sensing platforms such as satellites. These observation systems are inherently localized and unevenly distributed. Ground stations are sparse in oceans, deserts, and polar regions, while satellite measurements may be indirect or limited in temporal coverage. This spatial and temporal sparsity makes it challenging to construct a complete, globally consistent dataset directly from raw observations, thereby hindering truly global-scale forecasting.

To overcome this limitation, long-term global climatology must be reconstructed from sparse and heterogeneous observations. This reconstruction process is achieved through climate reanalysis. Climate reanalysis combines historical observational data with the output of numerical weather prediction (NWP) models through data assimilation techniques. By integrating observations into a physically consistent dynamical model of the atmosphere, reanalysis produces a temporally continuous, spatially complete, and physically coherent estimate of the Earth's past climate state.

One of the most widely used reanalysis datasets is European Centre for Medium-Range Weather Forecasts (ECMWF)'s ERA5 dataset. ERA5 provides a comprehensive, gridded representation of atmospheric, land, and oceanic variables over several decades. It is generated using advanced data assimilation systems and state-of-the-art forecast models, ensuring high physical consistency and reliability.

ERA5 is a gridded reanalysis dataset in which the Earth's surface is discretized into a regular latitude–longitude grid. Each grid point contains values of multiple meteorological variables—such as temperature, pressure, wind components, and geopotential height—corresponding to that geographic location and time step. The dataset is available at multiple spatial resolutions, allowing researchers to balance computational cost and spatial detail depending on the application.

In this thesis, two spatial resolutions of the ERA5 dataset were used:A coarse resolution of 5.6°, corresponding to a grid of 64 × 32 points.A higher resolution of 1.4°, corresponding to a grid of 240 × 121 points.

Using multiple resolutions enables systematic evaluation of model performance across different spatial scales, as well as analysis of the trade-offs between computational efficiency and predictive accuracy in deep learning-based weather forecasting models.

3.3 Preprocessing and Data Pipeline

3.3.1 Normalization

The dataset contains variables with different magnitudes and units. Without normalization, variables with large numerical ranges overtake gradient updates, leading to poor model performance. To avoid this standard z-score normalization was used.

$$z_{score} = \frac{x - \mu}{\sigma}$$

where μ and σ are computed from the training dataset. These are computed per variable per pressure level across time dimension. Normalization improves model training stability, accelerates convergence, and reduces gradient explosion risks.

3.3.2 Problems with pytorch dataloader

The ERA5 dataset used in this work is stored in Zarr format, which is designed for scalable storage and parallel access to large multidimensional arrays. In Zarr, data is divided into chunks, where each chunk represents a contiguous block of data stored in physical storage. Instead of storing the entire dataset as a single large array, chunking allows the dataset to be partitioned into smaller manageable blocks. This structure allows for efficient parallel processing as individual chunks can be accessed and processed independently without reading the entire dataset.

Xarray [Hoyer and Hamman, 2017]is a standard python library used to manipulate zarr format datasets. Xarray provides a high level interface for working with multi-dimensional arrays commonly found in climate and geospatial datasets. ERA5 dataset contains dimensions like time, latitude, longitude, and pressure levels. Dask is a parallel computing backend that integrated well with xarray package. This allows for lazy loading of large datasets.

Lazy loading is important for large scale datasets like ERA5 which can easily be in terabyte scales. Loading entire dataset into memory would require an enormous amount of RAM and would slow down preprocessing. Instead, Dask allows for data to only be loaded into memory when it is explicitly called or is used in computations. All the intermediate

operations are represented as computational graph that store the manipulations and calculations assigned to data. This deferred execution model reduces memory usage as well as unnecessary I/O operations, making it possible to work with terabyte scale datasets.

For training deep learning models, PyTorch [Paszke et al., 2019] the most popular framework used in academic as well as industrial setting. It has been widely adopted because it has extensive tools for model development, training, and distributed computations. An important component of PyTorch is its dataloader pipeline. This dataloader makes use of parallel worker processes to efficiently feed data the the training process.

However, the default architecture of this DataLoader class assumes that the underlying data is integer indexable, meaning each train sample can be called with a single index. This design works if the data to be trained is in an contiguous array or stored as files separately. While such architecture can work with zarr format dataset, it is inefficient approach as it doesn't rely on the underlying chunking present in stored data. The integer indexing here causes unnecessary computation overhead as each call for a sample need to be translated and mapped to a chunk and then to a train sample in that chunk which is the loaded into memory. The same chunk may be called again and again to create a single batch of train data.. Repeatedly, performing this leads to a significant slowing down of the training process as majority of the train time is now wasted on compiling and loading batches.

This mismatch between PyTorch's sample based indexing and Zarr's chunk based storage became a major bottleneck in the training pipeline, especially while working with high-resolution datasets. To address this issue, a more efficient batching strategy that works with zarr's chunking was required.

3.3.3 Efficient dataloader

A custom dataloader was designed to address the I/O bottleneck inspired by the producer-consumer paradigm. The key idea is to utilize the zarr chunking in the dataloading process itself, thereby minimizing the amount of disk access required.

Instead of fetching individual training samples and constructing a batch from joining them, the dataloader retrieves stores a batch directly to memory. To make this possible, a separate dataset was created from original ERA5 dataset. This new dataset contains only the variables required for training and excludes unnecessary fields present in the full reanalysis dataset. By isolating the relevant variables and storing them separately, overall storage footprint and data access complexity is reduced.

This new dataset was reordered such that each chunk now contained eight complete training examples. This chunking allows for multiple training samples to be read from disk in a single operation. The custom dataloader uses this structure by loading entire

chunks and combining them to form a training batch. The final batchsize used is 32 corresponding to 4 chunks. Sampling was performed with replacement, to ensure that sufficient data is present while training.

This strategy significantly reduces the amount of disk reads required. As each chunks contains multiple samples, the dataloader avoids slicing small pieces of data from large chunks. This producer-consumer design also allowed loading to be done asynchronously with model training, further improving throughput.

The performance improvements from this approach were noticeable in the 1.4° dataset. In this case the custom dataloader reduced the fetching time by more than 5× compared to the standard PyTorch Dataloader. This resulted in increasing training efficiency and better GPU utilization.

Algorithm 1 ERA5Prefetcher: Asynchronous Prefetching for ERA5 Training Batches

Require: ERA5 dataset D, batch size B, sequence length L, queue size Q

Require: Optional normalization statistics μ, σ

Ensure: Batches of shape (B, C, L, H, W) ready for training

Initialize queue $\mathcal{Q}$ with capacity Q
Start background worker thread
procedure WorkerThread
    Initialize random seed
    while stop flag not set **do**
        Sample batch start indices $l \sim$ Uniform(dataset length)
        Construct temporal indices: $t_{\text{idx}} \leftarrow l + [0, 1, \ldots, L-1]$
        Slice dataset: $X \leftarrow D[\text{time} = t_{\text{idx}}]$
        Rearrange to (B, C, L, H, W) layout
        Trigger computation and convert to NumPy
        if normalization enabled **then**
            Normalize: $X \leftarrow (X - \mu)/\sigma$
        end if
        if NaN checking enabled **then**
            if NaN found in X **then**
                Log problematic indices
            end if
        end if
        Convert to PyTorch tensor: $T \leftarrow$ torch.from_numpy(X)
        if CUDA available **then**
            Pin tensor to GPU memory
        end if
        Push T into queue $\mathcal{Q}$
    end while
end procedure
procedure GetBatch
    return next tensor from queue $\mathcal{Q}$
end procedure
procedure Stop
    Set stop flag
    Drain remaining tensors from queue
    Join worker thread
end procedure

3.4 Model Architecture

3.4.1 Model Architecture

The forecasting model used in this work is based on the Spherical Fourier Neural Operator (SFNO) architecture. SFNO extends the Fourier Neural Operator framework to data defined on the surface of a sphere, making it particularly suitable for global climate datasets such as ERA5. Since atmospheric variables naturally evolve on the Earth's spherical geometry, using spherical spectral transforms avoids distortions that arise from planar approximations.

The model takes as input the atmospheric state tensor

$$u_t \in \mathbb{R}^{C \times H \times W}$$

where C denotes the number of variables (channels), and $H \times W$ represents the spatial resolution of the latitude–longitude grid. The model predicts the future atmospheric state at the next time step with the same dimensionality.

The implementation used in this work is configured as follows:

```
model = SFNO(
    operator_type='driscoll-healy',
    img_size=(nlat, nlon),
    num_layers=8,
    scale_factor=2,
    embed_dim=384,
    pos_embed="latlon",
    use_mlp=True,
    activation_function="gelu",
    normalization_layer="layer_norm",
    hard_thresholding_fraction=1,
    in_chans=channels,
    out_chans=channels
)
```

The spatial resolution of the model is determined directly from the dataset. The variables *nlat* and *nlon* correspond to the number of latitude and longitude grid points respectively, ensuring that the model architecture adapts automatically to different dataset resolutions.

The parameter `operator type='driscoll-healy'` specifies the spherical harmonic transform algorithm used within the SFNO layers. The Driscoll–Healy formulation provides an efficient and numerically stable method for performing spherical harmonic transforms on equiangular latitude–longitude grids, which are commonly used in atmospheric datasets.

The model contains eight SFNO layers (`num layers=8`). Each layer performs a spectral transformation of the feature maps using spherical harmonic transforms, applies learnable weights to the spectral coefficients, and transforms the result back to physical space. Stacking multiple layers allows the network to capture increasingly complex spatial features and nonlinear interactions between atmospheric variables.

The internal feature representation uses an embedding dimension of 384 channels (`embed dim=384`). This higher-dimensional latent representation allows the model to encode complex spatial patterns and interactions across variables.

The parameter `scale factor=2` controls the channel expansion inside the network, increasing representational capacity within intermediate layers.

To provide the model with information about geographic location, latitude-longitude positional embeddings are used (`pos embed="latlon"`). These embeddings encode spatial coordinates explicitly, allowing the model to differentiate between locations such as equatorial and polar regions where atmospheric dynamics differ significantly.

Each SFNO block also includes a feedforward multilayer perceptron (MLP) component (`use mlp=True`) after the spectral operator. This component introduces additional nonlinear transformations and improves the expressive capacity of the network. The activation function used throughout the network is Gaussian Error Linear Unit (GELU) , which has been shown to perform well in transformer-like architectures and spectral models.

To stabilize training and improve gradient flow, layer normalization [Ba et al., 2016] is applied within the network (`normalization layer="layer norm"`). Layer normalization helps maintain consistent feature statistics across layers and accelerates convergence.

The parameter `hard thresholding fraction=1` indicates that all spectral modes are retained during the spherical harmonic transform, meaning no explicit truncation of frequency components is applied in this configuration.

Finally, the number of input and output channels (`in chans` and `out chans`) corresponds to the number of meteorological variables included in the dataset. The model therefore learns a mapping

$$\mathcal{F}_\theta : \mathbb{R}^{C\times H\times W} \to \mathbb{R}^{C\times H\times W}$$

which predicts the future atmospheric state for all variables simultaneously.

This architecture allows the model to capture global spatial dependencies, multi-

variable interactions, and large-scale atmospheric dynamics, making it well suited for data-driven weather forecasting tasks using global reanalysis datasets.

3.5 Data split

3.5.1 Train-test-val split

The ERA5 reanalysis dataset used in this work spans the period from 1959 to 2022 with a temporal resolution of $\Delta t = 6$ hours. Each timestep represents the global atmospheric state on a latitude–longitude grid and can be written as

$$X_t \in \mathbb{R}^{C \times H \times W},$$

where C denotes the number of atmospheric variables and $H \times W$ corresponds to the spatial grid resolution. The forecasting task consists of learning a mapping that predicts the atmospheric state at the next timestep, i.e. predicting $X_{t+\Delta t}$ from the current state X_t.

Data Quality and NaN Detection

During an initial inspection of the dataset, a small number of timesteps were found to contain missing values (NaNs). These values primarily occurred in the early years of the dataset.

Since neural networks propagate invalid values through all subsequent computations, even a single NaN in the input tensor would blow up the entire forward pass of the model. To ensure numerical stability during training, all timesteps containing NaN values were removed from the dataset.

The removal process was performed carefully to preserve temporal consistency. Because training samples are constructed as pairs of consecutive timesteps $(X_t, X_{t+\Delta t})$, any timestep containing NaNs required removal of both the corrupted timestep and its associated input pair. This ensured that all training examples correspond to valid and physically consistent temporal transitions.

Temporal Train–Validation–Test Split

Weather forecasting is inherently a time series prediction problem, and therefore the dataset cannot be randomly split as in typical machine learning tasks. Random splits would introduce information leakage from future timesteps into the training data.

Instead, a chronological split was used. The dataset was divided into training, validation, and test periods as follows:

Dataset Split	Time Period
Training	1959 – 2016
Validation	2017 – 2018
Test	2019 – 2022

Table 3.1: Chronological split of the ERA5 dataset used for training, validation, and testing.

The training set contains the majority of the data and is used to learn model parameters. The validation set is used for hyperparameter tuning and monitoring model performance during training, while the test set is reserved exclusively for final evaluation. This chronological separation ensures that the model is always evaluated on atmospheric conditions that occur strictly after the training period.

Temporal Shuffling During Training

Even if the data split saves chronological ordering, the training batches used are randomly sampled from the timesteps present in this data. A sampled input state X_t along with its corresponding target state $X_{t+\Delta t}$ forms a single train example for the initial phase one training.

Randomly sampling states is important as weather exhibits strong temporal correlations. If training batches are created squentially, the successive model gradient updates would be highly correlated and this could slow down convergence. This also improves the diversity of training data seen by the model.

Overall, the data preparation pipeline preserves chronological ordering across training, validation, and testing while still allowing randomized sampling within the training set to improve optimization efficiency.

3.6 Training Strategy and Learning rate Schedular

3.6.1 Training Strategy

The training procedure was designed to ensure stable convergence while maintaining computational efficiency across both spatial resolutions. Spectral neural operator models are known to be sensitive to optimization hyperparameters, particularly learning rate schedules and rollout training strategies. Therefore, several learning rate configurations were empirically evaluated before selecting the final training scheme.

Training Procedure for 5.6° Dataset

For the lower resolution 5.6° dataset, training was performed in four stages similar to the GraphCast training strategy. The stages are summarized in Table 3.2.

Stage	Description
Part 1	Linear Schedule for 10 epochs
Part 2	Constant LR for 10 epochs
Part 3	Autoregressive training
Part 4	Fine-tune

Table 3.2: Four-stage training strategy used for the 5.6° resolution dataset.

The staged approach stabilizes optimization during early training and gradually exposes the model to longer forecast horizons. During the autoregressive stage, the model predictions are recursively fed back as inputs, and the training loss is accumulated across all forecast steps in the rollout.

In the first phase, the model was trained for 10 epochs using a linear learning rate scheduler that gradually increased the learning rate from a small initial value to the target learning rate. This warm-up phase allowed the model to stabilize early optimization and prevented large gradient updates.

In the second phase, training continued for another 10 epochs with a reduced learning rate. This stage allowed the model to refine the learned representation while maintaining stable training dynamics.

In the third phase, autoregressive training was introduced. During this phase the rollout horizon was gradually increased from 2 forecasting steps to 15 steps. At each training iteration the model recursively generated forecasts and the loss was accumulated across all predicted timesteps. This approach encourages the model to learn stable temporal dynamics and reduces error accumulation during long-range forecasting.

The final phase consisted of regional fine-tuning. In this stage the model was trained specifically on data corresponding to the Indian region. Fine-tuning on a regional subset allows the model to adapt to localized weather patterns such as the Indian monsoon system while retaining the global dynamics learned during earlier stages of training.

Training Procedure for 1.4° Dataset

For the higher resolution 1.4° dataset, a modified training strategy was used to reduce training instability and data loading overhead.

Stage	Description
Part 1	Linear learning rate followed by cosine annealing
Part 2	Autoregressive training
Part 3	Fine-tuning on the Indian region

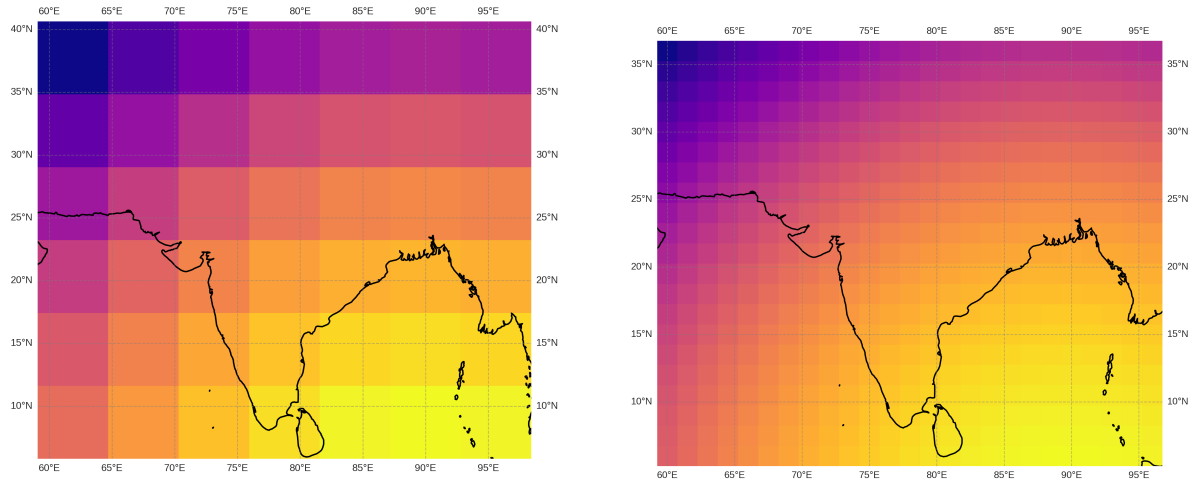
Table 3.3: Training strategy used for the 1.4° resolution dataset.

The first two phases of the GraphCast-style training procedure were combined into a single stage. In this stage, the learning rate was increased linearly for the first 1000

training steps, followed by a cosine annealing schedule for the next 1000 steps. This cycle was repeated nine times, resulting in a total of 10,000 training steps.

After this initial training stage, autoregressive training was performed similarly to the third phase of the 5.6° training procedure. The model was trained with a fixed learning rate while generating multi-step forecasts and accumulating the loss across the rollout horizon.

Finally, regional fine-tuning was performed on the Indian region. During this phase, the learning rate was kept constant and training samples were restricted to spatial domains corresponding to the Indian subcontinent. This final step improved regional forecasting performance while preserving the globally learned atmospheric dynamics.



(a) 5.6° resolution

(b) 1.4° resolution

Figure 3.1: India mask used for evaluation at different spatial resolutions. Gradient grid shows available data points over the region.

3.6.2 Learning Rate Scheduling

Phase 1 is the most important part of the training process. Most of the model train time in spent here. A low loss at phase 1 determines overall model performance. So,majority of the hyper parameters were chosen according to phase 1 performance.

Several fixed and decaying learning rate strategies were initially tested, including constant learning rates, step decay schedules, and cosine annealing schedules. Inspired by the staged training approach proposed in GraphCast, we also experimented with a multi-phase training schedule that progressively adjusted the learning rate to stabilize early training and refine later optimization.

After systematic experimentation, we adopted a two-phase learning rate schedule that provided the most stable convergence and lowest validation error. The schedule is defined as follows:

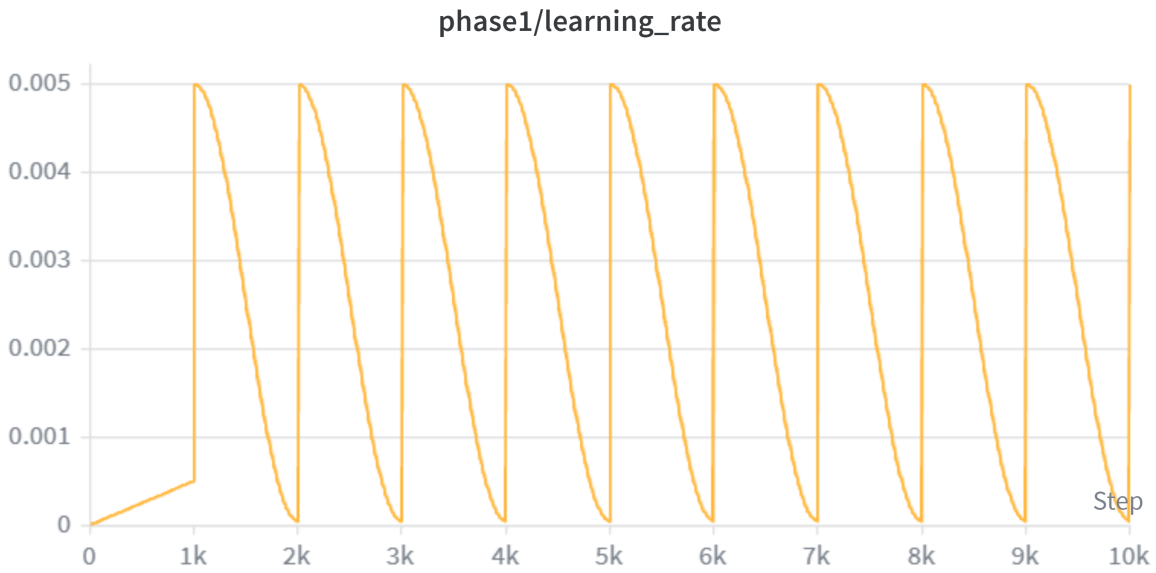


Figure 3.2: Learning rate schedule for Phase 1.

1. **Warm-up Phase (First 1000 batches):** The learning rate increases linearly from a small value to a predefined peak value. This helps in model stability, by reducing the chances of convergence to a local minimum. The incremental increase prevents the possibility of gradient explosion.

2. **Cosine Annealing Phase (Next 9000 batches):** After the initial warm-up phase, the learning rate follows a repetitive cosine annealing schedule:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{\pi(T_w)}{T_c}\right)\right), \tag{3.6}$$

   where $\eta_{\max}$ is the peak learning rate, T_w is the number of batches since the last annealing restart, and $T_c = 1000$ is the duration of the cosine decay phase. The cosine annealing part repeats 9 times in total.

The total training duration was fixed at 10,000 batches. This combination of linear warm-up followed by cosine decay resulted in smoother loss curves and improved generalization compared to fixed or abruptly decaying schedules.

3.6.3 Optimizer

The model was trained using the Adam [Ahn et al., 2024] optimizer. Adam was selected due to its adaptive moment estimation, which is well-suited for high-dimensional, non-convex optimization problems encountered in deep learning-based weather forecasting. The standard parameter settings were used:

$$\beta_1 = 0.9, \quad \beta_2 = 0.999,$$

along with standard weight decay for regularization. The use of weight decay helped prevent overfitting, particularly at higher spatial resolutions where model capacity is larger.

3.6.4 Batch Size Selection

Batch size was chosen based on a trade-off between GPU utilization, memory constraints, and data loading efficiency. Smaller batch sizes reduce memory usage but increase the relative cost of data loading and gradient updates. Larger batch sizes improve GPU throughput but may lead to longer data fetching times, especially for the 1.4° dataset.

Empirical testing showed that a batch size of 32 provided the best balance between efficient GPU utilization and acceptable dataloading latency.

Therefore, all final experiments were conducted using a batch size of 32 for both spatial resolutions.

Overall, the adopted optimization strategy—linear warm-up followed by cosine annealing with Adam and a batch size of 32—provided stable convergence, improved generalization, and efficient training across multi-resolution ERA5 datasets.

3.7 Loss Function Formulation

Selecting an appropriate loss function is critical for training deep learning models for global weather prediction. The loss function must capture both the magnitude of prediction errors and the spatial characteristics of atmospheric fields. In this work, several loss formulations were evaluated on the lower-resolution 5.6° ERA5 dataset before selecting the final objective used for training.

Mean Squared Error

The most basic loss function considered was the mean squared error (MSE), defined as

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 , \tag{3.7}$$

where y_i represents the ground truth value and $\hat{y}_i$ represents the predicted value at spatial location i, and N is the total number of spatial points. While MSE provides a straightforward measure of prediction error, it does not account for the varying physical area represented by different grid cells on the latitude–longitude grid. Near the poles,

grid cells represent smaller physical areas compared to those near the equator. As a result, treating all grid points equally introduces a bias in the loss function.

Latitude Weighted Loss

To correct for this imbalance, a latitude-weighted loss inspired by the ForCcastNet and GraphCast models was introduced. The weighting scheme approximates the true surface area of latitude–longitude grid cells. The latitude weight for a grid point at latitude θ_j is defined as

$$w(\theta_j) \approx \sin(\theta_j)\, d\theta, \tag{3.8}$$

which ensures that summation across discrete grid points approximates the continuous surface integral over the sphere.

Using these weights, the spatial loss can be written as

$$\mathcal{L}_{\text{lat}} = \frac{1}{|G|} \sum_{i \in G} a_i \left(\hat{x}_i - x_i\right)^2, \tag{3.9}$$

where G represents the spatial grid, x_i and $\hat{x}_i$ denote the ground truth and predicted values at grid location i, and a_i is the normalized area weight of that grid cell. These weights correct for the non-uniform area distribution across latitude bands.

GraphCast Training Objective

GraphCast proposes a more comprehensive training objective that aggregates errors across spatial locations, variables, and autoregressive forecast lead times. The objective minimizes the mean squared error across a sequence of T_{train} forecast steps. The loss is defined as

$$\mathcal{L}_{\text{GC}} = \frac{1}{|D_{\text{batch}}|} \sum_{d_0 \in D_{\text{batch}}} \frac{1}{T_{\text{train}}} \sum_{\tau=1}^{T_{\text{train}}} \frac{1}{|G|} \sum_{i \in G} \sum_{j \in J} s_j\, w_j\, a_i \left(\hat{x}_{i,j}^{d_0+\tau} - x_{i,j}^{d_0+\tau}\right)^2. \tag{3.10}$$

In this formulation:

$d_0 \in D_{\text{batch}}$	forecast initialization time within a training batch	(3.11)
$\tau \in [1, T_{\text{train}}]$	autoregressive forecast lead time	(3.12)
$i \in G$	spatial grid location (latitude and longitude)	(3.13)
$j \in J$	variable index and pressure level	(3.14)
$x_{i,j}^{d_0+\tau}$	ground truth atmospheric variable	(3.15)
$\hat{x}_{i,j}^{d_0+\tau}$	predicted atmospheric variable	(3.16)
a_i	area weight of grid cell i	(3.17)
w_j	per-variable loss weight	(3.18)
s_j	inverse variance scaling for variable j	(3.19)

In this work, the variance scaling term s_j was omitted for simplicity, while the spatial area weights and variable weights were retained.

Variable-Level Weighting

Different atmospheric variables exhibit different magnitudes and physical importance. To balance their contributions during training, per-variable weights w_j were introduced. Surface variables such as 2-meter temperature were assigned higher importance, while other surface variables were given smaller weights. Atmospheric variables defined at pressure levels were weighted proportionally to their pressure level:

$$w_j = \frac{p_j}{1000}, \tag{3.20}$$

where p_j represents the pressure level corresponding to variable j.

Variable	Channel Weight
zonal and meridional (u and v) wind 250	0.25
zonal and meridional (u and v) 500	0.50
zonal and meridional (u and v) 850	0.85
temperature 250	0.25
temperature 500	0.50
temperature 850	0.85
geopotential 250	0.25
geopotential 500	0.50
geopotential 850	0.85
vertical velocity 250	0.25
vertical velocity 500	0.50
vertical velocity 850	0.85
specific humidity 250	0.25
specific humidity 500	0.50
specific humidity 850	0.85
10m u component of wind	0.10
10m v component of wind	0.10
2m temperature	1.00
mean sea level pressure	0.10
total column water vapour	0.10

Table 3.4: Per-variable channel weights used in the training loss function. Atmospheric variables are weighted according to their pressure level, while surface variables use fixed weights.

Spectral Loss Component

In addition to the spatial loss, a spectral loss term [Kochkov et al., 2024] was introduced to penalize discrepancies in the frequency domain. Since the SFNO model operates in spectral space using spherical harmonic transforms, enforcing consistency in spectral coefficients helps improve the reconstruction of spatial patterns across scales.

Let $\mathcal{S}$ denote the spherical harmonic transform. The spectral representations of the predicted and target fields are

$$\hat{U}_{\ell m} = \mathcal{S}(\hat{x}), \quad U_{\ell m} = \mathcal{S}(x). \tag{3.21}$$

The spectral error is computed as

$$\mathcal{L}_{\text{spec}} = \frac{1}{N} \sum_{\ell,m} (\ell + 1)^{\alpha} \left| \hat{U}_{\ell m} - U_{\ell m} \right|, \tag{3.22}$$

where ℓ represents the spherical harmonic degree and α controls the emphasis on higher-frequency modes. Larger values of α increase the penalty on errors at smaller spatial scales.

Final Loss Function

The final training function combines the latitude weighted spatial loss with the spectral loss:

$$\mathcal{L}_{\text{final}} = \mathcal{L}_{\text{spatial}} + \lambda_{\text{spec}} \mathcal{L}_{\text{spec}}, \tag{3.23}$$

where $\lambda_{\text{spec}} = 100$ controls the relative contribution of the spectral term. The value of λ was chosen to balance the contribution of each loss term.

The spatial loss term incorporates both latitude-based area correction and variable-level weighting:

$$\mathcal{L}_{\text{spatial}} = \sum_j w_j \frac{\sum_i a_i \left(\hat{x}_{i,j} - x_{i,j}\right)^2}{\sum_i a_i}. \tag{3.24}$$

This combined loss encourages the model to minimize absolute prediction error while simultaneously preserving the spectral structure of atmospheric fields across spatial scales. Empirically, this formulation provided the most stable training and produced the best forecasting performance among the loss functions evaluated during experimentation.

3.8 Evaluation Metrics

To evaluate forecasting performance, three metrics were used: Root Mean Squared Error (RMSE), Anomaly Correlation Coefficient (ACC), and Spectral Error. These metrics measure forecast accuracy in physical space, pattern similarity between predicted and true atmospheric fields, and consistency of predictions across spatial frequencies. Together they provide a comprehensive evaluation of the forecasting skill of the model.

Root Mean Squared Error (RMSE)

Root Mean Squared Error measures the average magnitude of prediction errors across the spatial domain. Since atmospheric variables are defined on a spherical latitude–longitude grid, a latitude-dependent weighting factor is introduced to account for the varying physical area of grid cells.

The weighted RMSE for a forecast variable v at lead time l is defined as

$$\mathrm{RMSE}(v,l) = \sqrt{\frac{1}{NM}\sum_{m=1}^{M}\sum_{n=1}^{N} L(m)\left(X_{\mathrm{pred}}^{(l)}[v,m,n] - X_{\mathrm{true}}^{(l)}[v,m,n]\right)^2} \tag{3.25}$$

where $X_{\mathrm{pred}}^{(l)}[v,m,n]$ and $X_{\mathrm{true}}^{(l)}[v,m,n]$ denote the predicted and true values of variable v at grid location (m,n) and forecast lead time l. The latitude weighting factor $L(m)$ corrects for the curvature of the Earth and is defined as

$$L(m) = \frac{\cos(\mathrm{lat}(m))}{\frac{1}{N_{\mathrm{lat}}}\sum_{m=1}^{N_{\mathrm{lat}}}\cos(\mathrm{lat}(m))}. \tag{3.26}$$

This normalization ensures that the average weight across all latitudes equals one. RMSE is widely used in numerical weather prediction because it directly measures the magnitude of forecast errors and is expressed in the same physical units as the predicted variable.

Anomaly Correlation Coefficient (ACC)

While RMSE measures absolute error, it does not capture whether the spatial structure of atmospheric patterns is predicted correctly. For this reason, the Anomaly Correlation Coefficient (ACC) is also used.

ACC evaluates the similarity between predicted and true anomaly fields. Anomalies are computed by subtracting the long-term climatological mean from each variable. If $\bar{X}$ denotes the climatological mean, the anomaly field is

$$\tilde{X} = X - \bar{X}. \tag{3.27}$$

Using these anomaly fields, the weighted ACC for a variable v at forecast lead time l is defined as

$$\mathrm{ACC}(v,l) = \frac{\sum_{m,n} L(m)\,\tilde{X}_{\mathrm{pred}}^{(l)}[v,m,n]\tilde{X}_{\mathrm{true}}^{(l)}[v,m,n]}{\sqrt{\sum_{m,n} L(m)\left(\tilde{X}_{\mathrm{pred}}^{(l)}[v,m,n]\right)^2}\sqrt{\sum_{m,n} L(m)\left(\tilde{X}_{\mathrm{true}}^{(l)}[v,m,n]\right)^2}}. \tag{3.28}$$

Here $\tilde{X}_{\mathrm{pred}}$ and $\tilde{X}_{\mathrm{true}}$ denote the anomaly fields of the predicted and ground truth variables respectively. The long-term mean used to compute anomalies is estimated from historical samples in the training dataset.

ACC takes values between -1 and 1, where a value of 1 indicates perfect spatial agreement between the predicted and true anomaly fields. This metric is widely used in medium-range forecasting because it measures the similarity of large-scale atmospheric patterns such as Rossby waves and circulation structures, independent of absolute bias.

Power Spectrum Analysis

In addition to spatial evaluation metrics, the spectral characteristics of the forecasts were analyzed using power spectrum analysis. Atmospheric dynamics exhibit strong scale-dependent behavior, with different physical processes dominating at different spatial wavelengths. Examining the power spectrum therefore provides insight into how well the model preserves energy across spatial scales.

To compute the power spectrum of a predicted or ground-truth field, a two dimensional Fast Fourier Transform (FFT) was applied. Although spherical harmonic transforms provide a more exact spectral decomposition on the sphere, FFT-based methods allow consistent comparisons across different papers.

A two-dimensional Fourier transform is then applied:

$$F(k_x, k_y) = \mathcal{F}\left(X(m, n)\right), \tag{3.29}$$

where k_x and k_y denote the zonal and meridional wavenumbers respectively. The spectral power associated with each frequency component is computed as

$$P(k_x, k_y) = \frac{|F(k_x, k_y)|^2}{(N_{\text{lat}} N_{\text{lon}})^2}. \tag{3.30}$$

To obtain an isotropic power spectrum, the two–dimensional spectrum is radially averaged across shells of constant total wavenumber

$$k = \sqrt{k_x^2 + k_y^2}. \tag{3.31}$$

The total spectral power within each wavenumber shell is computed by summing the energy of all Fourier coefficients whose radial wavenumber falls within that shell:

$$P(k) = \sum_{(k_x, k_y) \in k} P(k_x, k_y). \tag{3.32}$$

This procedure produces the one–dimensional power spectrum $P(k)$, which describes how atmospheric energy is distributed across spatial scales.

Complementarity of Metrics

RMSE quantifies absolute physical error, ACC measures structural similarity of atmospheric patterns, and Power spectrum evaluates scale-dependent accuracy. Using all three ensures a balanced evaluation of magnitude, structure, and multi-scale dynamics, which is essential for assessing deep learning models in global weather forecasting.

Chapter 4

Results

4.1 Experimental Overview

Experiments were conducted at two spatial resolutions: a coarse grid of 5.6° and a higher-resolution grid of 1.4°. The evaluation focused on both qualitative and quantitative aspects of forecasting performance.

Qualitative results are presented using long rollouts over multiple lead times. The quantitave results are shown using standard evaluation metrics introduced in the previous chapter.

4.2 Performance at 5.6° Resolution

We first evaluate the performance of the SFNO model on the 5.6° resolution dataset. This lower-resolution setting allows faster training while capturing the large-scale atmospheric dynamics.

4.2.1 Forecast Evolution Across Lead Times

To qualitatively evaluate model performance, we visualize predicted atmospheric fields across multiple forecast lead times. Each figure compares the predicted field with the corresponding ERA5 ground truth.

Mean Squared Error Loss

Lead times include:

$$t \in \{6, 66, 120\} \text{ hours} \tag{4.1}$$

These lead times represent short-range to medium-range forecasts.

U10m

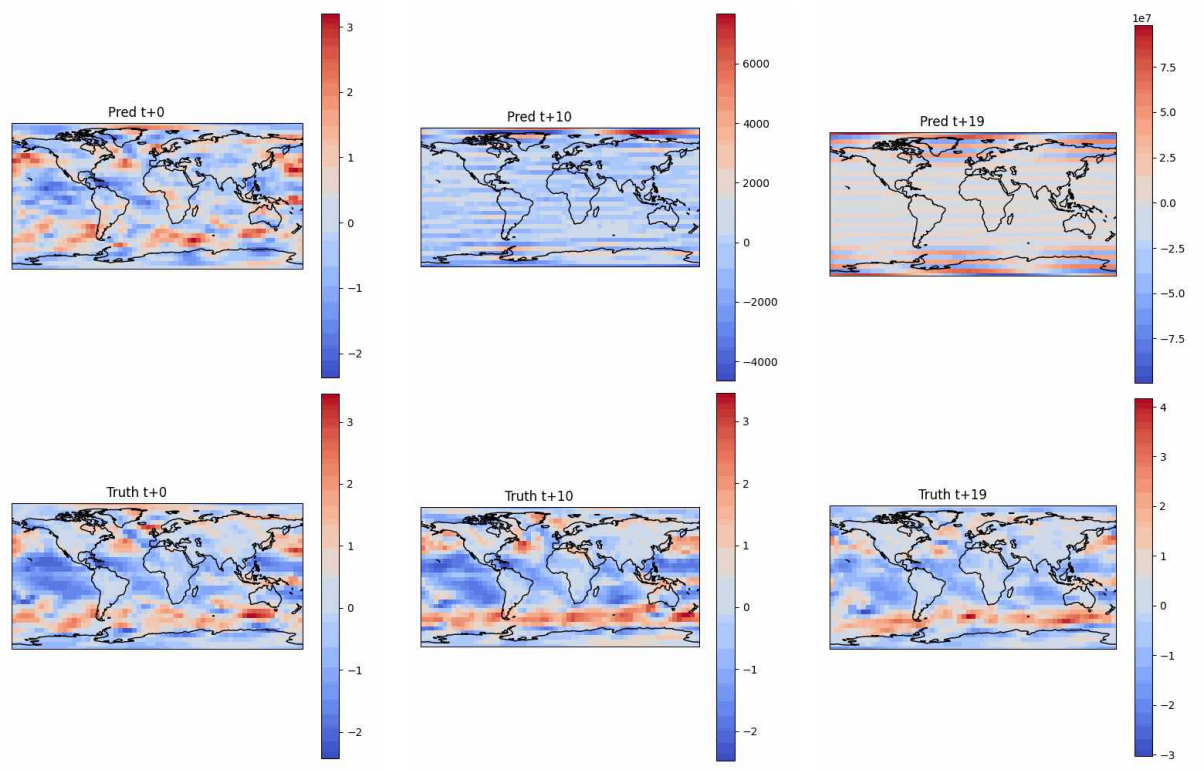


Figure 4.1: Autoregressive rollout forecasts for U10m at 5.6° resolution.

T2m

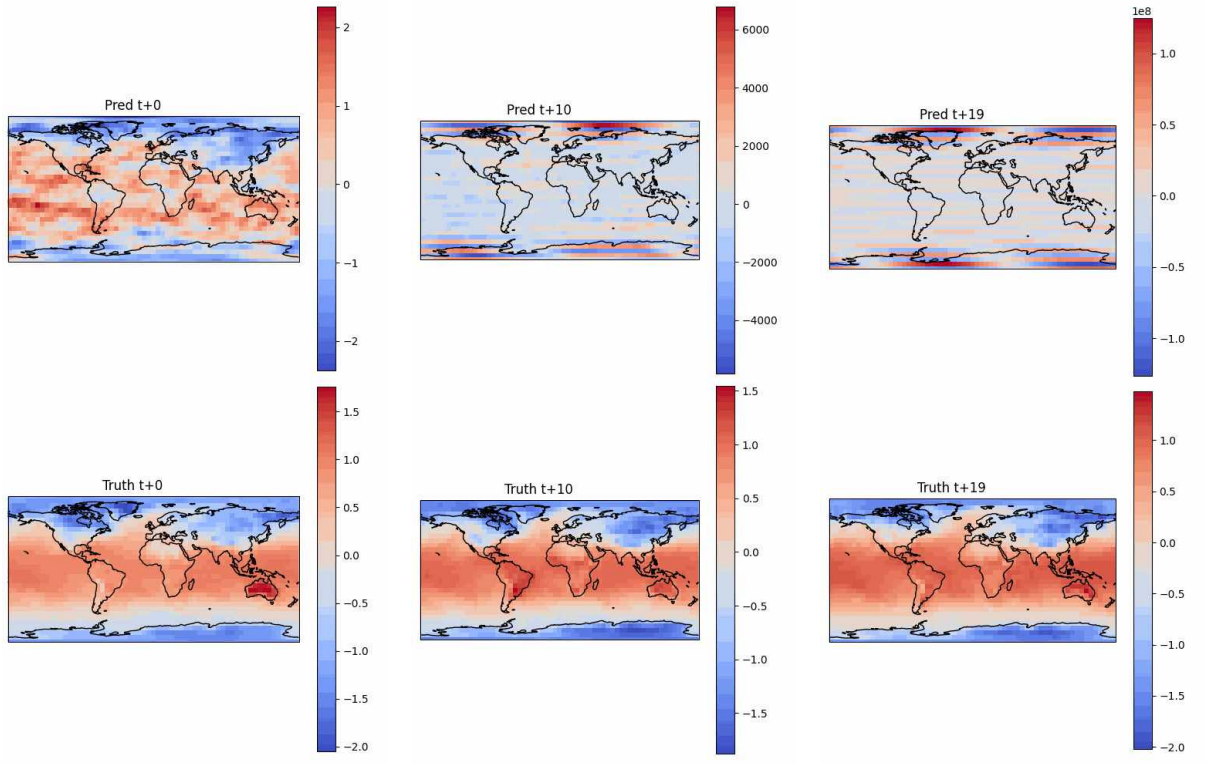


Figure 4.2: Autoregressive rollout forecasts for T2m at 5.6° resolution.

We first examine the qualitative behavior of the model forecasts at different lead times.

At a lead time of 6 hours, the model retains some descriptive capability. Although the predicted fields appear noticeably smoother than the ERA5 ground truth, the large-scale atmospheric structures are still preserved. This indicates that the model is able to capture coarse spatial patterns in the short-range forecast regime.

At longer lead times, the forecast quality deteriorates significantly. In the 66 hour forecasts, horizontal artefacts with large magnitudes begin to appear across the predicted fields. These artefacts indicate numerical instability in the autoregressive rollout, suggesting that the model predictions start to diverge from physically plausible atmospheric states.

By 120 hours, the forecasts have completely blown up, with no recognizable large-scale circulation patterns remaining. The predicted fields are dominated by spurious high-magnitude structures and lack any meaningful correspondence with the ground truth.

These results suggest that optimizing the model using only the Mean Squared Error (MSE) loss is insufficient for stable long-range forecasting in deep learning based weather models.

Latitude Weighted Squared Error Loss

Lead times include:

$$t \in \{6, 66, 120\} \text{ hours} \tag{4.2}$$

corresponding to 0.25, 2.75 and 5 days respectively.

U10m

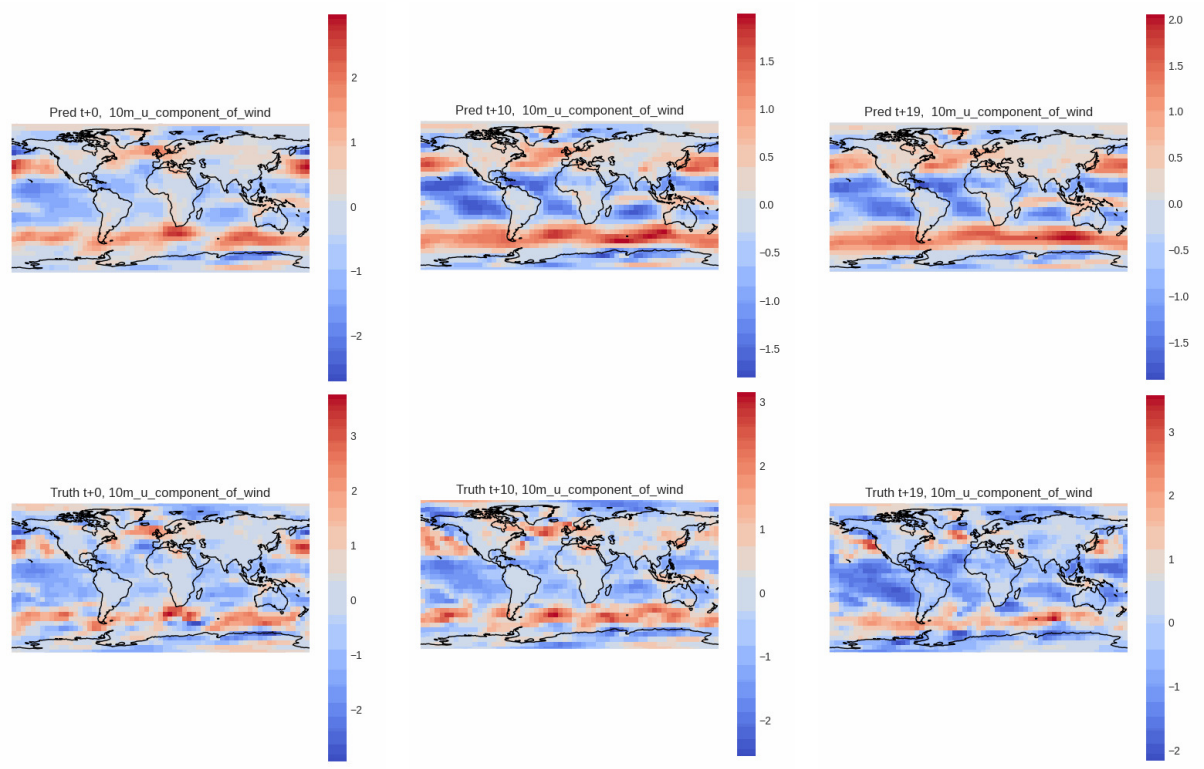


Figure 4.3: Autoregressive rollout forecasts for U10m at 5.6° resolution.

T2m

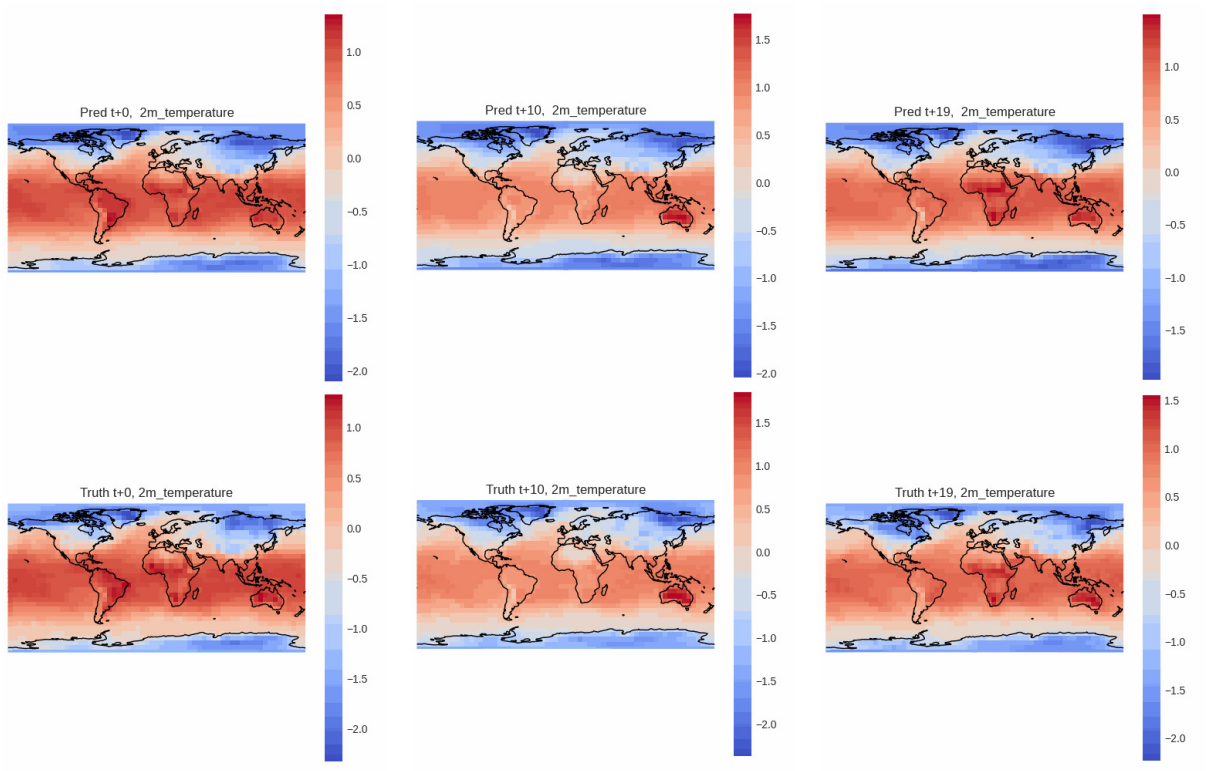


Figure 4.4: Autoregressive rollout forecasts for T2m at 5.6° resolution.

We now compare the forecasts obtained using the standard Mean Squared Error (MSE) loss with those trained using a latitude-weighted loss. The latitude weighting accounts for the varying physical area represented by grid cells on a latitude–longitude grid, assigning larger importance to regions closer to the equator.

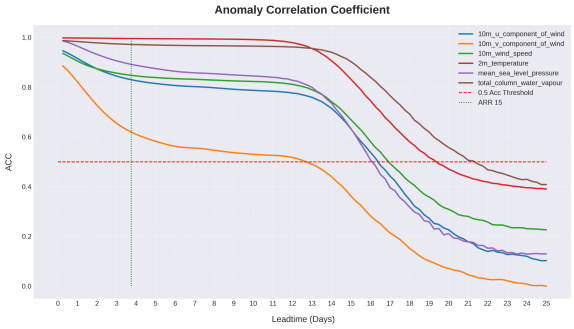
Comparing the 6 hour forecasts, we observe that the introduction of latitude weighting significantly improves the stability of the model predictions. Unlike the MSE-only model, whose forecasts quickly diverge, the latitude-weighted model maintains coherent large-scale structures over longer forecast horizons. In particular, the blow-up observed in the 66 hour forecasts with MSE is no longer present, indicating that the modified loss function helps stabilize the autoregressive rollout.

However, some degradation is still visible at longer lead times. In the t+99 forecasts, the predictions exhibit noticeable artefacts. These artefacts differ from those observed in the MSE-trained model. Instead of horizontal high-magnitude structures, the predictions show a repeating spatial pattern in which the overall magnitude of the field increases while the spatial structure remains largely unchanged.

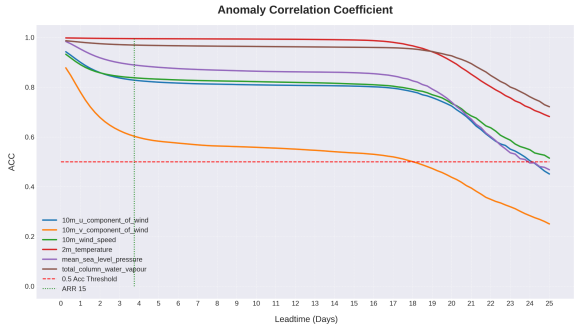
This suggests that while latitude weighting improves stability and prevents early divergence, the model still accumulates errors during long autoregressive rollouts. The persistence of spatial structure alongside increasing magnitude indicates that the model

captures the dominant modes of the atmospheric field but fails to correctly regulate their amplitude over extended forecast horizons.

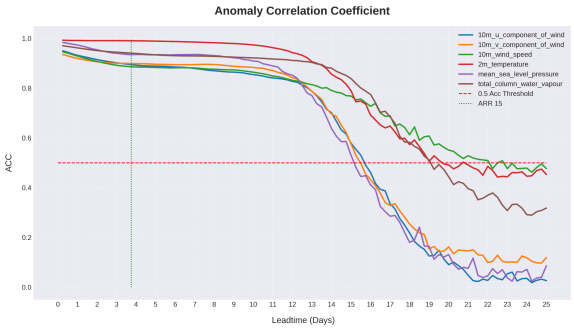
4.2.2 ACC



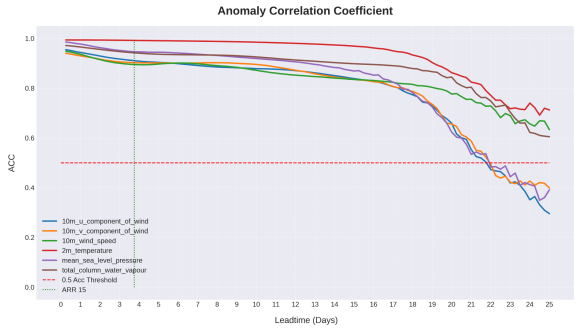
(a) Global ACC before training



(b) Global ACC after training



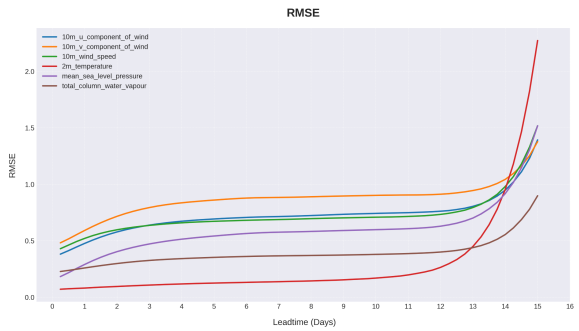
(c) India ACC before training



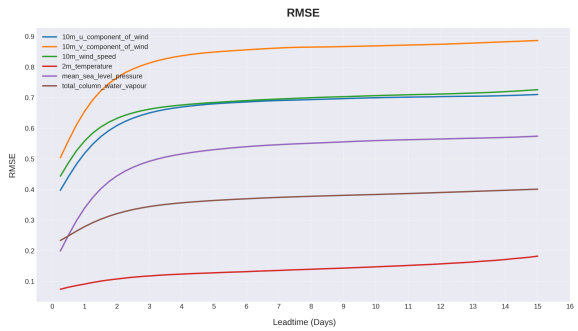
(d) India ACC after training

Figure 4.5: Anomaly Correlation Coefficient (ACC) comparison before and after training for global and India-masked evaluation at 5.6° resolution.

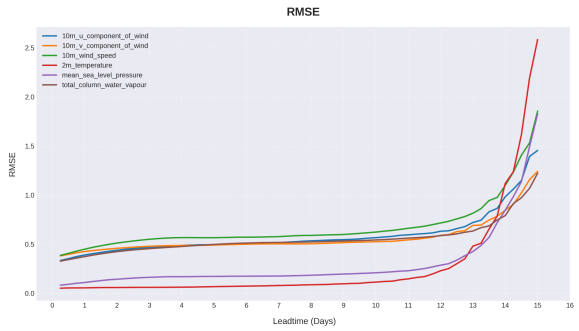
4.2.3 RMSE



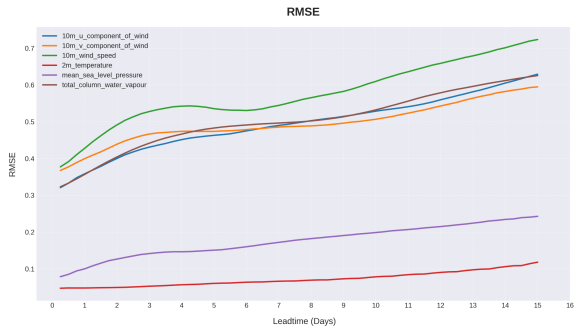
(a) Global RMSE before training



(b) Global RMSE after training



(c) India RMSE before training



(d) India RMSE after training

Figure 4.6: RMSE comparison before and after training for global and India-masked evaluation at 5.6° resolution.

Both the Anomaly Correlation Coefficient (ACC) and Root Mean Square Error (RMSE) show similar trends across these experiments. After fine-tuning the model on the Indian region, improvements are observed in both the global and regional ACC and RMSE values. This suggests that the initial training scheme has still not yet converged and the training duration should be increased.

It is important to note that the experiments conducted at 5.6° resolution were primarily intended as a proof-of-concept for the final high-resolution training at 1.4°. The hyperparameters used in the 5.6° experiments therefore served as an initial configuration for the higher-resolution training setup, allowing rapid experimentation and validation before scaling to the more computationally expensive 1.4° dataset.

4.3 High-Resolution Performance at 1.4°

The SFNO model is evaluated on the higher-resolution 1.4° dataset, which captures finer-scale atmospheric structures.

The 1.4° dataset contains a larger number of variables and significantly higher spatial resolution compared to the 5.6° dataset, resulting in a substantially larger model input space and increased number of effective parameters. This increase in complexity was the

primary motivation for initially conducting experiments on the lower-resolution dataset.

Using the same hyperparameters, learning rate schedules, and training scheme that were successful for the 5.6° experiments did not yield satisfactory results when directly applied to the 1.4° dataset. In particular, the resulting forecasts exhibited noticeably blurry predictions, indicating that the training configuration was not well suited for the higher-resolution setting.

Consequently, the training procedure was further refined for the high-resolution experiments. The final training configuration used for the 1.4° model is described in detail in the Methodology section.

4.3.1 Forecast Evolution Across Lead Times

Similar to the coarse-resolution experiments, forecast maps at 1.4° resolution are evaluated across multiple lead times in order to analyze the temporal evolution of prediction errors during autoregressive rollout.

The lead times considered in this analysis are 6, 36, and 84 hours. Across these lead times, all evaluated variables show a clear improvement compared to the coarse-resolution experiments. The rollout forecasts remain stable and retain coherent large-scale atmospheric structures even at longer forecast horizons.

Furthermore, the higher spatial resolution enables the model to capture finer-scale atmospheric features that were not well represented in the 5.6° experiments. This is particularly visible in the spatial structure of the predicted fields, where smaller-scale patterns and gradients are preserved more effectively in the forecasts.

MSLP

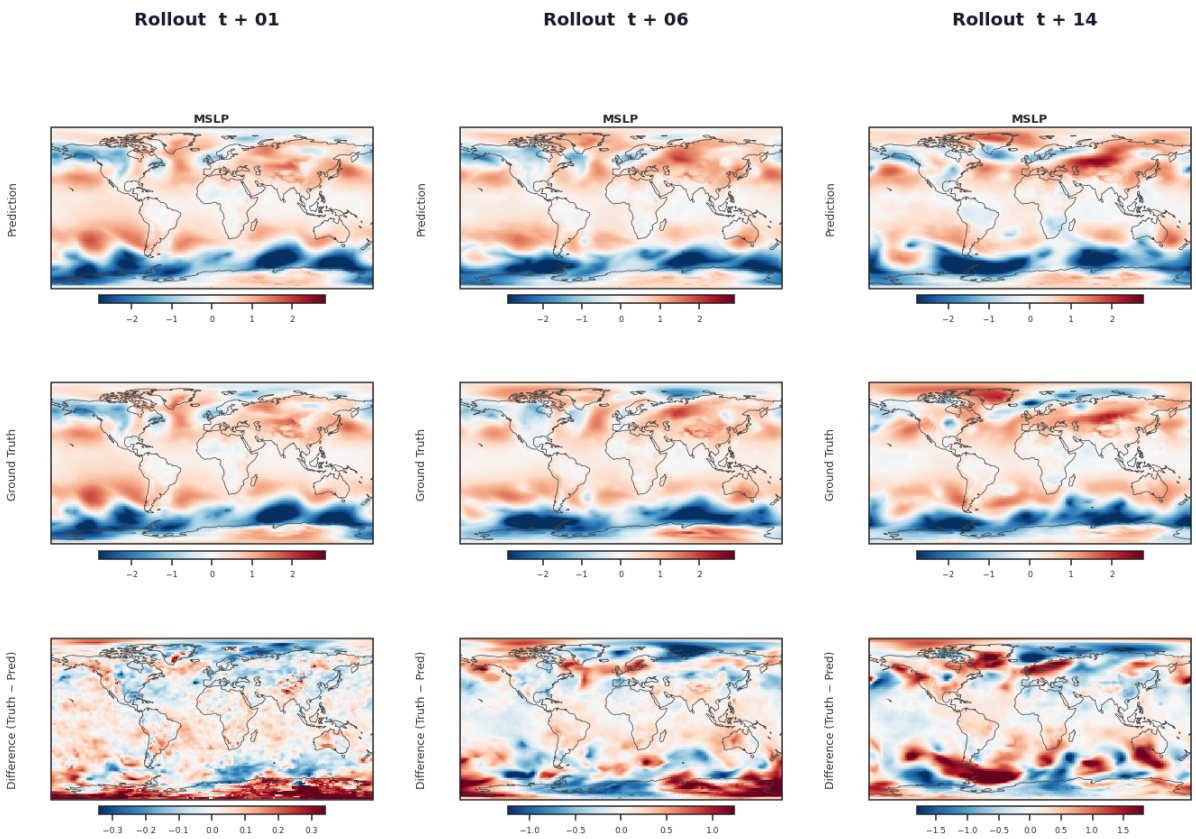


Figure 4.7: Autoregressive rollout forecasts for MSLP at 1.4° resolution.

T2M

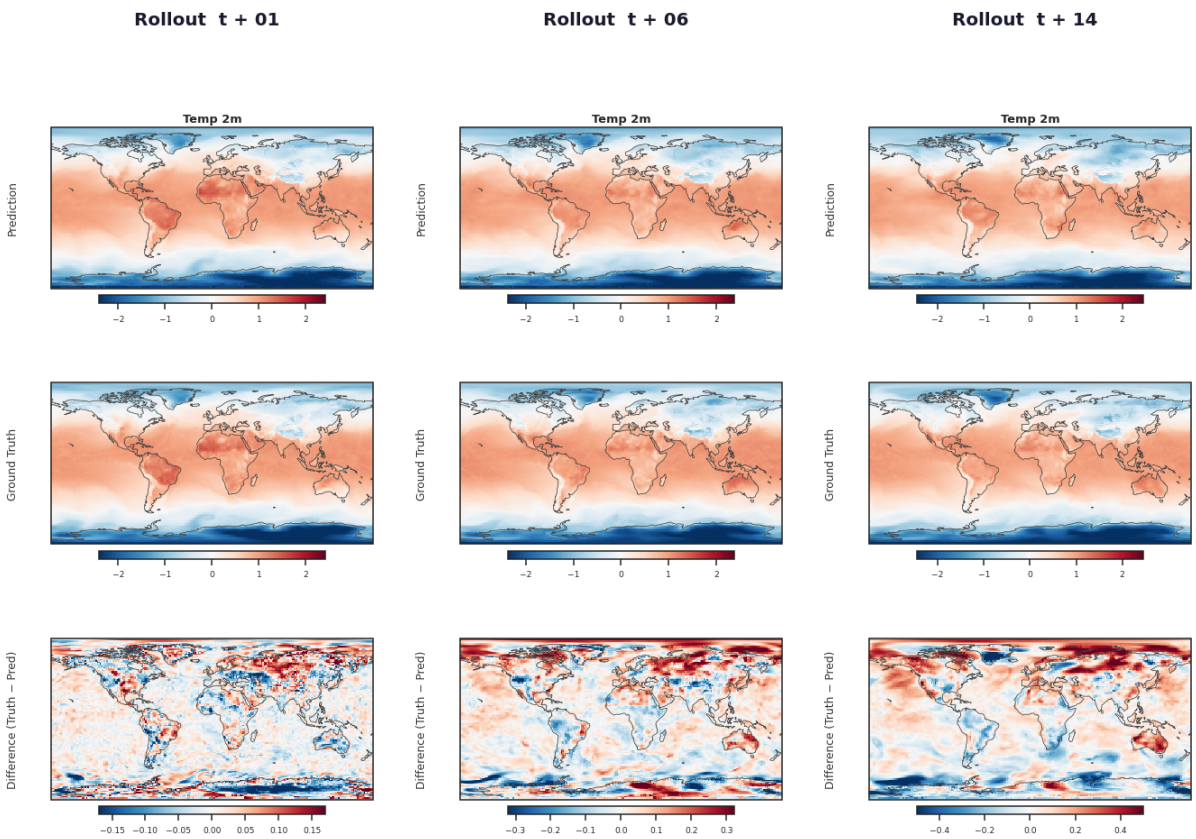


Figure 4.8: Autoregressive rollout forecasts for T2M at 1.4° resolution.

TCWV

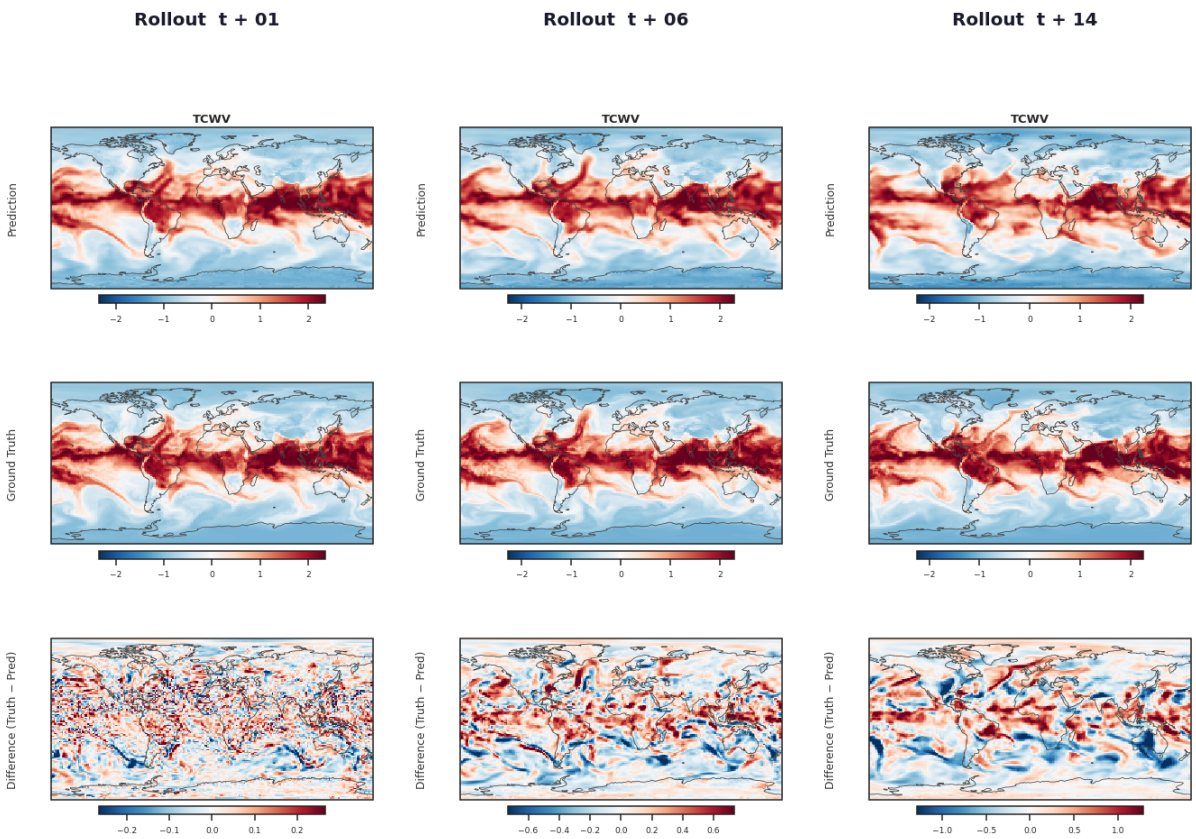


Figure 4.9: Autoregressive rollout forecasts for TCWV at 1.4° resolution.

U Wind 250

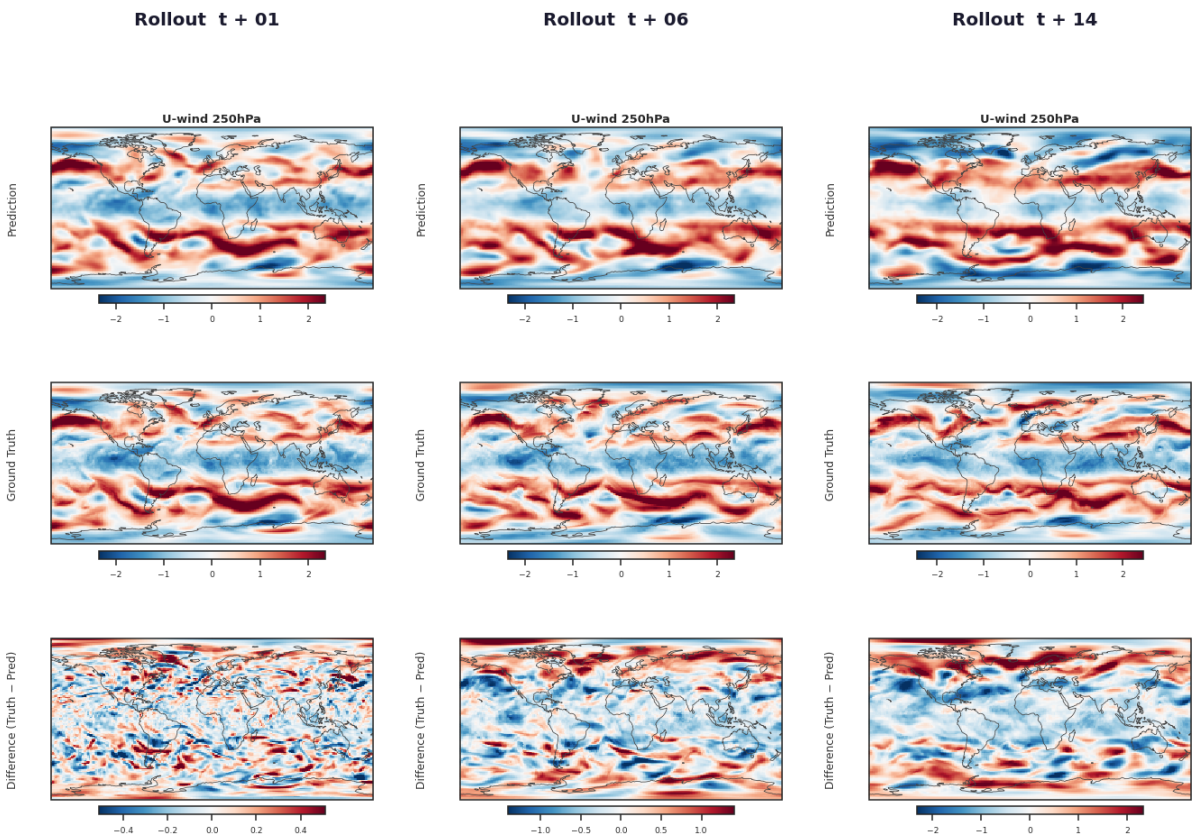


Figure 4.10: Autoregressive rollout forecasts for U-wind at 250 hPa at 1.4° resolution.

U10

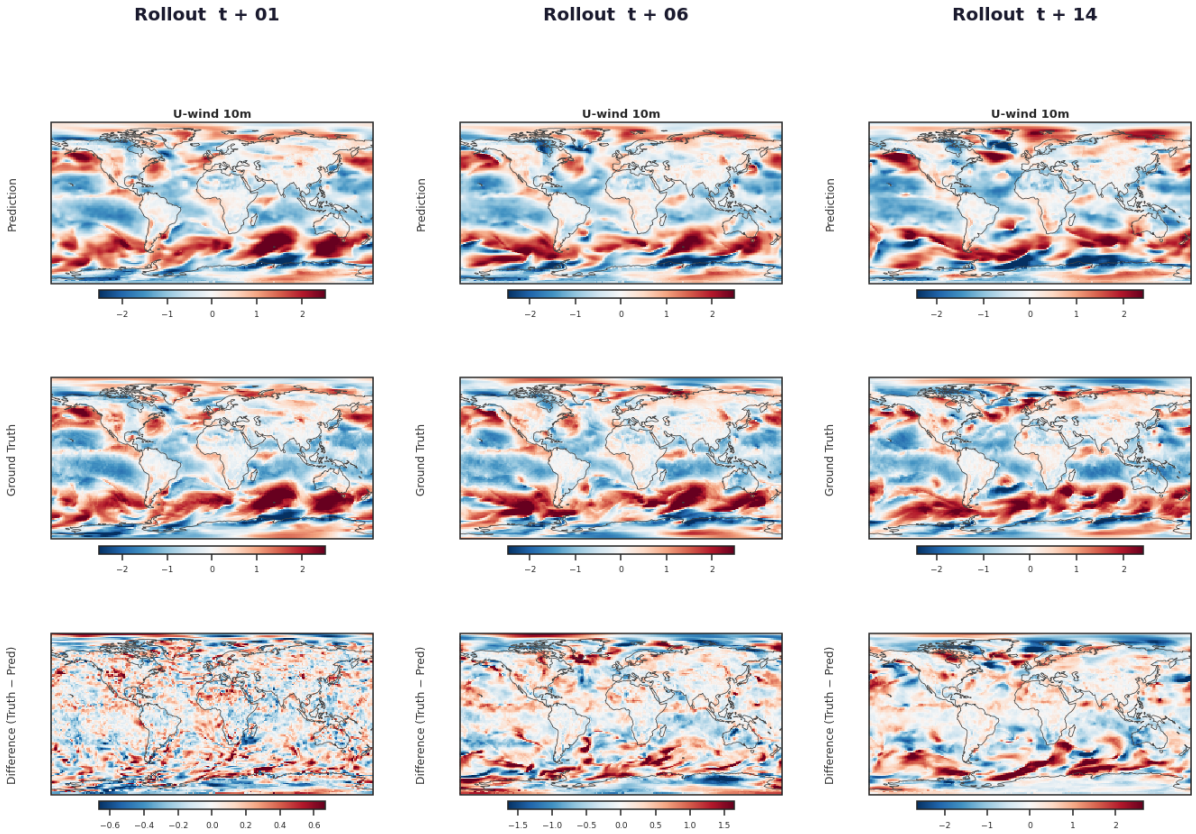
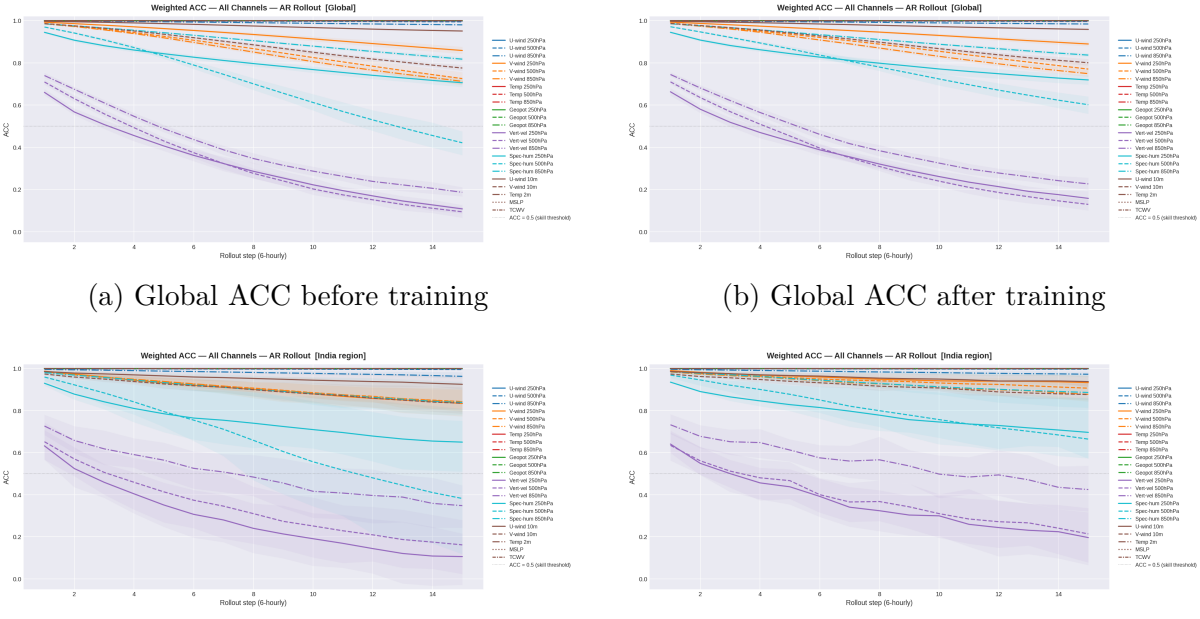


Figure 4.11: Autoregressive rollout forecasts for $U10$ wind at 1.4° resolution.

4.3.2 ACC



(a) Global ACC before training

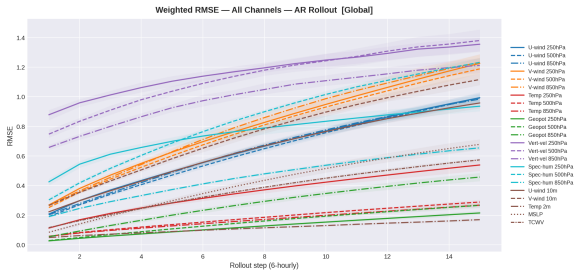
(b) Global ACC after training

(c) India ACC before training

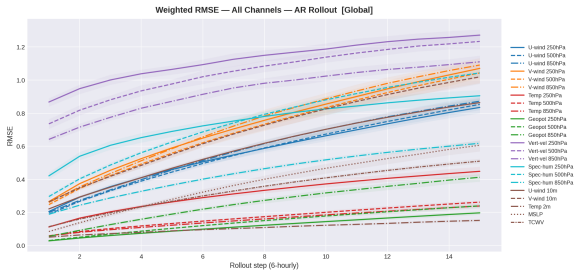
(d) India ACC after training

Figure 4.12: ACC

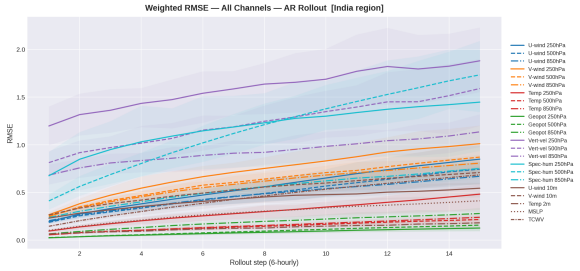
4.3.3 RMSE



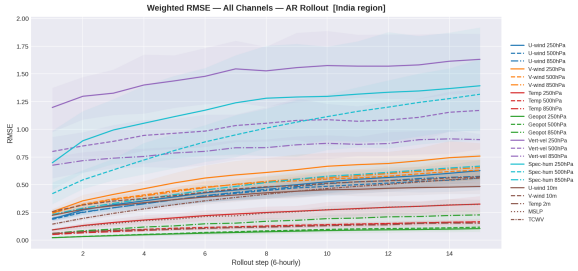
(a) Global RMSE before training



(b) Global RMSE after training



(c) India RMSE before training



(d) India RMSE after training

Figure 4.13: RMSE

4.3.4 Power Spectrum

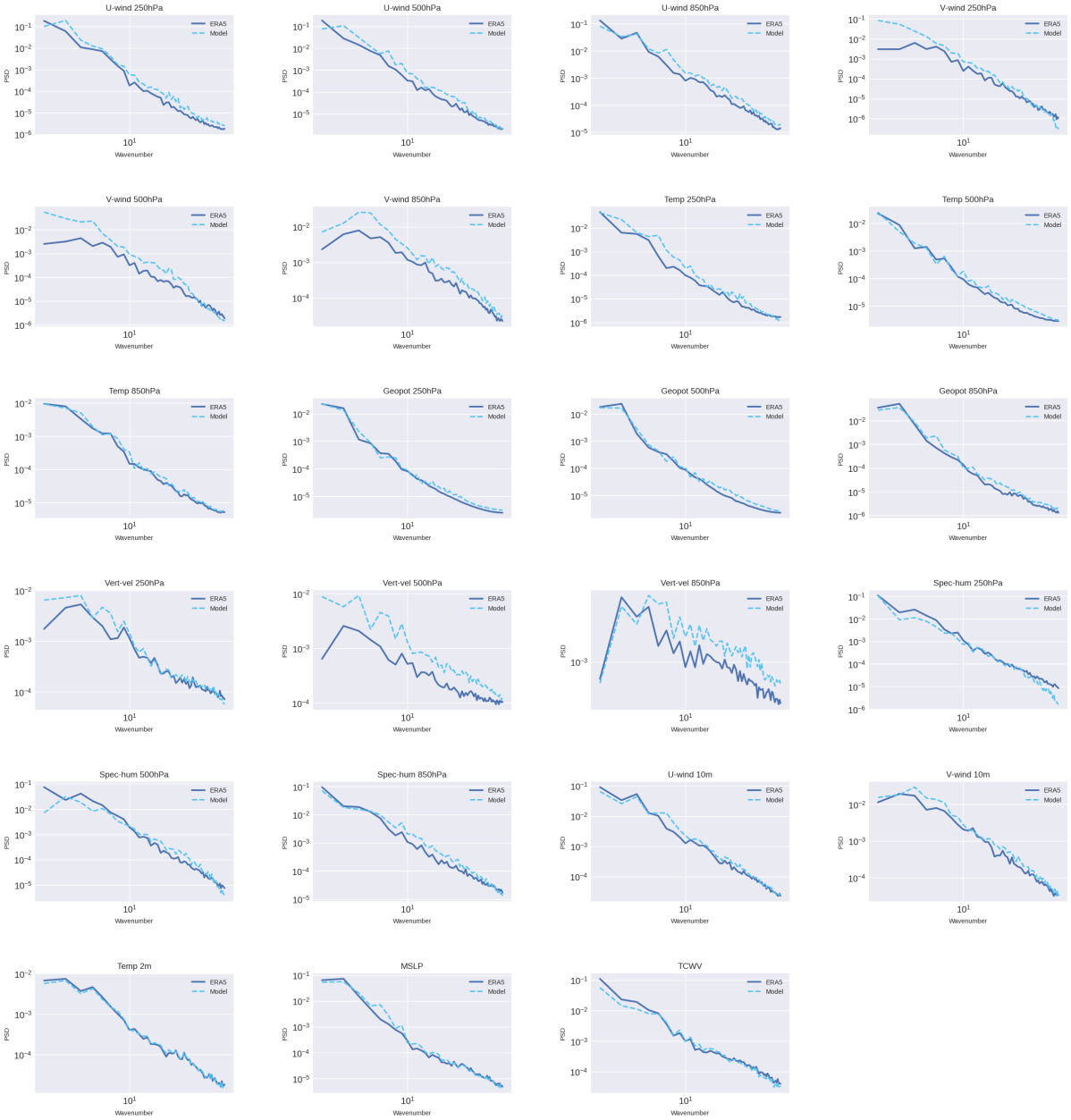


Figure 4.14: Power Spectrum

The evaluation metrics for the 1.4° experiments show a consistent picture across different measures of forecast quality. The Anomaly Correlation Coefficient (ACC), Root Mean Square Error (RMSE), and the spectral power analysis all indicate similar trends in model performance throughout the forecast rollout.

However, the model performance is not uniform across all atmospheric variables. Several variables demonstrate strong predictive capability over the entire rollout horizon. In particular, total column water vapor (TCWV), mean sea level pressure (MSLP), temperature, geopotential height, and the zonal and meridional wind components (u and v) across multiple pressure levels maintain relatively high ACC values and stable RMSE growth. These variables preserve coherent spatial structures and exhibit stable forecast behavior even at longer lead times.

In contrast, the model shows lower predictive accuracy for variables such as vertical velocity and specific humidity across pressure levels. These variables display comparatively lower ACC values and higher RMSE throughout the rollout. These variables have a finer structure present. The model performance suggests that variables that have a fine structure show low accuracy, this could be improved by creating a better spectral loss function.

Chapter 5

Conclusion

5.1 Summary of the Research Problem

A fundamental scientific and engineering challenge is accurate and efficient weather prediction. Traditional NWP systems solve physically based equation on a grid and require massive supercomputers, and have a long runtime. Recent development in deep learning based models and architectures have enables a new paradigm for weather prediction that learn the spatio-temporal dynamics of atmosphere directly through data.

Among these models neural operators have emerged as a promising method that learns mappings from function spaces rather than direct fixed dimensional input output pairs. Models such as Fourier Neural Operators (FNO) and its spherical variant have shown an ability to capture large scale as well as small scale atmospheric dynamics. However, reproducing benchmarks and model evaluations across resolution remain a challenge.

The primary objective of this thesis was to reproduce and systematically evaluate a SFNO based model for weather forecasting using ERA5 dataset. Particularly, this work aimed to study the performance of SFNO across different spatial resolutions, training regimes, analyze forecasting skill for multiple atmospheric variables, and investigate the stability and degradation of forecasts over extended rollout horizons.

5.2 Key Findings

The experiments demonstrate that neural operator–based models are capable of producing stable and physically plausible weather forecasts across multiple lead times. The results show that the forecasting performance varies significantly across different atmospheric variables. Large-scale dynamical fields such as geopotential height, temperature, mean sea level pressure, and horizontal wind components exhibit relatively strong predictive skill throughout the forecast rollout. These variables are characterized by smoother spatial structures and strong dynamical constraints, which may make them easier for the

model to learn.

In contrast, variables such as vertical velocity and specific humidity show lower predictive accuracy across most pressure levels. These variables typically exhibit higher spatial variability and are strongly influenced by small-scale physical processes such as convection and turbulence, which are more difficult for data-driven models to capture.

Across different spatial resolutions, the evaluation metrics and qualitative analyses consistently reveal similar patterns in model performance. Both ACC and RMSE curves indicate that the model maintains reasonable skill for several forecast steps before gradually degrading as the rollout horizon increases. Spectral analyses of the forecasts show that the model captures large-scale structures effectively but gradually loses accuracy at smaller spatial scales as forecast lead time increases.

Overall, the results suggest that SFNO-based models can learn meaningful representations of atmospheric dynamics and generate coherent global forecasts. However, the forecasting skill remains variable across atmospheric variables and scales.

5.3 Limitations

Despite the promising results, several limitations remain. First, the experiments focus on a limited set of atmospheric variables and resolutions, which may not fully capture the complexity of operational weather forecasting systems. Second, while neural operator models can learn large-scale atmospheric dynamics, their ability to represent small-scale physical processes remains limited.

Another limitation is the reliance on autoregressive rollouts for multi-step forecasting. Errors introduced in early prediction steps can accumulate over time, leading to gradual degradation of forecast accuracy. This phenomenon is common in data-driven forecasting models and remains an active area of research.

Finally, the experiments were conducted under specific training configurations and computational constraints. Exploring alternative architectures, larger datasets, and longer training regimes could further improve model performance.

5.4 Future Work

Several directions can extend the work presented in this thesis. One promising direction is the development of multi-scale neural operator architectures capable of simultaneously modeling large-scale atmospheric circulation and smaller-scale weather phenomena.

Another important direction involves uncertainty quantification. Providing probabilistic forecasts rather than deterministic predictions would allow neural operator models to better capture the inherent uncertainty in atmospheric systems.

Finally, improvements in data retrieval, training strategies, and representation learning may further enhance the forecasting skill of neural operator models. Continued progress in these areas could enable data-driven models to play an increasingly important role in the future of weather prediction.

Gen AI Usuage

I declare that Generative AI tools were used in a limited capacity to improve the clarity, structure, and flow of the text in this thesis. The use of such tools was restricted to language refinement and did not contribute to the core scientific ideas, presented in this work.

All technical content, analysis,and experiments, are my own, and any assistance from Generative AI was carefully reviewed and appropriately modified before inclusion.

Bibliography

Era5 hourly data on pressure levels from 1940 to present. *European Centre for Medium-Range Weather Forecasts*, Jan 2018. doi: https://doi.org/10.24381/cds.bd0915c6. URL `https://cds.climate.copernicus.eu/datasets/reanalysis-era5-pressure-levels?tab=overview`.

2025. URL `https://github.com/NVIDIA/makani/tree/main/makani`.

Kwangjun Ahn, Zhiyu Zhang, Yunbum Kook, and Yan Dai. Understanding adam optimizer via online learning of updates: Adam is ftrl in disguise. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL `https://arxiv.org/abs/1607.06450`.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere, 2023.

S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi: 10.5334/jors.148. URL `https://doi.org/10.5334/jors.148`.

Dmitrii Kochkov, Janni Yuval, Ian Langmore, Peter Norgaard, Jamie Smith, Griffin Mooers, Milan Klöwer, James Lottes, Stephan Rasp, Peter Düben, et al. Neural general circulation models for weather and climate. *Nature*, 632(8027):1060–1066, 2024.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, Dec 2023. doi: https://doi.org/10.1126/science.adi2336. URL `https://www.science.org/doi/10.1126/science.adi2336`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.