Generative Machine Learning in Atmospheric Models

by Varshini Thavakumar

Department of Mathematical Sciences, University of Bath

Supervisors:

Lisa Kreusser, Teo Deveney

Industrial Supervisor:

Helena Reid

September, 2024

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. in Mathematics with Data Science for Industry.

Abstract

This project investigates the use of generative machine learning in atmospheric modelling to improve the representation of uncertainties associated with sub-grid scale processes in its vertical structure. Atmospheric models, essential for accurate weather predictions, often struggle to detail the complex physical processes that occur at scales smaller than the model grid resolution. Traditional parameterisation methods assume a uniform distribution in its vertical structure that may not accurately reflect the stochastic nature of these processes across different vertical levels. In this research, denoising diffusion probabilistic models (DDPMs) are applied to learn atmospheric profiles in its vertical structure . By training with historical data from the Met Office, the models learn to generate plausible outcomes for sub-grid processes based on observed conditions in the larger-scale data. The results indicate that diffusion models perform well in learning the probabilistic nature of sub-grid processes. However, the substantial computational requirements for training and deploying these models pose challenges, particularly for operational uses where rapid processing is essential. This work could potentially contribute to the integration of generative machine learning into existing paramaterisation schemes. Future research will aim to optimise the computational efficiency of these models and assess their practical application.

Acknowledgements

I want to thank my supervisors for their invaluable support throughout this project. Thanks to Helena for her guidance on the physics and data, and to Teo and Lisa for their help with the modelling and mathematics. This project was challenging, but the support from my supervisors made it a rewarding experience. I'm grateful for the opportunity to contribute to this field.

Contents

1. Introduction

1.1 Problem Background

The Met Office use atmospheric models that discretise the atmosphere into a three dimensional grid to simulate and predict weather patterns. These models must account for physical processes such as convection, radiation, and precipitation, which typically occur on scales much smaller than the grid itself. To address this, the Met Office use parameterisation techniques to estimate the effects of these sub-grid scale processes based on the available grid scale data. This is a complex task, as much of atmospheric model development focuses on refining these parameterisation techniques. The Met Office's models rely on non-hydrostatic compressible Euler equations which are solved by the Met Office's dynamical cores like ENDGame [1], used in the Unified Model (the current operational atmospheric model) and the newer GungHo [2], which is under active development for next-generation modelling systems. These equations describe horizontal motion, hydrostatic equilibrium, continuity, thermodynamics and many more physical processes to generate accurate predictions. These models are structured as three-dimensional arrays, representing variables such as wind, temperature, pressure, water vapour, and cloud cover across various latitudes, longitudes, and altitudes. By numerically solving these equations, the models predict how these variables evolve over time, with each new atmospheric state serving as the initial condition for the next time step, which can be calculated at varying intervals depending on the purpose. Figure 1.1 is a simple diagram showing some of the sub-grid scale physical processes that are parameterised.

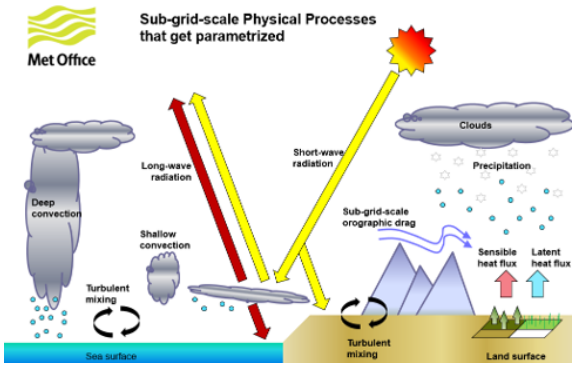

Figure 1.1: Diagram showing the sub-grid-scale physical processes parameterised in atmospheric modelling. Image credit: The Met Office

To achieve accurate predictions, very high-resolution grids are necessary, ideally as fine as possible to capture detailed atmospheric processes. However, this resolution is constrained by computational limitations. As the atmospheric model is three dimensional, halving the grid length results in an eightfold increase in the number of data points, which in turn requires 8 times more memory and computational resources. Additionally, the time step must be reduced with the grid size, as with a smaller grid, air moving too fast relative to the timestep could cross multiple grid boxes in one step, causing unrealistic behaviour in the model. Thus by simply having the grid length, simulations are made up to 16 times more computationally expensive.

The Met Office use a few schemes that model unresolved sub-grid processes using stochastic physics. Stochastic physics incorporates randomness or uncertainty into the modelling of physical systems. These models acknowledge that unresolved processes, which are too small to be directly captured by the grid, introduce uncertainty. This randomness stems from our incomplete understanding of the sub-grid scale processes and how they interact with the larger system. A key challenge in using stochastic approaches is deciding how to represent the random component, where in this project we find a way to represent the randomness in terms of its vertical distribution. For example, we need to determine whether the randomness is constant with height, independent at each vertical level, or if the moments of the distribution vary over time. Additionally, we must decide how to represent correlations between different vertical levels. These are difficult choices that require significant manual tuning. Generative machine learning offers an alternative approach. It is data-driven and can learn from extensive historical atmospheric data, enabling them to capture complex patterns that might be less explicitly modelled by stochastic approaches. Importantly, they can bypass the need to manually specify the vertical distribution of random components. Instead, a generative model encodes this distribution within the learned weights of the model, based on the training data. This ability to learn and represent sub-grid scale processes in detail could potentially enhance the overall accuracy and detail of atmospheric simulations.

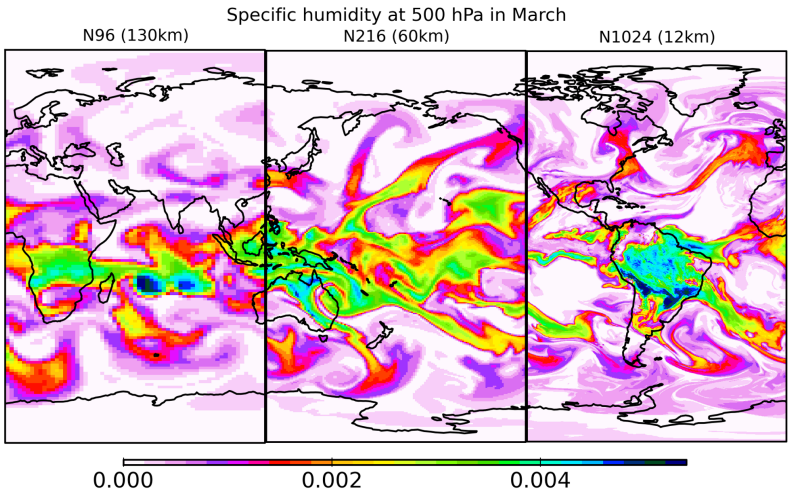

Figure 1.2: Atmospheric models with increasing resolution from left to right. Image credit: The Met Office

1.2 Proposed Solution

The aim of this project is to use generative machine learning, particularly diffusion models, to improve the parametrisation of uncertainty in unresolved sub-grid scale processes. Diffusion models, a class of generative models, have shown recent significant potential in producing high-quality outputs that often surpass GANs in terms of realism and diversity. By applying these models to atmospheric data, this project seeks to develop a more realistic method for representing the uncertainty in unresolved sub-grid scale processes, making the representation more natural and based on local conditions rather than relying on simple assumptions. The proposed solution involves training a diffusion model on historical atmospheric data, enabling it to learn the underlying distributions and relationships between different atmospheric variables at various scales. Once trained, the model can be used to generate more accurate samples of sub-grid scale processes based on specific atmospheric conditions, providing a more detailed representation of uncertainty.

This research could potentially serve as the preliminary basis for integrating generative machine learning models into the Met Office's existing stochastic physics scheme. From this research, the performance of this approach would be validated by comparing model outputs against both observational data and traditional parameterisation methods, testing across various atmospheric conditions and scales. This could pave the way for more accurate and efficient weather forecasting and climate modelling, leading to many more further advancements in atmospheric sciences.

1.3 Literature Review

This section analyses various papers on parameterisation schemes that involve stochastic physics, machine learning or a combination of both.

1.3.1 Papers on Stochastic Physics in Parameterisations

The paper titled "*Impact of a Stochastic Kinetic Energy Backscatter Scheme Across Time-Scales and Resolutions*" by Sanchez et al.(2014) [3] provides a detailed analysis of the Stochastic Kinetic Energy Backscatter (SKEB2) scheme used in the Met Office Unified Model. The SKEB2 scheme aims to address the loss of kinetic energy due to modelling by introducing stochastic perturbations that reintroduce some of this lost energy. This is achieved through evolving a spectral gaussian noise field through time with a simple autocorrelation. The study explores the effectiveness of SKEB2 across various time scales, from short-term weather forecasts to long-term climate simulations. It shows that while SKEB2 can reduce the accuracy of individual short-term weather forecasts in some metrics like RMSE and ACC, it improves the model's mean bias, particularly in extratropical regions where energy loss is a significant issue. However, the study also points out that SKEB2 can sometimes lead to an excessive increase in wind strength in tropical regions, indicating the need for further tuning depending on the model's resolution and the specific atmospheric conditions.

A similar paper by Palmer et al.(2009) [4], provides a detailed examination of stochastic parametrisation schemes, specifically SKEB and Stochastically Perturbed Parametrisation Tendencies (SPPT) schemes. SPPTs are useful in forecasting using ensemble data

assimilation, which involves using multiple model forecasts to estimate the state of the atmosphere, accounting for uncertainties in both the observations and the model itself. The SPPT scheme contributes to this processs by introducing perturbations to the physical tendencies by multiplying them by random numbers, which helps to simulate the effects of model uncertainty more realistically. The paper concludes by discussing that although these schemes have shown promising results, there is still underdispersion in temperature predictions, suggesting further stochastic elements might need to be incorporated into other components of the forecast model.

A paper published by Ollinaho et al.(2017) [5] builds upon the foundation established in previous research, which introduced and refined stochastic parametrisation schemes to address model uncertainties. This paper introduces the Stochastically Perturbed Parametisations (SPP) scheme, which directly perturbs 20 key parameters and variables within different model components. These perturbed quantities are chosen from parametrisations of turbulent diffusion, subgrid orography, convection, clouds and large-scale precipitation and radiation. Compared to the existing SPPT scheme, SPP demonstrate similar performance for short-term forecasts (up to 24 hours) and offers improved accuracy for surface temperature predictions and precipitation representation. The article concludes by suggesting to combine both SPP and SPPT schemes which could provide a more comprehensive model to represent uncertainty.

The relevance of Sanchez et al.'s paper to this project lies in their use of structured randomness in stochastic schemes, particularly in the horizontal and temporal dimensions. My project extends this concept by focusing on generating randomness with vertical structure, with the hope that having a naturalistic vertical structure as well could improve similar schemes. While the SPPT scheme in Palmer et al.'s work applies randomness to the outputs of other schemes to account for uncertainty, my approach aims to apply machine learning-generated randomness to the inputs of these schemes.

1.3.2 Papers on Machine Learning Applications in Parameterisations

In the paper "*Could Machine Learning Break the Convection Parameterization Deadlock?*" by Gentine et al. (2018) [6], the authors explore the potential of machine learning, specifically artificial neural networks (ANNs), to improve the representation of unresolved moist convection in climate models. Traditional parameterisation of convection in coarse-scale models has been a significant issue, leading to biases in climate simulations. The paper proposes using a machine learning-based approach, termed "Cloud Brain" (CBRAIN), trained on data from a superparameterised climate model, which explicitly resolves convection at high resolution. CBRAIN demonstrates an ability to predict key features of convective processes, such as heating, moistening, and radiative effects, with a high degree of accuracy while being much more computationally efficient than traditional superparameterisation methods. However, the deterministic nature of ANNs means that the approach lacks the inherent stochastic variability of the original superparameterised model. Despite this, the authors argue that machine learning offers a promising alternative to current parameterisation methods, potentially improving the accuracy of climate models while reducing computational costs.

Yuvan and O'Gorman's paper in 2020 [7] also explores the potential of machine learning for enhancing subgrid process representations in climate modelling. While Gentine et al. demonstrated the use of neural networks in capturing convective processes, Yuval

and O'Gorman use Random Forests (RFs) to parameterise subgrid processes like vertical advection, cloud microphysics, and radiative heating. The paper notes some of the issues faced by using an NN parametrisation, with many leading to instability and suffering from climate drift. Yuval and O'Gorman's study successfully implemented RF parameterisation in coarse-resolution climate models, showing that it can replicate the climate characteristics of high-resolution models with notable accuracy, including mean and extreme precipitation. The robustness of the RF parameterisation is tested by excluding certain latitude bands during training, revealing that the model can generalise well across different latitudes. The authors suggest that this ML-based approach can be a valuable tool for climate modelling, as it offers stability and accuracy across a range of resolutions. They conclude by outlining several areas for future research in the development of machine learning parametrisations for atmospheric models. These include how to deal with training over regions with topography, and how to deal with separate parametrisation of radiative heating in the stratosphere. Future research should continue to seek insights into how performance varies across different length scales and whether parametrisations should account for nonlocal effects in time and space.

1.3.3 Papers that combine Stochasticity and Machine Learning Applications in Parameterisations

In the context of this project, we are particularly interested in the potential of generative machine learning models to improve stochastic physics in atmospheric models, specifically for representing the uncertainty of unresolved sub-grid scale processes. Therefore, research that has already explored the integration of machine learning with stochastic approaches is highly relevant, as it demonstrates how data-driven models can enhance the accuracy and realism of parameterisations in climate and weather prediction models.

In the 2013 paper by Krasnopolsky et al.[8], the authors explore an innovative approach to replace traditional convection parameterisation schemes in climate and numerical weather prediction models by using a stochastic machine learning method. Specifically, they develop an ensemble of neural networks trained on high-resolution data from a cloud-resolving model (CRM). This ensemble approach aims to capture the inherent uncertainties and stochastic nature of subgrid-scale processes that are typically not resolved in global climate models. The key contribution of this paper lies in the use of an ensemble of NNs to emulate the the behaviour of fine-scale CRM simulations at larger GCM scales in a variety of conditions. This approach led to more realistic and computationally efficient representations of convection. This early example of combining machine learning with stochastic processes set the stage for future research in this area, highlighting the potential of ML-based approaches to address some of the longstanding challenges in atmospheric modelling.

Gagne et al.(2020) [9] explores the use of Generative Adversarial Networks (GANs) to develop a stochastic parameterisation framework for subgrid processes within atmospheric models. This study focuses on the Lorenz '96 model, a simplified 1D system often used to study chaos in atmospheric dynamics. The authors generated high-resolution "truth" data by running the full Lorenz '96 equations and then used truncated versions of these equations, replacing the omitted terms with GAN-based parameterisations. The GANs were trained to produce realistic subgrid-scale tendencies conditioned on the large-scale state of the model, thus providing a stochastic rather than deterministic approach to parameterisation. The research demonstrates that certain configurations of GANs not only

closely approximate the subgrid tendencies but also outperform traditional parameterisations in both weather and climate simulations. The stochasticity inherent in the GANs allows for more accurate representation of uncertainty and variability, which is crucial for realistic modelling of chaotic systems. This work is significant as it moves beyond deterministic machine learning parameterisations, which have dominated recent research, by showing the benefits of incorporating stochastic elements into probabilistic ML models.

Guillaumin and Zanna (2021) [10] introduce deep learning with stochastic methods in ocean modelling to improve the parametrisation of subgrid momentum forcing. In their study, they use a deep learning model trained on high-resolution data from the CM2.6 climate model, a high resolution coupled climate model. The key innovation in their approach is the use of a convolutional neural network (CNN) to predict both the mean and standard deviation of a Gaussian probability distribution for subgrid momentum forcing, rather than generating deterministic outputs. This method allows the generation of a stochastic parameterisation by sampling from the predicted distributions, capturing the uncertainty in sub-grid scale processes. The CNN is trained using a probabilistic loss function that minimises the negative log-likelihood. Then, the trained model is implemented in a lower-resolution ocean model to assess its impact. The results demonstrate that the stochastic parameterisation improves the representation of kinetic energy and sea surface height variability, bringing the low-resolution model's output closer to that of the high-resolution reference simulation. The study also shows the model's ability to generalise well across different regions and climates, including a scenario with increased CO2 levels, without requiring further tuning. This research highlights the broader applicability of ML techniques across both ocean and atmospheric modelling, where the representation of sub-grid processes remains a significant challenge.

1.4 Motivation

While these papers provide valuable contributions to advancing parameterisation schemes using stochastic physics and machine learning, one notable limitation is the treatment of uncertainty, particularly in the vertical distribution of sub-grid scale processes. Many existing methods, including those discussed in the literature, tend to assume a uniform distribution of uncertainty across vertical levels, which may not fully capture the variable nature of these processes. By learning the vertical variation of sub-grid scale processes, we aim to improve the representation of this uncertainty. The papers discussed in Sections 1.3.2 and 1.3.3 demonstrate the potential of machine learning applications in parametrisations, proving the ability to enhance stability and accuracy in results. While these methods show the benefits of learning from the data, most of the machine learning approaches explored in these studies rely on deterministic or discriminative models, which learn a direct mapping from inputs to outputs. While these methods have shown promise, they often fall short in capturing the stochastic nature of atmospheric processes, particularly at finer scales. Recent research has shown a rise in generative machine learning, arguing that discriminative models often lack semantic understanding of its training data as they are unable to account for the inherent uncertainty that exists in the real world [24]. Gagne et al. (2020) demonstrated the potential of probabilistic machine learning, specifically GANs, for stochastic parameterisation by capturing subgrid variability and improving weather and climate simulations in the Lorenz '96 model. Despite the

promising results, Gagne et al. (2020) noted several limitations in their use of GANs for stochastic parameterisation. One major challenge was the instability in training GANs, which often struggled to reach a stable equilibrium between the generator and discriminator networks. This can lead to issues such as mode collapse, where the model fails to capture the full range of variability in the data [11]. Additionally, GANs were sensitive to hyperparameter settings, particularly the magnitude of the noise injected into the model, and required extensive tuning to achieve optimal performance.

Recent research has shown that diffusion models are emerging as a better alternative for generative tasks, particularly in capturing finer details and reducing training instability, which GANs are known to suffer from [12]. Diffusion models operate by iteratively transforming simple noise distributions into data distributions, offering a more stable training process and better performance in capturing the probabilistic nature of its training data. Thus, in this project, we will explore the application of diffusion models to represent vertical variability in sub-grid scale processes. The following chapters build on this motivation, starting with Chapter 2, which explores the Met Office Unified Model outputs and describes the preprocessing steps necessary for handling the coarse and fine grid data. This section also discusses the atmospheric variables that will be trained on, including the dataset dimensions needed for model selection. Chapter 3 introduces the core mathematical concepts and algorithms underpinning the machine learning models used in this project. It is split into two sections, starting with an introduction to deep neural networks, focusing on the architectures of feed-forward neural networks, convolutional neural networks, and U-Nets. The second half covers generative modelling, including various types of deep generative models, with a detailed explanation of diffusion models and their suitability for our problem background. Chapter 4 discusses the Denoising Diffusion Probabilistic Model (DDPM), the chosen architecture for training our dataset, tying in the mathematical concepts from Chapter 3. The most suitable model for our project was researched and implemented to adapt to our dataset. It also outlines the training steps, computational requirements, and the development from a simple unconditional model to a conditional model. Chapter 5 evaluates the model's performance, presenting the results of both the unconditional and conditional models and their effectiveness in capturing vertical variability in sub-grid processes. Finally, Chapter 6 summarises the key findings, addresses the limitations faced, such as data constraints and time limitations, and suggests directions for future work. All of the code for this project is available on my GitHub repository, which can be accessed here.

2. Data

2.1 Data Source

The dataset was generated using the Met Office's Unified Model (UM), from a nesting suite [32] . The global model operated under the Global Atmosphere 6 configuration [33], with a resolution of N1280 which corresponds to 2N points of longitude and 1.5 points of latitude. This setup results in grid boxes of approximately 15km at the equator, with smaller grid boxes at higher latitudes. The model ran with a 6-minute timestep. The global model was used to update the lateral boundary conditions of a higher-resolution regional model, running over Ireland and the British Isles, every hour. This limited area model (LAM) utilised the Regional Atmosphere and Land configuration (RAL2-T), [34], operating with a uniform horizontal grid length of 1.5 km and a 1-minute timestep.

The LAM domain consisted of 512×512 grid points, though points within 16 grid spaces of the boundary were excluded from the dataset due to spatial spin-up effects. This is due to the boundary only containing coarser data which heavily influences the model output, thus the domain is reduced to 480×480 grid points. As it takes time and distance for the fine scale details to spin-up, the first day of the simulation, which took place on the final day of December, was also discarded. Both models use a stretched vertical grid with 70 levels, 50 of which were below 18 km, and a model top at 80 km. Regional atmospheric configurations typically compress the vertical dimension to a 40km top for weather forecasting purposes, as opposed to global climate modelling timescales. In weather-forecasting timescales, it is less important to assess high in the stratosphere and more significant to resolve the boundary layer and the lower troposphere, nearer the surface, where the weather is happening. This was not applied in this instance to maintain consistent vertical resolution between the training data and global models, because ultimately, if machine learning models trained on this data, such as this one, were to be used, they would be integrated into our global model.This approach aimed to facilitate the direct use of machine learning models without the need for vertical interpolation, which could introduce complications. The atmospheric component of the model was the Unified Model (UM), while the land component was JULES, both of which were coupled together.

The simulations were conducted for the duration of January 2020. The global model was reinitialised every 24 hours using operational analyses to correct any drift so that the global model was regularly bumped back to the actual atmosphere on that day. No ocean model was coupled to the land and atmosphere components, instead, sea surface temperatures were supplied in advance by the OSTIA system [35]. The higher resolution model ran continuously without being reinitialised throughout the whole month.

Although 80 LAMs were distributed globally, the one-way communication between the global model and the LAMs (the global model informs the LAMs about the edges but the LAMs do not inform the global model anything) means this does not affect the current dataset. However, it is worth noting for future work that the framework could be easily extended to accommodate additional LAMs.

2.2 Data Processing

The data was provided via a public FTP server which could be accessed by credentials and instructions given by my industrial supervisor. The data was downloaded using Windows commands in Powershell, which took approximately 30 minutes. The data consists of directories, "input" and "target", with each containing "fixed" and 'temporal' directories. In each of these directories, there are 360 numPy files of the form '$r77\_YYYYMMDD\_HH.npy$', where each file indicates the components modelled at each hour in a two week period of January 2020. The "r77" region refers to an area covering Ireland and part of the UK. In the original dataset, there are 80 regions worldwide which totals to $5TB$ of data, but we will be using only this one region as this amount of data is too cumbersome. As mentioned previously, the domain consists of 480×480 grid points which corresponds to approximately $720km \times 720km$ (one 7-degree by 7-degree region), divided into 144 grid boxes on a coarse 12×12 grid, achieved through horizontal averaging. Thus the low resolution data is given at a 60km scale whilst the high resolution data is at a 1.5km scale, as mentioned in the Source section previously. A random sample of 100 high resolution samples (out of 1600) is taken from each larger grid box. This was to reduce the size of data from more than 100GB to around 9GB.

The "input" directory contains the data from the coarse grid, while the "target" directory contains data from the original fine grid. The "fixed" directory contains variables which are fixed in time, however the 100 randomly selected fine grid-points per coarse grid box is not fixed for all times. These consist of orography, which refers to the nature of a region with respect to its elevated terrain [36] and the land-sea fraction. Thus the fixed input data has the shape (144,2), with the first dimension being an index of which coarse grid box it is, and the second referring to land sea mask and orography. The two horizontal dimensions are collapsed to one because the horizontal distribution of the data is not significant. The fixed target data has the shape (144, 100, 2), where the 100 refers to the randomly selected fine grid points. The "temporal" directory contains variables expected to change over time. These variables are specific humidity, pressure and temperature. The temporal data has an additional dimension of length 70, which is the vertical axis, thus the temporal input and temporal target data have shapes (144, 70, 3) and (144, 70, 100, 3) respectively. This vertical axis is irregularly spaced, with approximately 50 levels intended to lie in the troposphere, corresponding to the first 10-20km depending on latitude and conditions, and around 20 levels in the stratosphere. However, since the height of the troposphere varies and the 50th level is fixed, the actual number of levels in the troposphere and stratosphere can fluctuate. Additionally, the lower 50 levels are terrain-following, meaning they become compressed when the surface altitude is higher. As mentioned before, assessing the lower boundary and troposphere is more significant, which is why there are more levels at lower altitudes. The data has already been crudely normalised to get it all onto roughly the same scale. Orography has been divided by 4000m, air pressure divided by 106000Pa, specific humidity divided by 0.025kg/kg, and temperature has been rescaled by subtracting 140K and dividing by (320-140)K.

2.3 Exploratory Data Analysis

To understand the dataset's characteristics, we performed exploratory data analysis on the samples taken at a specific time, first focusing on the variables on the coarse grid scale. The fixed variables analysed include the land-sea fraction, orography, and the standard deviation of orography. This is shown in Figure 2.1, which shows the geographic and topographic variations within the selected region "r77", which covers Ireland and part of the United Kingdom at midnight on the 1st of January 2020. A subtle outline of the UK can be observed in all three of the plots, with the land-sea fraction plot showing it most distinctly, as expected. The first plot in Figure 2.1 displays the land-sea fraction, which

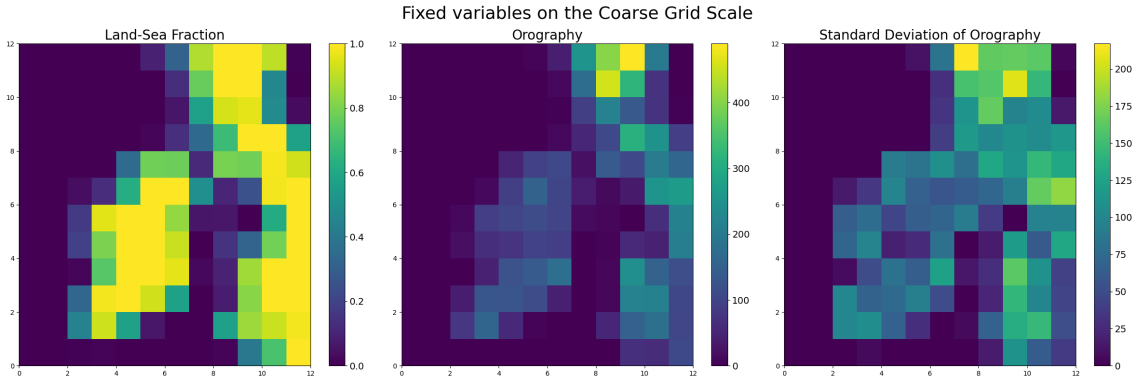

Figure 2.1: Fixed variables shown on the coarse grid scale

represents the proportion of land versus sea within each grid cell on the coarse 12×12 grid. The values range from 0 to 1, where 0 represents a complete sea coverage and 1 represents complete land coverage. The second plot illustrates the orography. The colour scale shows variations in elevation, with lower values representing areas closer to sea level and higher values indicating elevated terrains. The third plot shows the standard deviation of orography, which was calculated using the finer sub-grid samples within each coarse grid cell. This measure provides insight into the variability of orography, where higher standard deviation values suggest more varied terrain, while lower values suggest more uniform terrain. Initial analysis of these variables form the basis for further investigation into how these fixed variables influence the temporal atmospheric variables. Figure 2.7 shows the temporal variables at the surface (Level 0). The first plot

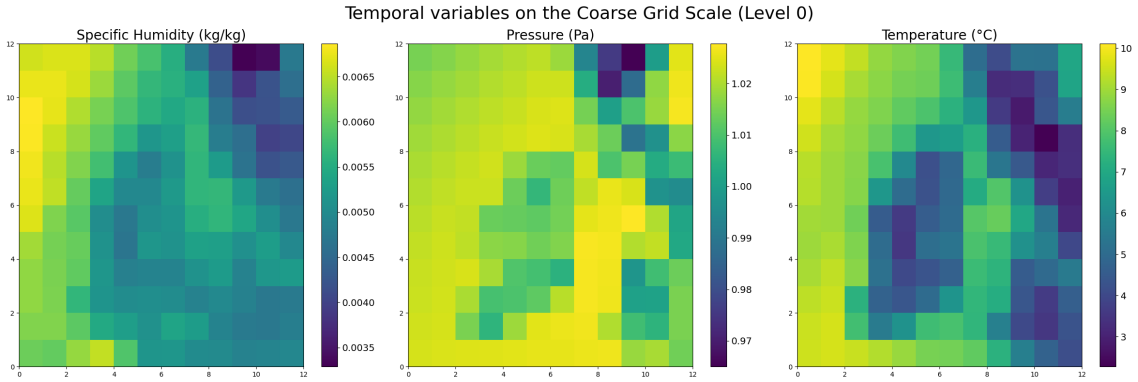

Figure 2.2: Temporal variables shown on the coarse grid scale at the surface

shows the distribution of specific humidity at the surface across the coarse grid scale. Specific humidity measures the mass of water vapour per unit mass of moist air expressed in kg/kg. Coastal and sea regions are more humid compared to the inland areas, which is consistent with expected geographical patterns of humidity distribution. The second plot depicts the surface pressure, where again a rough outline of the UK can be seen. Low pressure values can be directly compared to areas with high orography in Scotland as shown in Figure 2.1. Finally, the third plot illustrates the temperature distribution across the coarse grid. Similar to the previous plots, the temperature distribution also outlines the fixed variables in the region, with coastal areas appearing relatively warmer and some inland areas cooler.

We then observed the sub-grid samples of specific humidity, pressure, and temperature

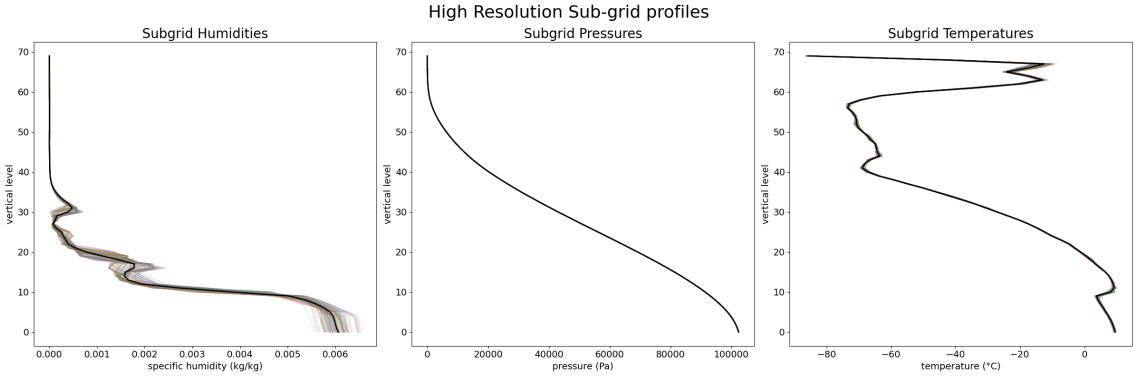

Figure 2.3: High resolution Sub-grid profiles from the first coarse grid box

profiles across the vertical axis. We focused on the 100 sub-grid samples within the first coarse grid box, with the black line representing the low-resolution sample of the coarse grid (mean value of all high-resolution samples) as shown in Figure 2.3. The first plot of the specific humidity profiles show significant variability among the sub-grid samples, particularly in the lower troposphere (vertical levels 0-40). This shows that the many fine-scale humidity variations that are caused by local convective processes or surface moisture sources [6], are underrepresented in the coarse data. The second plot showing the pressure profiles, depict a strong consistency among the sub-grid samples, with all profiles closely aligned with the black coarse grid line. This shows that pressure is relatively uniform across the sub-grid scale, with little fine-scale variability. The temperature profiles, similar to specific humidity, also show noticeable variability among the sub-grid samples, particularly in the lower troposphere. Several sub-grid profiles still indicate potential temperature inversions or varying lapse rates that are not fully captured by the coarse grid average [6]. However, the variability in temperature is generally smaller than the variability observed in specific humidity.

When training a model to generate high-resolution atmospheric profiles, it may be ineffective to rely solely on the absolute profiles of specific humidity, pressure and temperature. As shown in Figure 2.3, there is generally little variability from the coarse grid sample, particularly in pressure and, to a lesser extent, in temperature. If we train the model directly on the absolute profiles, there is a risk that the model might overly focus on reproducing the mean coarse grid profile because of its dominant presence. By training on the differences, we may be able to remove the influence of the mean coarse

grid, encouraging the model to be more sensitive to the subtle variations of the profiles that are critical for high-resolution predictions. This would improve the model's ability to generalise, allowing it to better reproduce the variety of atmospheric profiles seen in high-resolution data. Figure 2.4 shows the residual profiles of specific humidity, pressure and temperature respectively. The specific humidity residuals, as expected from Fig-

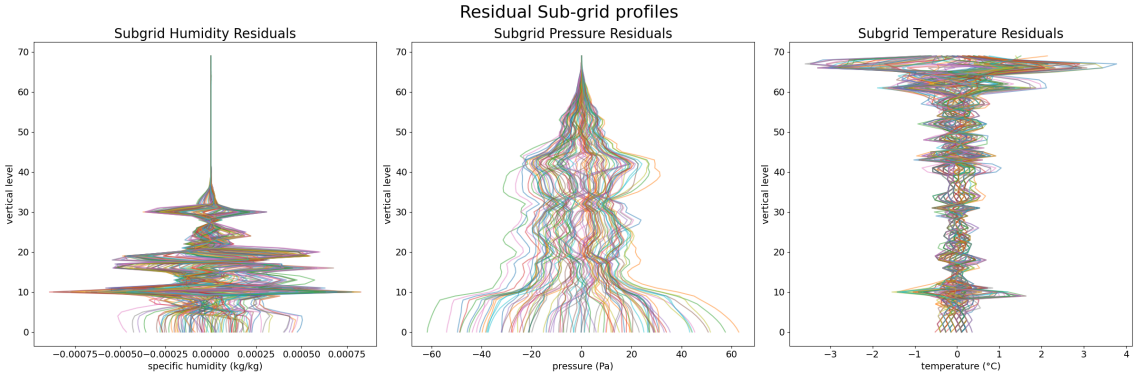

Figure 2.4: Residual Sub-grid profiles from the first coarse grid box

ure 2.3, show significant variability, particularly between vertical levels 10 and 40. The pressure residuals show a more symmetric distribution around the mean. The patterns in deviations can be more clearly seen in this plot as opposed to Figure 2.3, with the greatest variability occurring near the surface and again at around level 30. Temperature residuals also show significant variability, with residuals reaching up to $\pm 3^{\circ}C$ in the upper stratosphere. This further shows that the range of variability at different levels is not captured within the coarse grid. When training we will be training on the absolute profiles, but will be focusing mainly on training on the residual profiles. This approach would aim to reduce bias towards the coarse grid means, improving the model's ability to generalise and capture the fine-scale variability. When the model is tasked with predicting the difference rather than the absolute values, it can allocate more of its capacity to learning complex patterns in the data rather than spending resources on capturing the basic structure (which is already captured by the mean profile). However dealing with residuals carries the risk of increased complexity and noise, making them potentially more challenging to model than absolute values.

Distribution of Variables

The data is deliberately formatted to avoid capturing any horizontal dependencies, which is why the two horizontal dimensions are collapsed into a single dimension. The initial goal is to use a simple unconditional diffusion model to generate samples that are not dependent on any conditions. While this approach helps in forming a basic understanding, it might not be very practical. A more effective method could involve training the model using a conditional diffusion approach. This would entail dividing the data into classes based on specific properties of the data and then training the model conditionally on those classes. This section will explore the distribution of certain conditions and variables within the data. Considering all sub-grid samples across all time points, we have a total of 5,184,000 samples, calculated as $144 \times 100 \times 360$. In the fixed target files, the land-sea fraction is represented by either 0 or 1, while in the fixed input files, this

value ranges between 0 and 1, reflecting the average of the sub-grid samples. Therefore, we select it so that the classification of a sample as land, sea, or coastal is determined by the fixed input files. We observe that 34%, 15.3%, and 50.3% of the samples fall into the sea, land and coastal categories, respectively. It is important to note that many samples close to 0 or 1 might more appropriately belong in the Land or Sea categories. This is shown in Figure 2.5, which includes a simple pie chart displaying the distribution of sub-grid samples across land, sea, and coastal classes, along with a histogram that shows the distribution of land-sea fractions specifically for the samples classified as coastal. The highest density is observed near 0 and 1, indicating that many of the samples are almost entirely land or sea, with fewer samples having land-sea fractions near the mid-range values (e.g., around 0.5). To refine the classification, we adjust the criteria so that the sea class includes samples with a land-sea fraction of less than 0.2, coastal includes samples with a fraction between 0.2 and less than 1, and land is classified as having a land-sea fraction equal to 1. This adjustment results in a new distribution where 15.3% of the samples are classified as land, 33.3% as coastal, and 51.4% as sea.

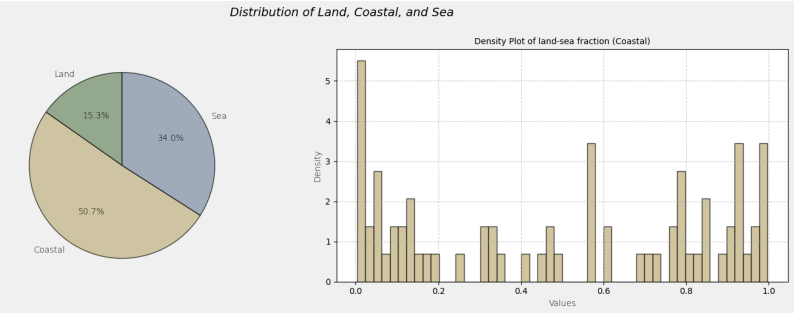

Figure 2.5: Left: Pie chart showing the distribution of sub-grid samples categorised into land, sea, and coastal classes. Right: Histogram of the distribution of land-sea fractions for samples within the coastal class.

We begin by analysing the distribution of orography and the standard deviation of orography within samples classified as either land or coastal, as shown in Figure 2.6. The first plot shows the distribution of orography for land samples, where a significant portion of values are relatively low, centred around 100 metres, with a range extending up to approximately 500 metres. The orography distribution for coastal regions also tends to be low, with the majority of samples falling below 100 metres. However, the range for coastal samples is narrower, around 250 metres, which is about half of the range observed in land samples. The third plot displays the standard deviation of orography for land samples, where most of the values cluster between 50m and 100m. The density increases steadily after 100m, suggesting the variation in orography is more constrained in the higher ranges. The fourth plot shows that the variation in coastal orography is concentrated between 50m and 100m, similar to land but slightly smaller in range. The highest density occurs around 50m to 70m, with fewer values abover 150m.

Next we analyse the presence or absence of tropospheric temperature inversions as a factor for determining data classification. Normally, the temperature in the troposphere decreases with altitude until the stratosphere is reached. However, in some cases, a warm layer of air lies above a cooler one, creating an inversion that hinders vertical mixing in the atmosphere. This is important because such inversions can lead to less correlation between the lower and higher levels of the troposphere, impacting the subgrid profiles.

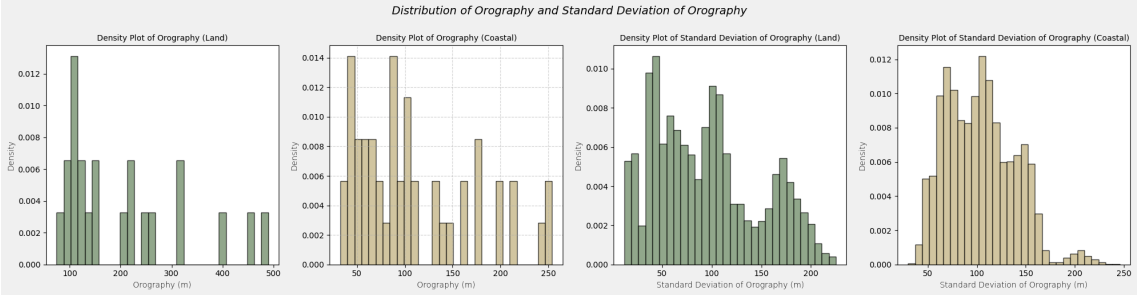

Figure 2.6: Distribution of Orography and the Standard Deviation of Orography for land and coastal samples.

To detect these inversions, the temperature gradients within the troposphere can be computed, checking for any positive gradients, which would indicate an inversion. Figure 2.7 shows on the left that 23.5% of samples contain a tropospheric temperature inversion, while 76.5% do not. The bar chart on the right illustrates how the percentage of samples with inversions varies throughout the day, showing temperature inversions are more common in the afternoon when the sun is strongest, peaking around 16:00.

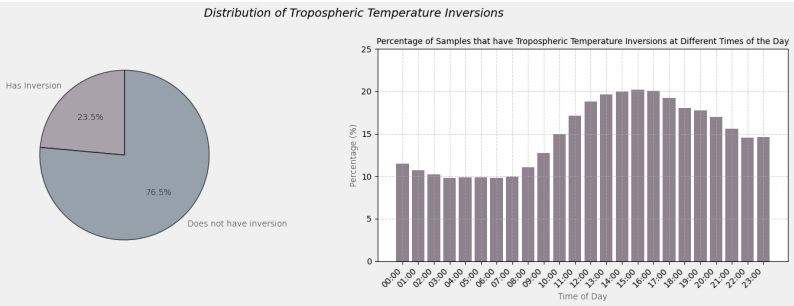

Figure 2.7: The left pie chart shows the percentage of samples with and without tropospheric temperature inversions. The bar chart on the right illustrates the percentage of samples with inversions at different times of the day.

3. Mathematical Background

This chapter provides the theoretical background necessary for understanding the machine learning techniques used in this project. The diffusion model we use is built upon a U-Net architecture, which is a type of deep neural network. To provide context, we begin by explaining key concepts within deep neural networks, focusing on their basic architecture and the process of training through loss functions. The more traditional and widely studied deep neural networks fall under the umbrella of discriminative learning, where the model is trained by learning the direct relationship between input data and labeled outputs. This is achieved by minimising a loss function, which quantifies the difference between the model's predictions and the true labels. Discriminative models, such as standard neural networks and convolutional neural networks are widely studied due to their effectiveness in tasks like classification and regression. However, our focus in this dissertation is on generative modelling, which extends beyond simple prediction to model the underlying data distribution itself. While discriminative models aim to learn clear mappings from inputs to outputs, generative models are trained to generate new data points that resemble the training data, capturing the full range of variability present in the dataset. In Section 3.2, we explore several types of generative models, including autoregressive models, flow-based models, and latent variable models. Among these, diffusion models have emerged as a powerful class of generative learning methods, particularly in capturing fine details and providing more stable trainings. The chapter ends with a detailed description of the mathematical theory behind diffusion models.

3.1 Deep Neural Networks

Deep Learning is a subset of Machine Learning that uses multi-layered neural networks to mimic the structure and function of the human brain. The human nervous system consists of cells referred to as neurons, which are connected by synapses. Changes in external stimuli influence the strength of synaptic connections, causing neural activations within layers. Similarly, artificial neural networks consist of interconnected nodes (or artificial neurons) organised into layers. Each connection in these networks has an associated weight, analogous to the synaptic strength between neurons. A diagram of a biological neuron versus an artificial neural network is shown in Figure 3.1. These artificial neurons receive input data, process it through weighted connections, and apply activation functions to determine the output. The process of adjusting the weights of these connections during training helps the network learn from data, improving its ability to make accurate predictions or decisions. This training process mimics the brain's learning by adjusting synaptic strengths, effectively enabling the neural network to model complex patterns and relationships in large datasets [19].

The recent successes of neural networks are largely due to the exponential growth in computing power and the abundance of available data, which have surpassed the limitations of traditional machine learning systems. While conventional machine learning may still excel with smaller datasets due to the ease of interpreting models and creating understandable features, it falls short when faced with high-dimensional data or larger datasets. In contrast, deep learning effectively maps input vectors to output vectors given sufficiently large models and labelled training data. Additionally, chip manufacturers are now tailoring their GPUs specifically for the core operations of deep learning [13].

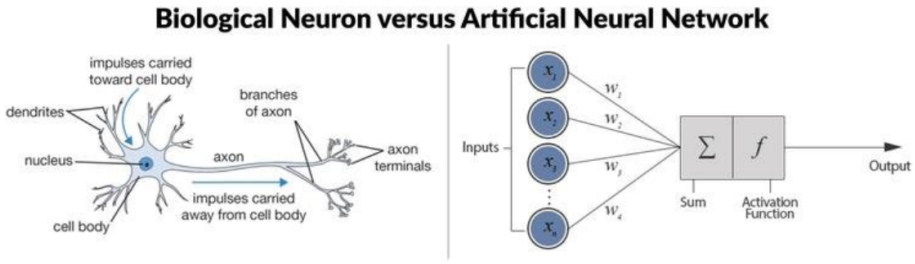

Figure 3.1: Diagram of the structure of a biological neuron versus an artificial neural network [14]

3.1.1 Feed-forward Neural Networks

Basic Architecture

A single computational layer is often referred to as the perceptron, thus feed-forward neural networks are also known as multi-layer perceptrons [20]. An L-layer feed-forward neural network consists of three types of layers:

- The *input layer* receives the initial data and passes it onto the next layer. Since there is no processing here, this layer is not formally counted as part of the L layers. This layer contains the features of the input data points, $x^{(i)} \in \mathbb{R}^n, i = 1, \cdots, m$.
- In an L-layer network, there are $L-1$ *hidden layers*, where each layer processes the data using linear and non-linear activation functions. $a_j^{(L)}$ denotes the j^{th} node of the L^{th} layer. The *width* of the network is the number of nodes within the largest hidden layer. Each input is multiplied by a *weight* and a *bias* is added, which are parameters the network learns during training. Denote $w_{kj}^{(L)}, b_k^{(L)} \in \mathbb{R}$, the weight and bias of the connection between the j^{th} node of the $(L-1)^{th}$ layer and the k^{th} node of the l^{th} layer. Thus for a given training example $x^{(i)}$, the weighted sum is calculated as follows:

$$z_k^{(L)(i)} = \sum_{j=1}^{n^{L-1}} w_{kj}^{(L)} a_j^{(L-1)} + b_k^{(L)} = (w_k^{(L)})^\top a^{(L-1)} + b_k^{(L)}$$

  After calculating the weighted sum, the neuron applies a non-linear activation function, σ to produce the output of the neuron. This is analogous to the neuron firing in a human brain [13]. Common activation functions include:

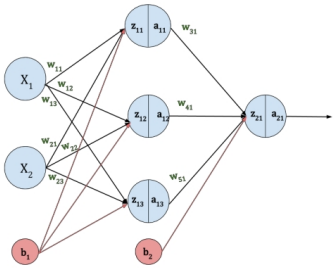

Figure 3.2: Example of a feedforward neural network with L=2 [15]

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- ReLU (Rectified Linear Unit):$\sigma(z) = max(0, z)$
- Tanh: $\sigma(z) = tanh(z)$

Plots of these activation functions can be found in Figure 3.3. The choice of activation function is usually dependent on the task. The output of one neuron becomes the input for the neurons in the next layer, which continues through the hidden layers until the output layer.

- The *output layer* produces the final output of the network, usually a prediction or classification. The total number of layers, L is known as the *depth* of the model.

To summarise, the network computes the function $f : \mathbb{R}^d \to \mathbb{R}$ in the following way: Let $\mathbf{x} \in \mathbb{R}^d$ be the input, and $\mathbf{z_0} = \mathbf{x}$. In matrix form, the network outputs

$$\widehat{\mathbf{y}} = f_\theta(\mathbf{x}) = \mathbf{z_L} = \sigma_L(\mathbf{W}^{(L)}\sigma_{L-1}(\mathbf{W}^{(L-1)}\sigma_{L-2}(\cdots) + \mathbf{b}^{(L-1)}) + \mathbf{b}^{(L)})$$

where

$$\mathbf{W}^{(L)} = \begin{bmatrix} \left(\mathbf{w}_1^{(L)}\right)^\top \\ \left(\mathbf{w}_2^{(L)}\right)^\top \\ \vdots \\ \left(\mathbf{w}_{n^{(L)}}^{(L)}\right)^\top \end{bmatrix} \in \mathbb{R}^{n^{(L-1)} \times n^{(L-1)}} \quad \text{and} \quad \mathbf{b}^{(L)} = \begin{bmatrix} b_1^{(L)} \\ b_2^{(L)} \\ \vdots \\ b_{n^{(L)}}^{(L)} \end{bmatrix} \in \mathbb{R}^{n^{(L)}}$$

are the weight matrix and the bias vector of the L^{th} layer respectively. Activation function denoted by $sigma_L : \mathbb{R} \to \mathbb{R}$. This can be seen as a composition of functions connected in a chain, where $\theta := \{(\mathbf{W^{(1)}}, \mathbf{b^{(1)}}), \cdots (\mathbf{W^{(L)}}, \mathbf{b^{(L)}})\}$ denotes the set of all trainable parameters [13].

Training

In order to obtain a predicted value $\widehat{y}^{(i)}$ as close as possible to the true value $y^{(i)}$, the network is trained through an optimisation process. A loss function $\mathcal{L}$ is formulated to be minimised on the training set to achieve an accurate approximation of the relationship between the input vectors and their associated outputs. The learning process aims to minimise the *expected risk*, which is the expectation of the loss function with respect to the underlying probability distribution $p(x, y)$. As the underlying distribution of the data is unknown, training is conducted through *empirical risk minimisation* which simply minimises the average of the loss function on the available samples of the training set.

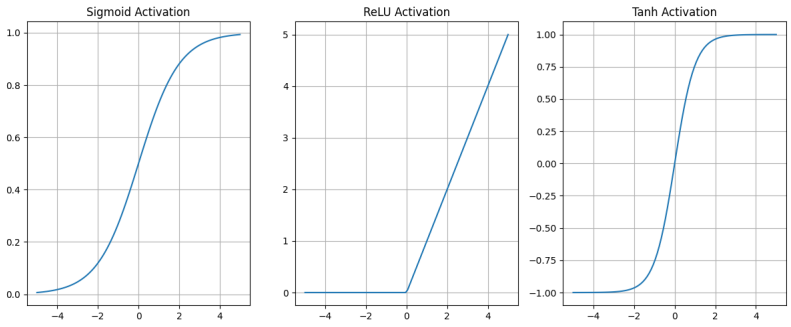

Figure 3.3: Plots of common activation functions [16]

Denote $\mathcal{J} : \Theta \to \mathbb{R}$, the *training error*. Thus the learning process comes down to finding the optimal parameter θ, so that the $f_\theta(x)$ minimises $\mathcal{J}$, solving the following minimisation problem:

$$\min_\theta \mathcal{J}(\theta) = \min_\theta \left\{ \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f_\theta(x^{(i)}), y^{(i)}) \right\}$$

In discriminative learning tasks, common choices for the loss function is the *Hinge loss* or the *0-1 indicator function* for classification tasks or the *mean squared error* for regression tasks.

Training works by computing an output $\hat{\mathbf{y}}$ given an input $\mathbf{x}$ based on the current set of parameters θ, which is then compared to the true output . This is known as *forward propagation*. Through *back propagation*, the gradients of the loss function are computed using the chain rule and differential calculus. These computed gradients are used to update the parameters where $\hat{\mathbf{y}}$ is computed again using the updated parameters. This process is repeated until resulting loss is as desired [21].

Minimising empirical risk is challenging, as closed form solutions are often impractical. Instead, optimisation is typically done iteratively using gradient-based methods, which adjust parameters so that the objective function $\mathcal{J}(\theta_k)$ decreases over time. Gradient descent, the simplest method, updates parameters by taking small steps in the opposite direction of the gradient, as this is the steepest descent direction [22]. Formally, the update is:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta \mathcal{J}(\theta_k)$$

where $\nabla_\theta \mathcal{J}(\theta_k)$ is the gradient of the objective function, and α_k is the learning rate in deep learning.

3.1.2 Convolutional Neural Networks (CNNs)

One of the most widely used deep learning models is the class of convolutional neural networks (CNNs). CNNs are specifically designed to handle multidimensional grid-like inputs that exhibit strong spatial dependencies, particularly in localised regions. This makes them especially effective for image related tasks, as images often display spatial relationships (such as neighbouring pixels having similar colour values). CNNs are also applicable in other areas like video recognition, natural language processing, and financial time series analysis, all one-dimensional data inputs analogous to our dataset.

Like fully connected neural networks, CNNs can be trained using gradient descent techniques, the back-propagation algorithm, and the optimisation strategies. The key difference between CNNs and fully connected networks lies in the use of convolution operations instead of general matrix multiplication in at least one of their layers. This convolution based architecture offers several advantages, including computational efficiency, which has contributed to its success across numerous applications [13].

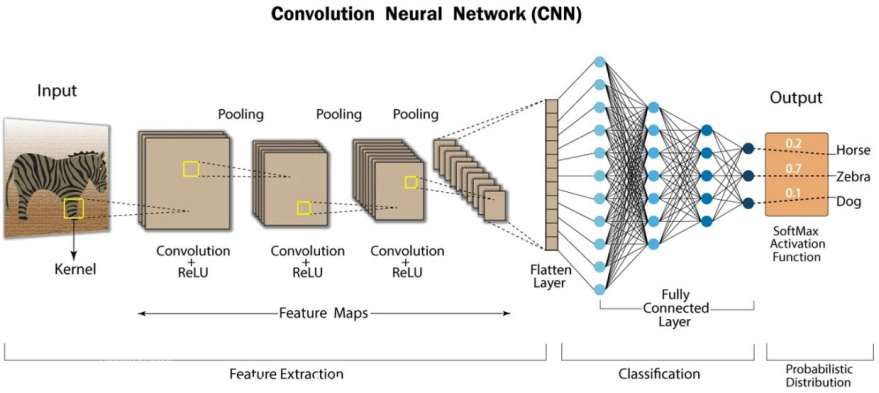

Figure 3.4: Diagram of a CNN architecture for image classification [17]

Generally, a CNN involves two types of operation, convolutions (followed by a non-linear operation e.g. ReLU) and pooling. The last layer may also consist of a final set of fully connected layers, depending on the task. The 1D discrete convolution is denoted by $*$ of the two sequences $\mathbf{f} = (f_j)_{j\in\mathbb{Z}}$ and $\mathbf{h} = (h_j)_{j\in\mathbb{Z}}$, defined as follows:

$$(f * h)_j = \sum_{j'=-\infty}^{\infty} f_{j'} h_{j+j'}$$

This can be interpreted as the weighted average of $\mathbf{f}$, which gives more importance to certain values based on the corresponding values of $\mathbf{h}$. The first argument is referred to as the *input*, and the second as the *filter*, or *kernel*. The output of the convolution is known as the *feature map*. The infinite summation is actually calculated as a finite summation as it is assumed these functions are zero everywhere except where the values of the input and kernel are stored. Commonly, convolutions in CNNs involve multidimensional arrays, known as *tensors*, which have a *width*, *height* and *depth*. The kernel is a multidimensional array containing parameters adapted by the learning algorithm. In practice, a convolution is computed by placing the kernel at each possible position in the input (image or hidden layer), performing a dot product between the filter and the input grid. The number of these alignments determines the spatial dimensions of the next hidden layer [20].

For instance, a 2D convolution with a filter reduces the size of the output layer compared to the input. For a tensor of size $n_H \times n_W$ convolved with an $f \times f$ kernel, the output size becomes $(n_H - f + 1)(n_W - f + 1)$. While this reduction might cause information loss, especially along borders, it can also be useful in memory constrained situations. Fortunately, there are ways to adjust the output size if needed. *Padding* involves adding rows and columns of zeros around the input to widen it, allowing the filter to extend beyond the borders during convolution. The padded zeros don't affect the dot product,

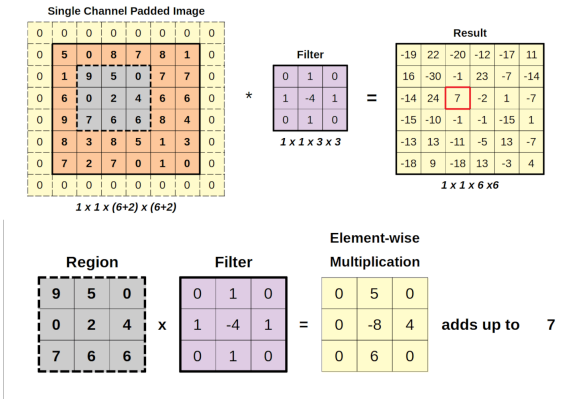

Figure 3.5: Illustration of a 2D convolution [18]

and padding helps control the output size. If p is the padding per side, the output size becomes $(n_H + 2p - f + 1) \times (n_W + 2p - f + 1)$. Common padding types include "valid" (no padding) and "same" (where $2p = f - 1$, making the output size equal to the input). Stride defines how many pixels the filter shifts after each operation. While a stride of 1 is common, larger strides can reduce the spatial resolution and help with overfitting or memory constraints [13]. For stride s, the output size of a padded, strided convolution is

$$\left(\frac{n_H + 2p - f}{s} + 1\right) \times \left(\frac{n_W + 2p - f}{s} + 1\right)$$

Basic Architecture

To formalise convolution in a CNN, simplifying the notation, we focus on a single training example, but extending this to a batch scenario only requires adding a fourth dimension. Let $W^{[q,l]} = (w_{ijk}^{[q,l]})$ represent the tensor of trainable parameters for the q^{th} filter in the l^{th} layer (where i, j, k represent the positions across the height, width and depth of the filter). Filters have dimensions $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$, where the number of channels (the depth) must be the same as the number of channels from the previous layer. The feature maps in layer l are denoted by $Z^{[l]} = (z_{ijk}^{[l]}$, with size $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$. The convolution of the q^{th} filter with the feature maps produces the output for the q^{th} feature map in the next layer, $l + 1$, as the following:

$$z_{ijq}^{[l+1]} = \sum_{u=1}^{f^{[l]}} \sum_{v=1}^{f^{[l]}} \sum_{k=1}^{n_C^{[l]}} w_{uvk}^{[q,l+1]} z_{i+u-1,j+v-1,k}^{[l]}$$

for all $i = 1, \cdots, n_H^{[l]}, j = 1, \cdots, n_W^{[l]}$ and $q == 1, \cdots, n_C^{[l+1]}$, where $n_H^{[l]} = \left(\frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1\right)$, and $p^{[l]}$, $s^{[l]}$ denote the padding and stride at layer l. A CNN layer contains a large number of filters (ranging from hundreds to thousands), which are typically small in size (e.g., 3x3, 5x5, or 7x7). More filters increase the number of trainable parameters and the network's capacity. It's common to add a bias term to each 2D feature map, so at layer l, the bias has dimensions of $1 \times 1 \times n_C^{[l]}$ (each filter has a unique bias). Similar to feed-forward networks, a non-linear function is applied after convolution to each feature map without

altering their dimensions [13]. The following equations describe one CNN layer:

$$z_{ijq}^{[l+1]} = \sum_{u=1}^{f^{[l]}} \sum_{v=1}^{f^{[l]}} \sum_{k=1}^{n^{[l]}_C} w_{uvk}^{[q,l+1]} a_{i+u-1,j+v-1,k}^{[l]} + b_q^{[l+1]}$$

$$a_{ijq}^{[l+1]} = g^{[l+1]}(z_{ijq}^{[l+1]})$$

where $A^{[l]} = (a_{ijk}^{[l]})$ is the activation tensor with size $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$.

A CNN functions similarly to a traditional fully connected neural network, as discrete convolutions can be seen as multiplications by sparse, structured matrices. These matrices have mostly zero entries, with some non-zero entries that are equal to each other due to the structure of the kernels, which are smaller than the input image. Unlike traditional feed-forward networks, CNNs handle variable input sizes and leverage sparse connectivity, parameter sharing, and translation invariance. Sparse connectivity reduces trainable parameters by using kernels smaller than the input, allowing for fewer operations. Parameter sharing reuses the same parameters across multiple input positions, reducing storage needs. Translation invariance ensures that the network responds consistently to shifted inputs, making CNNs particularly effective in tasks like image processing. These features result in lower memory requirements and improved efficiency compared to fully connected networks [13].

In CNNs, pooling layers are used to reduce the spatial size of feature maps without losing much information. Unlike convolutional layers, pooling layers have no trainable parameters and apply the pooling function independently to each feature map, preserving depth. Common pooling methods include max pooling, which selects the maximum value in a region, and average pooling, which takes the mean. Pooling layers act as downsampling kernels, with hyperparameters such as stride (s) and size (f). The output dimensions of a pooling layer are given by the formula:

$$\left(\frac{n_H + 2p - f}{s} + 1\right) \times \left(\frac{n_W + 2p - f}{s} + 1\right) \times n_C$$

Different CNN architectures are designed for various tasks and have unique features. *LeNet*, one of the earliest CNNs, was developed for digit recognition and features a simple architecture with alternating convolutional and pooling layers. *AlexNet* revolutionised image classification with its deep structure and use of ReLU activations, significantly improving performance in the ImageNet competition. *ResNet* introduced residual connections, allowing very deep networks to be trained effectively by mitigating the vanishing gradient problem. *U-Net*, designed for biomedical image segmentation, features an encoder-decoder structure with skip connections to capture fine grained details. Each of these architectures addresses specific challenges and has influenced the development of more advanced models [13]. We will explore U-Nets in more detail, focusing on their architecture and applications in segmentation tasks.

3.1.3 U-Nets

The U-Net architecture was first introduced by O. Ronneberger, P. Fischer and T. Brox in their paper titled "*U-Net: Convolutional Networks for Biomedical Image Segmentation*"[23]. Unlike traditional CNNs that typically output a single class label for an entire image,

U-Nets enable dense pixel wise predictions, making them highly suitable for segmentation tasks where each pixel of an image needs to be classified. The U-Net architecture builds on the strengths of fully convolutional networks but introduces a unique symmetric expanding path, which gives it a ”U” shape as shown in Figure 3.6. This U-shape is characterised by an encoder-decoder structure, where the encoder path captures context through successive convolutions and down-sampling, while the decoder path performs up-sampling and combines it with high-resolution features from the encoder. This design allows U-Nets to achieve high accuracy even with a relatively small number of training images, a significant advantage in the biomedical domain where annotated data is often limited.

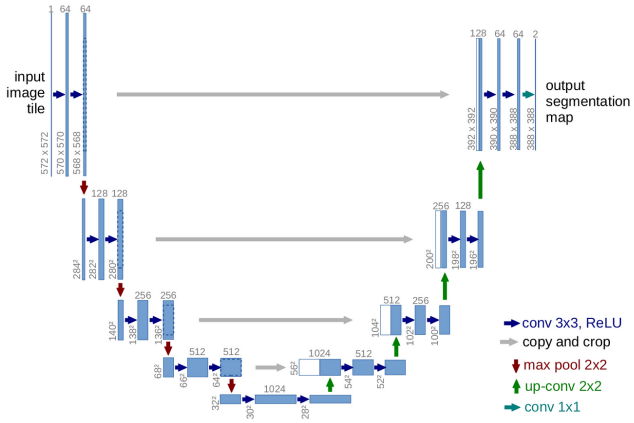

Figure 3.6: Illustration of the U-net architecture [23]

The network architecture, shown in Figure 3.6, is composed of two main parts: a contracting path on the left side and an expansive path on the right. The contracting path follows the standard design of a convolutional network, involving repeated applications of two unpadded 3x3 convolutions, each followed by a ReLU and a 2x2 max-pooling operation with a stride of 2 for downsampling. With each downsampling step, the number of feature channels is doubled. In the expansive path, each step begins with upsampling the feature map, followed by a 2x2 convolution (referred to as an ”up-convolution”) that reduces the number of feature channels by half [23]. This is then concatenated with the correspondingly cropped feature map from the contracting path. The process continues with two 3x3 convolutions, each followed by a ReLU. Cropping is necessary to account for the loss of border pixels in each convolution. The final layer uses a 1x1 convolution to map each 64-component feature vector to the desired number of classes. Overall, the network contains 23 convolutional layers. To improve the architecture’s performance, skip connections are a crucial component of the U-net. These involve copying the activation values from the left side to the right side of Figure 3.6, meaning from earlier layers to later layers with the same dimensions (as indicated by the grey arrows). This approach is beneficial because it enables the network to transfer detailed spatial information to the later layers, which might otherwise be lost during the contracting path [13].

3.2 Generative Modelling

Generative modelling is seen as the next frontier in machine learning because it goes beyond the traditional capabilities of discriminative modelling, which has driven much of the progress over the past two decades. While discriminative models are effective for tasks like classification or regression, they are less suitable for our project because they are designed to predict a single, specific outcome given an input, rather than understanding the latent structure of the data. In contrast, our project requires a model that can generate multiple plausible outcomes based on input data, reflecting the stochastic nature of atmospheric processes. A discriminative model would fail to account for this variability, as it focuses on minimising error by finding the most probable single output, which may not adequately capture the full range of possible atmospheric behaviours. Generative models are gaining popularity due to their ability to not only categorise but also create new data, as demonstrated by projects like Meta's Make-A-Video, OpenAI's GPT-4, and DALL-E 3. These models are opening up new possibilities in fields such as game design, healthcare, reinforcement learning, and beyond [25].

Generative modelling focuses on modelling the underlying distribution of data to generate new samples that resemble the training data. A generative model needs to be probabilistic rather than deterministic. If the model is just a fixed calculation, like averaging the pixel values in a dataset, it cannot be considered generative since it always produces the same output. To truly be generative, the model must incorporate a stochastic component that affects the variability in the samples it generates [24]. Unlike discriminative modelling, generative models are typically associated with unsupervised learning, where the model learns patterns from an unlabelled dataset. However, generative models can also be applied to labelled datasets, allowing them to generate observations for each distinct class. Discriminative modelling estimates the probability $p(y|\mathbf{x})$ of a label y given observation $\mathbf{x}$ whereas generative modelling estimates the probability of observing an observation $\mathbf{x}$, $p(\mathbf{x})$ [25]. The general process for generative modelling can be found in Figure 3.7

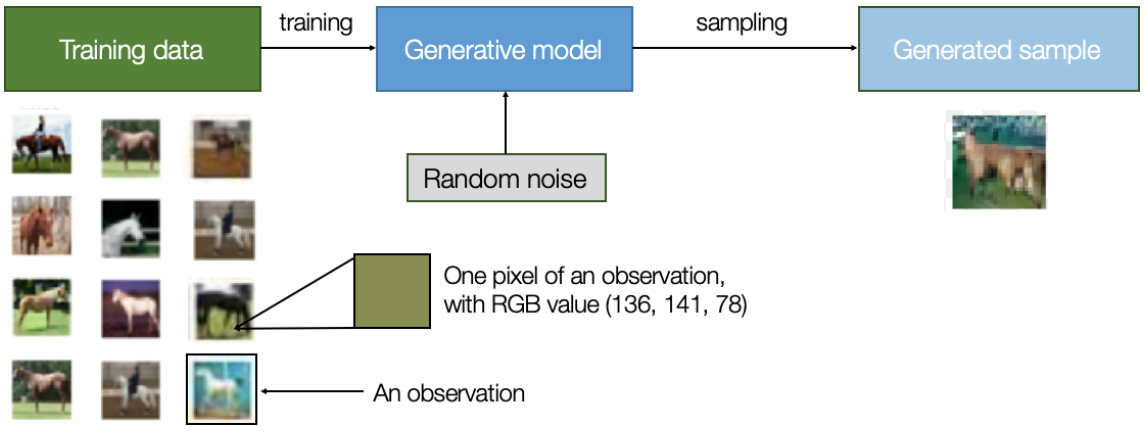

Figure 3.7: The generative modelling process [25]

The Generative Modelling Framework

We start with a dataset of observations X and assume these observations were generated by some unknown distribution, p_{data}. A generative model, p_{model}, aims to replicate p_{data}. If successful, we can sample from p_{model} to create observations that resemble those from p_{data}. A generative model is considered effective if:

1. It can generate examples that convincingly appear as though they were drawn from p_{data}.
2. It produces examples that are sufficiently distinct from the original observations in X, rather than merely replicating them [25].

3.2.1 Types of Deep Generative Models

Generative modelling can be implemented without the use of neural networks, however neural networks are often used due to their flexibility and power, making them a popular choice for parameterising generative models. This is known as *deep generative modelling* [24]. Deep generative modelling can be split into four main groups:

- Autoregressive generative models (ARM)
- Flow-based models
- Latent variable models
- Energy-based models

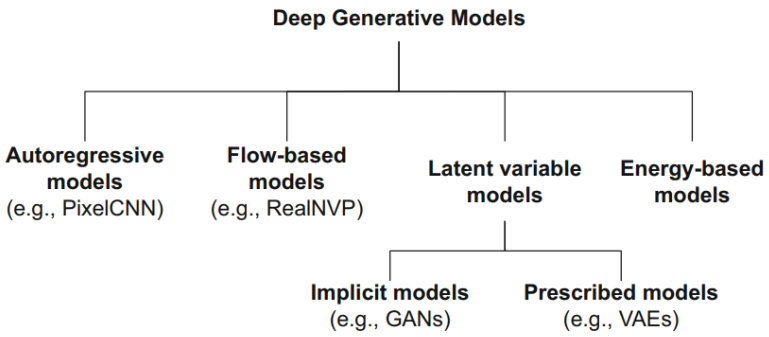

Figure 3.8: The four main types of deep generative models [24]

Autoregressive Generative Models (ARMs)

Autoregressive models generate data one component at a time, sequentially predicting each part of the data conditioned on the previous parts. These models decompose the joint probability distribution of the data into a product of conditional probabilities.

$$p(x) = \prod_{i=1}^{D} p(x_i | x_1, x_2, \ldots, x_{i-1})$$

Here, x is a data point with D dimensions (e.g., pixels in an image or words in a sentence), and each $p(x_i | x_1, x_2, \ldots, x_{i-1})$ represents the probability of the i-th component given all

previous components [24]. Examples are PixelCNN, PixelRNN or GPT (for text).

Flow-based Models

Flow-based models use invertible transformations to map a simple base distribution (e.g., a Gaussian) to a complex data distribution. These transformations are designed to allow exact computation of the data likelihood and efficient sampling.

$$p(x) = p(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

where $z = f(x)$ is the transformation of the data point x into the latent variable z, f is the invertible function, and the determinant term accounts for the change in volume under the transformation [24]. Examples are RealNVP, Glow, NICE.

Latent Variable Models

Latent variable models introduce additional hidden (latent) variables z to explain the observed data x. These models generate data by first sampling the latent variables and then generating the data conditioned on these variables. This often involves learning a distribution over z and a mapping from z to x.

$$p(x) = \int p(x|z)p(z)\,dz$$

where $p(z)$ is the prior distribution over the latent variables, and $p(x|z)$ is the likelihood of the data given the latent variables [24]. Examples include Variational Autoencoders (VAEs), Diffusion Models, Generative Adversarial Networks GANs (with implicit latent variables).

Energy-based Models (EBMs)

Energy-based models assign an energy score to each configuration of the data, with lower energy corresponding to more likely configurations. The probability distribution is defined in terms of these energy scores. Unlike other models, EBMs do not directly model probabilities but rather focus on the relative energy of different states.

$$p(x) = \frac{\exp(-E(x))}{Z}$$

where $E(x)$ is the energy function that assigns a scalar energy to each data point x, and $Z = \int \exp(-E(x))dx$ is the partition function that normalises the distribution [24]. Examples include Boltzmann Machines, Deep Energy Models, Contrastive Divergence.

Each of these generative modelling approaches has its own strengths and is suitable for different types of data and tasks. The choice of model depends on factors like the complexity of the data distribution, the need for exact likelihood computation, and the desired properties of the generated samples. Latent variable models are the best choice

for our project due to their ability to effectively simulate the data's true distribution, represent uncertainty, and generate diverse atmospheric profiles. These are all key features we look for modelling the uncertainty associated with sub-grid scale atmospheric processes in its vertical structure. ARM models generate data sequentially, meaning each output is conditioned on the previous one. While this is effective for tasks such as text and audio generation, it is not so useful for our problem. Also, the sequential nature of ARMs can be computationally expensive and slow for generating high-dimensional data. Flow-based models, while extremely useful for anomaly detection tasks, tend to focus on the more common modes of the data distribution, making it less suitable for our case where we would like to steer away from simply outputting the coarse grid mean. EBMs are known for being difficult to train due to their reliance on optimisation of energy functions, which can be unstable and slow, especially when dealing with high-dimensional data. They typically require significant computational resources and time to converge to meaningful solutions [24].

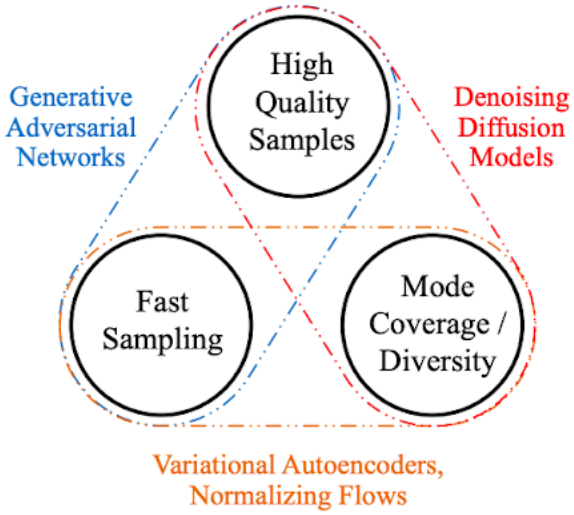

Figure 3.9: The generative learning trilemma [12]

Three commonly used latent variable models are Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and diffusion models. These models ideally need to balance three main factors: high-quality sampling, mode coverage and sample diversity, and fast, computationally efficient sampling. This balancing act is referred to as the *generative learning trilemma*, shown in Figure 3.9. The challenge imposed by this trilemma is that existing methods often make trade-offs and cannot satisfy all requirements simultaneously [12]. First, high quality output generation is essential for this project, as accurately modelling sub-grid processes requires fine detail and precision. Diffusion models are particularly strong in this area, surpassing other generative approaches like Variational Autoencoders. Ensuring diversity in generated outputs, or mode coverage, is equally important. Diffusion models are particularly effective at learning the full range of variability, avoiding the problem of mode collapse that is more common in GANs. Finally, computational efficiency is important for real-time applications. While diffusion models are generally more computationally intensive, recent advancements have improved their speed and efficiency. The trilemma presents the challenge in optimis-

ing all three factors simultaneously, but diffusion models offer the best balance for this project's needs.

3.2.2 Diffusion Models

Diffusion probabilistic models were first introduced in a physics paper in 2015 by Sohl Dickstein et al. [27]. A diffusion model is a parameterised Markov chain trained to generate data samples. It learns transitions that reverse a diffusion process, which gradually adds noise to data until it is fully corrupted. When this noise is Gaussian, the model can use simple neural networks for parameterisation [31]. Although diffusion models are straightforward to define and train, their ability to generate high-quality samples was not demonstrated well until Ho et al.'s paper *Denoising Diffusion Probabilistic Models (DDPM)* in 2020 [28]. This setup involves two key processes:

1. A fixed forward diffusion process, denoted as q, where Gaussian noise is progressively added to an image until it becomes pure noise.

2. A learned reverse denoising diffusion process, denoted as p_θ, in which a neural network is trained to gradually remove the noise from an image, starting from pure noise and eventually reconstructing the original image.

Both the forward and reverse processes, indexed by t, occur over a finite number of time steps T. The process begins at $t = 0$ with a real image x_0 sampled from your data distribution. At each time step t, Gaussian noise is added to the image from the previous step. With a sufficiently large T and a well designed noise schedule, the image gradually transforms into an isotropic Gaussian distribution by $t = T$ [29]. Let $q(x_0)$ represent the

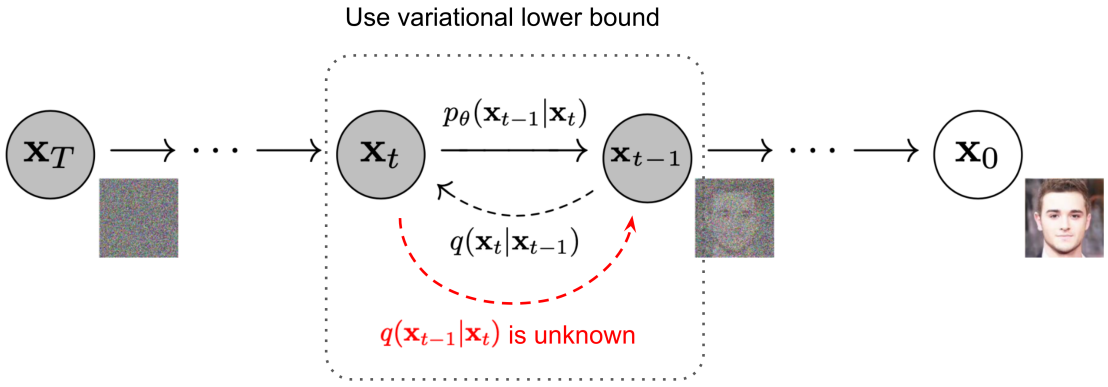

Figure 3.10: Markov chain of forward and reverse diffusion process [30]

real data distribution, from which we sample an image $x_0 \sim q(x_0)$. The forward diffusion process $q(x_t \mid x_{t-1})$ adds Gaussian noise at each time step t, following a known variance schedule $0 < \beta_1 < \beta_2 < \cdots < \beta_T < 1$:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t I)$$

Each noisy image at time step t is sampled from a conditional Gaussian distribution with mean $\mu_t = \sqrt{1-\beta_t}\, x_{t-1}$ and variance $\sigma_t^2 = \beta_t$, where x_t is obtained by sampling $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and setting:

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon$$

The β_t values vary over time according to a "variance schedule," which can be linear, quadratic, cosine, etc. Starting from x_0, we generate $x_1, \ldots, x_T$, where x_T is pure Gaussian noise if the schedule is set appropriately. To reverse this process, we need the conditional distribution $p(x_{t-1} \mid x_t)$. Since it's intractable, we approximate it using a neural network, denoted $p_\theta(x_{t-1} \mid x_t)$, where θ represents the network parameters, updated via gradient descent.

We need a neural network to model the conditional probability distribution of the reverse process. Assuming this reverse process is also Gaussian, it can be defined by the mean, parameterised by μ_θ and a variance, parameterised by Σ_θ. This gives us the process:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

where the mean and variance depend on the noise level t. However, in Ho et al.'s paper, the variance was fixed and the neural network only learned the mean μ_θ of this conditional distribution. To derive an objective function for learning the mean of the reverse process, the combination of q and p_θ is treated as a variational auto-encoder. They use the variational lower bound (ELBO) to minimise the negative log-likelihood with respect to the ground truth data sample x_0. The ELBO for this process is a sum of losses across each time step t, expressed as $L = L_0 + L_1 + \cdots + L_T$. Each loss term (except L_0) is the KL divergence between two Gaussian distributions, which can be simplified to an L2-loss for the means. A key result from the forward process q is that we can directly sample x_t at any noise level conditioned on x_0, since sums of Gaussians remain Gaussian. This is given by:

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I)$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$. This allows us to sample Gaussian noise, scale it, and add it to x_0 to get x_t directly [30]. The $\bar{\alpha}_t$ values, being functions of the known variance schedule β_t, can be precomputed. During training, this enables optimisation of random loss terms L_t by randomly sampling t. An advantage of this property, as demonstrated by Ho et al., is that instead of directly predicting the mean, we can reparametrise the process to have the neural network learn to predict the added noise, $\epsilon_\theta(x_t, t)$, for each noise level t in the KL loss terms. This effectively turns the neural network into a noise predictor rather than a mean predictor. The mean can then be computed as:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$$

The final objective function L_t for a random time step t, given $\epsilon \sim \mathcal{N}(0, I)$, is:

$$\|\epsilon - \epsilon_\theta(x_t, t)\|^2 = \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t)\|^2$$

Here, x_0 is the original uncorrupted image, and ϵ is the pure noise sampled at time step t. The neural network $\epsilon_\theta(x_t, t)$ is simply optimised using the mean squared error between the true noise and the predicted noise [29].

4. Model

While we have discussed the mathematical theory behind diffusion models, their specific implementation within a neural network and architectural details has not yet been addressed. This chapter aims to explain the employed model, detailing the computational demands and the training process. Following the processing and analysis of our dataset, we are now prepared to select a suitable model for training our data and generating sub-grid samples. For this purpose, we use the `denoising_diffusion_pytorch` package by Phil Wang, which is available publicly on GitHub at this link. This package implements the denoising diffusion probabilistic model in `PyTorch`, inspired by the original `Tensorflow` implementation discussed in Ho et al.'s 2020 paper "*Denoising Diffusion Probabilistic Model (DDPM)* [28], both popular deep learning libraries used in Python programming language. Given that our dataset comprises one-dimensional data samples, we use specific classes from this package, namely `Unet1D`, `GaussianDiffusion1D`, and `Trainer1D`.

4.1 Denoising Diffusion Probabilistic Model

The neural network used in our model is required to take in a noised image at a designated timestep and compute the predicted noise. Importantly, this predicted noise is a tensor that maintains the same dimensions as the input image, which means the network processes and outputs tensors of identical shape. The architecture is similar to an Autoencoder, which are characterised by a "bottleneck" layer in between the encoder and decoder. The encoder compresses the image into a compact hidden representation termed the "bottleneck" and then the decoder reconstructs the image from this condensed form. This design forces the network to preserve only the most crucial information within the bottleneck. For the architecture of our diffusion model, the DDPM authors opted for a U-Net configuration, first introduced by Ronneberger et al. in 2015 [27]. Similar to an autoencoder, the U-Net includes a central bottleneck that ensures the network learns only the most vital data. Additionally, it incorporates residual connections between the encoder and decoder, significantly enhancing the flow of gradients, a concept inspired by the ResNet architecture proposed by He et al. [37] in 2015.

First, we set up some helper functions and classes that we need for building the neural network. One important component we include is the `Residual` module. This module takes a function's output and adds it back to its input, effectively creating what's called a residual connection. As the neural network operates across various noise levels over time, it utilises sinusoidal position embeddings to encode these levels, drawing inspiration from the Transformer architecture described by Vaswani et al. in 2017 [38]. This embedding technique enables the network to recognise the specific time step, or noise level,

it is addressing at any point in the batch. The module, `SinusoidalPositionEmbeddings`, processes a tensor with dimensions (batch_size, 1), representing the noise levels of different samples within a batch. It transforms this tensor into a new shape, (batch_size, dim), where "dim" refers to the size of the embedding. These embeddings are subsequently integrated into each residual block to maintain awareness of the noise level during processing. Next, we set up the core component of the U-Net model, known as the Wide ResNet block, which was originally introduced by Zagoruyko et al. in 2016 [39]. Following this, we integrate an attention module between the convolutional layers, a technique central to the Transformer architecture by Vaswani et al. in 2017 [38]. Phil Wang adapted two types of attention for this model: the standard multi-head self-attention used in Transformers, and a linear attention variant by Shen et al. in 2018 [40]. Finally, the authors of the DDPM enhance the convolutional and attention layers of the U-Net by incorporating group normalisation, as proposed by Wu et al. in 2018 [41].

Now that we have outlined all the fundamental components, including position embeddings, ResNet blocks, attention, and group normalisation [29], we can construct the complete Unet1D neural network. The network is built up as follows:

- Initial Setup: The network starts with a convolutional layer that prepares the input for subsequent processing. It includes a self-condition option that doubles the input channels if activated.
- Position and Time Embeddings: Depending on the configuration, the network uses either random Fourier features or learned sinusoidal position embeddings to encode the time variable effectively. This setup helps the model understand at which point in the diffusion process it is operating.
- Downsampling Path: The network has a series of downsampling stages. Each stage consists of two ResNet blocks followed by a linear attention and a downsampling operation using either a strided convolution or a direct convolution if it's the final stage in the sequence.
- Middle Block: At the deepest layer, the model processes data through additional ResNet blocks and an attention layer designed to focus on the most crucial features at the lowest resolution.
- Upsampling Path: Mirroring the downsampling stages, the upsampling path includes sequences of ResNet blocks and linear attention layers, combined with upsampling operations to progressively restore the data to its original dimension.
- Final Layers: The model concludes with a final ResNet block and a convolutional layer that outputs the tensor in the desired dimensions, adjusted for whether the model has learned to predict variance in the data.

We initialise the U-Net architecture with the following code snippet:

```
model = Unet1D(dim = dim_data,
               dim_mults = (1, 2, 4, 8),
               channels = n_channels)
```

In this setup, `dim_data` represents the dimension of each vector, which must be a power of 2 to meet the architectural requirements of `Unet1D`. This constraint potentially affects the layer dimensions and the number of layers due to how downscaling and upscaling

are handled. The `n_channels` specifies the number of channels, indicating the depth of input features the model will process, while `dim_mults` refers to the multiplication factors for dimensions across layers. Our dataset dimensions are (5184000, 3, 70), where 5184000 denotes the number of samples, 3 is the number of features (representing our temporal variables: humidity, pressure, and temperature), and 70 is the vector dimension. To align with the `Unet1D` model's requirements, we truncate our vectors to a length of 64.

In the setup below, we configure the diffusion process with our `Unet1D` model to systematically introduce noise to data:

```
diffusion = GaussianDiffusion1D(model,
                                seq_length = dim_data,
                                timesteps = 100,
                                objective = 'pred_v')
```

This setup enables the Gaussian diffusion model to progressively add noise to an image from an actual distribution over a defined number of time steps, indicated by variable `timesteps`. The `pred_v` objective indicates that the model aims to predict the variance of this noise at each step. In the DDPM paper, the diffusion models followed a linear schedule to manage how noise is added over time. However, in a recent study by Nichol et al. in 2021 [42], it was shown that using a cosine schedule for noise variance can lead to better quality in the generated outputs.

4.2 Training

After setting up the model and the diffusion process, we are ready to train our data. We begin with an unconditional model approach, which involves training on all available sub-grid samples. We first train on the absolute values, and then train on the residuals calculated as the difference between the sub-grid samples and its coarse grid mean. The training process is initiated with the following code snippet:

```
training_seq =  torch.from_numpy(data)

trainer = Trainer1D(diffusion,
                    dataset = training_seq,
                    train_batch_size = batch_size,
                    train_lr = 1e-4,
                    train_num_steps = 100000,
                    gradient_accumulate_every = 2,
                    ema_decay = 0.995,
                    amp = True)
trainer.train()
```

The first line of code converts the dataset from a NumPy array format to a PyTorch tensor, making it suitable for processing. The `Trainer1D` class initialises the training process using the specified diffusion model and training dataset. `train_batch_size = batch_size` refers to the number of samples processed in each training iteration, `train_lr` is the learning rate, `train_num_steps`, is the total number of training steps to perform, `gradient_accumulate_every = 2`, the number of steps over which gradients are accumulated before updating model parameters, `ema_decay = 0.995`, the decay rate for the exponential moving average of the model parameters and `amp = True` enables automatic mixed precision training, which reduces memory usage and potentially speeds up computation without significantly impacting the accuracy of the model. Once trained, we use the following code to generate samples:

```
sampled_seq = diffusion.sample(batch_size = 1000)
```

As mentioned previously, we are interested in using conditional diffusion models that can generate samples based on specific criteria, ideally on a continuous scale. This involves integrating class labels as conditions while also incorporating positional time embeddings into the model. The Continuous Conditional Diffusion Models (CCDMs), as discussed in the paper by Ding et al. [43], are specially designed for this purpose. This is achieved by modifying the U-Net architecture to embed these regression labels directly into each step of the model. Mathematically, this integration involves conditioning both the forward and reverse diffusion processes on the regression labels y. For instance, the forward diffusion is defined by the following equation:

$$q(x_t|x_{t-1}, y) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

The reverse diffusion is also conditioned on y, using the information about the continuous condition to guide the denoising steps effectively:

$$p(x_{t-1}|x_t, y) = \mathcal{N}(x_{t-1}; \mu(x_t, x_0, y), \Sigma(t))$$

In these equations, $\mu(x_t, x_0, y)$ and $\Sigma(t)$ are functions that determine the mean and variance of the reverse diffusion process at each timestep, adapted to the specific regression label y. This ensures that the reverse diffusion is aware of the continuous conditions throughout the process.

However, incorporating Continuous Conditional Diffusion Models (CCDMs) into our current setup, which uses classes like Unet1D and GaussianDiffusion1D, is quite complex. Fully adapting our system to include CCDMs would involve significant changes to both the architecture and the training methods. This would require more research into the specific mathematical processes of CCDMs, ensuring the model accurately handles continuous conditions throughout the diffusion process. It would also be time-consuming and would need careful refinement to get right. Instead, we chose a simpler, more straightforward alternative that still allows us to achieve comparable results. Instead of redesigning our entire model to handle a range of continuous conditions, we split our dataset into smaller groups based on different classes or conditions, which was explored in Chapter 3. We then train separate models on each group. This method simplifies the training process and avoids the need for extensive modifications to our existing model structure. The data was grouped into classes based on the analysis in Chapter 2, which explored the distribution of variables such as land-sea masks, tropospheric temperature inversions, and orography. For example, one configuration divides the data into ten classes based on the presence of temperature inversions and variations in orography:

- Class 1: Sea samples where no temperature inversions are present (12.4 % of the dataset)
- Class 2: Land samples where no temperature inversions are present and Orography $<$240m (2.2 % of the dataset)
- Class 3: Land samples where no temperature inversions are present and Orography $>$240m (1.2 % of the dataset)
- Class 4: Coastal samples where no temperature inversions are present and Orography $<$120m (5.1 % of the dataset)

- Class 5: Coastal samples where no temperature inversions are present and Orography >120m (2.6 % of the dataset)
- Class 6: Sea samples where temperature inversions are present (38.9 % of the dataset)
- Class 7: Land samples where temperature inversions are present and Orography <240m (8.2 % of the dataset)
- Class 8: Land samples where temperature inversions are present and Orography >240m (3.7 % of the dataset)
- Class 9: Coastal samples where temperature inversions are present and Orography <120m (17.1 % of the dataset)
- Class 10: Coastal samples where temperature inversions are present and Orography >120m (8.5 % of the dataset)

In another configuration, the classes were organised using the standard deviation of orography:

- Class 1: Sea samples where no temperature inversions are present (12.4 % of the dataset)
- Class 2: Land samples where no temperature inversions are present and Standard Deviation of Orography <100m (1.9 % of the dataset)
- Class 3: Land samples where no temperature inversions are present and Standard Deviation of Orography >100m (1.5 % of the dataset)
- Class 4: Coastal samples where no temperature inversions are present and Standard Deviation of Orography <120m (5.4 % of the dataset)
- Class 5: Coastal samples where no temperature inversions are present and Standard Deviation of Orography >120m (2.4 % of the dataset)
- Class 6: Sea samples where temperature inversions are present (38.9 % of the dataset)
- Class 7: Land samples where temperature inversions are present and Standard Deviation of Orography <100m (6.9 % of the dataset)
- Class 8: Land samples where temperature inversions are present and Standard Deviation of Orography >100m (5.0 % of the dataset)
- Class 9: Coastal samples where temperature inversions are present and Standard Deviation of Orography <120m (17.9 % of the dataset)
- Class 10: Coastal samples where temperature inversions are present and Standard Deviation of Orography >120m (7.7 % of the dataset)

By splitting the data this way, we could better account for the most influential factors that shape sub-grid profiles. We can directly incorporate these contextual variables into the learning process, enabling the model to produce more accurate and realistic outputs tailored to specific conditions. Rather than training a single model to try handle a wide range of atmospheric scenarios, conditional models would allow us to simplify the training process by handling distinct subsets of data. We aim to improve the representation of sub-grid processes in its vertical structure by training the data in this way.

4.3 Computational Requirements

When training complex models like ours, using a GPU is crucial due to the heavy computational demands. GPUs (Graphics Processing Units) are necessary because they are optimised for parallel processing, which is ideal for the matrix operations and large-scale data handling required in neural network training. This allows for faster computations compared to using a CPU (Central Processing Unit), reducing training time from days to hours or even minutes. Initial attempts to train on a laptop without a GPU were unsuccessful due to long processing times and system crashes, even for a minimal number of training steps. The laptop was equipped with an 11th Gen Intel® Core™ i5-1135G7 CPU running at 2.40GHz and 8GB of RAM. To address these limitations, we used Google Collab, which offers free GPU credits. This platform allowed for training but was still not ideal as the available credits would often run out. Ultimately, we used the Maths4DL machine hosted on the university server. This machine significantly improved the training process, where the specifications of the Maths4DL workstation include a GPU with an RTX 4090 (24GB VRAM), a CPU with a Ryzen 9 7950x3d, and 96GB of RAM operating at 5200MHz. Using the university's VPN, we accessed the server via SSH, and specific GPU resources could be monitored using the "nvidia-smi" command. The metrics revealed that the GPU usually operated at 68% utilisation during training sessions. This indicated that while the GPU was being effectively used, there was still potential for higher usage without risking overloads. On the Maths4DL machine, training for 1000 steps took about 1 minute, 10,000 steps about 9 minutes, and 100,000 steps approximately 90 minutes. Using the Maths4DL machine significantly reduced training time and improved efficiency. Although we successfully trained on all samples in our dataset over an extended number of steps, we encountered a 'CUDA Out of Memory Error' when attempting to generate more than 10,000 samples.

5. Results

5.1 Unconditional Model

The more commonly studied, discriminative models are easier to evaluate academically, as their performance can be measured against clear benchmarks like classification accuracy. In contrast, generative models are harder to assess due to the subjective nature of output quality, making it challenging to determine the best approach. We first assess our model performance by training on absolute samples across 1,000, 10,000 and 100,000 steps. The full set of results can be found in Appendix A.1. The profiles of 100 random generated sample outputs and targets are plotted in Figure 5.1. Impressively, even after only 1,000 steps, a process that takes just one minute, the model was able to capture the general shape of the atmospheric profiles, although the outputs initially displayed a high degree of variance and jaggedness. However, as training progresses, there is notable convergence of the sample lines toward the target profiles, as shown by the plots in the middle row. While the visual data from the plots shows a clear improvement in how closely the model's outputs align with the target profiles as training progresses, it is difficult to assess quantitatively.

A better way of evaluating would be plotting the mean profiles of our samples against their respective target means, each represented with their standard deviations at each vertical level. This is shown in Figure 5.2, where a clearer comparison can be observed between the generated sample and target profiles. As expected, closer alignment can be seen from the model's outputs at 100,000 training steps. However, while the sample means are approaching the target means, the plots reveal that the deviations of the fine grid samples from the coarse grid means remain significant, especially in lower levels. Specifically in the pressure profiles, it remains to be difficult to assess whether our model has actually learned the full variation of the sub-grid scale processes or if it is biased towards the coarse grid mean, since the error bars are too small to allow for a meaningful comparison.

As previously discussed, we shifted our focus to training the model on residuals - the differences between each fine-grid sample and the coarse-grid mean. This approach allows the model to focus on the finer-scale variations within the sub-grid profiles rather than simply fitting to the broader trends represented by the coarse grid. This adjustment better aligns with the project's objective of capturing the sub-grid scale variability in atmospheric profiles. Figures 5.3, 5.4, and 5.5 display the generated and target residuals for humidity, pressure, and temperature after 1,000, 10,000, and 100,000 training steps, respectively. These plots provide a clearer representation of the vertical variability at each level and highlight the model's ability to learn the finer details of sub-grid processes in

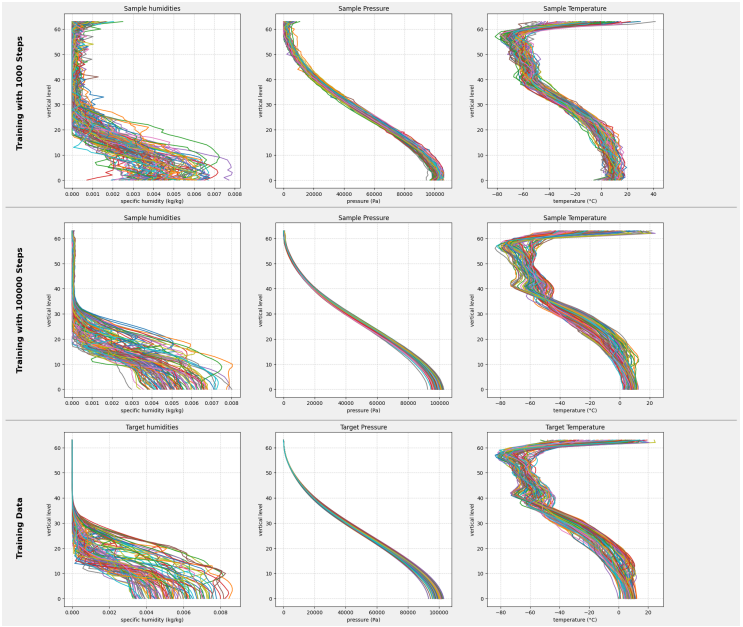

Figure 5.1: From left to right: humidity, pressure, temperature profiles across 64 levels. Top row: 100 generated samples from training over 1,000 steps. Middle row: 100 generated samples from training over 100,000 steps. Bottom row: 100 random samples from our training data.

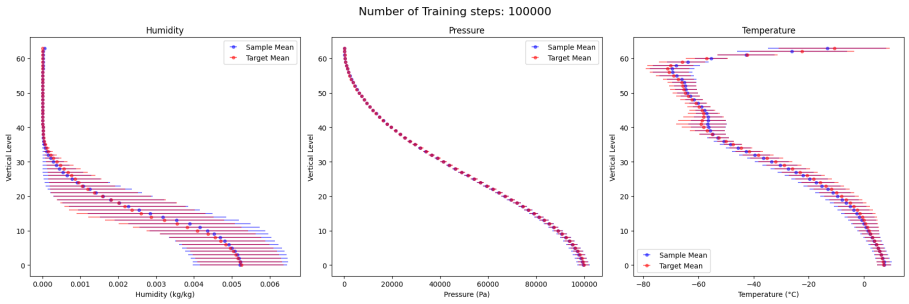

Figure 5.2: Mean and Standard Deviations of the vertical profiles for humidity, pressure, and temperature at 100,000 training steps.

its vertical structure over an increasing number of training steps.

Table 5.1: MSE for Humidity, Pressure, and Temperature across Different Training Steps

	1000 Steps		10000 Steps		100000 Steps	
Variable	MSE (Mean)	MSE (Std Dev)	MSE (Mean)	MSE (Std Dev)	MSE (Mean)	MSE (Std Dev)
Humidity (kg/kg)	1.54×10^{-10}	5.27×10^{-9}	1.28×10^{-10}	5.74×10^{-10}	2.11×10^{-10}	8.78×10^{-10}
Pressure (Pa)	23500	68700	7770	17400	745	500
Temperature (°C)	0.0281	0.119	0.004	0.084	0.00209	0.0419

In Table 5.1, we compare the total mean squared error (MSE) across all levels between the target means and generated sample means, as well as the MSE between the target and generated standard deviations. The MSE between the means provides a quantitative measure of how close the generated residuals are to zero. At the early stage of 1,000

training steps (Figure 5.3), the model struggles to learn the vertical structure of the residuals. The MSE values in Table 5.1 confirm this, showing relatively high errors across all variables. For example, the MSE for pressure is 23,500 (mean) and 68,700 (standard deviation), indicating significant deviations from the target data. Similarly, the temperature profile shows an MSE of 0.0281 (mean) and 0.119 (standard deviation), demonstrating the model's limited ability.

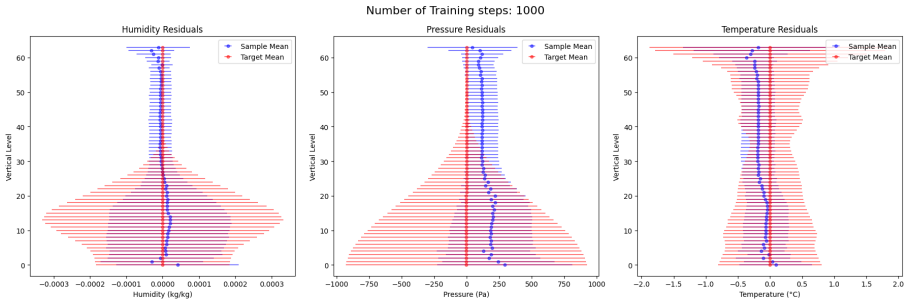

Figure 5.3: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 1,000 training steps.

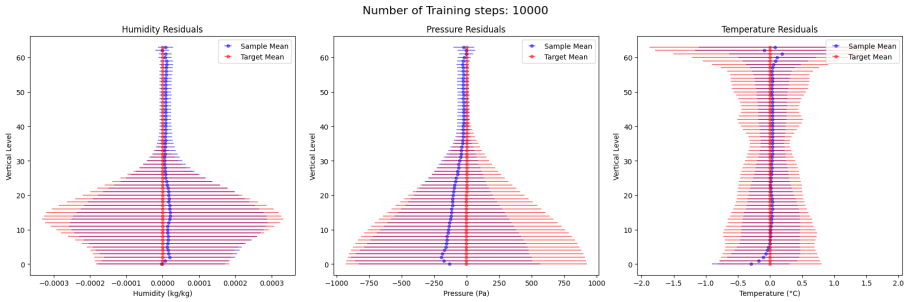

Figure 5.4: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps.

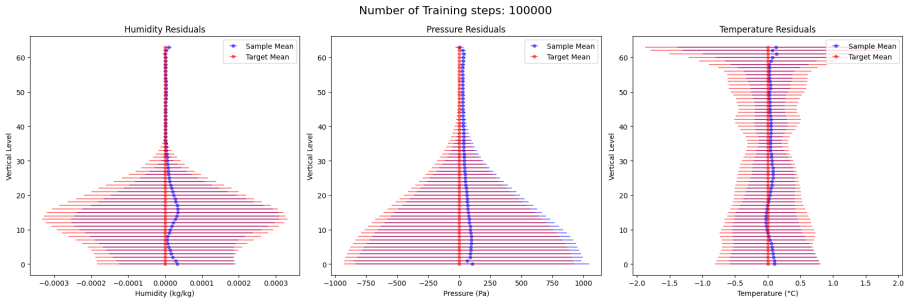

Figure 5.5: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps.

Visually, the plots reveal that the model fails to differentiate the finer details in the vertical profiles, especially for humidity and pressure above vertical levels 30-40, where the generated variation is greater than the target residuals. The pressure and temperature residuals show a lack of alignment with the target means, indicating that the model is

not even capturing the general trend, let alone the sub-grid variations effectively at this stage. At 10,000 training steps (Figure 5.4), the model shows notable improvements in its ability to capture the sub-grid variability. The MSE values in Table 5.1 illustrate this progress, with the MSE for all three variables decreasing by approximately a factor of 10. In the plots, we can see that the model is starting to capture more variation across vertical levels, as the shape more closely resembles the target variations. The pressure and temperature profiles now show clearer alignment between the generated and target residuals, particularly in the mid-levels, where sub-grid processes tend to have more significant variation. However, some discrepancies still remain, particularly in the the pressure and temperature profiles, where the model struggles to fully represent the full variability. Overall, while the model has improved, it still requires further training to better represent the complete variations in the data. By 100,000 training steps (Figure 5.5), the model shows significant improvement, as reflected in both the visual plots and MSE values. For pressure, the MSE drops dramatically to 745 (mean) and 500 (standard deviation), indicating that the model has learned the structure of the pressure residuals quite well. Similarly, for temperature, the MSE decreases to 0.00209 (mean) and 0.0419 (standard deviation). Humidity and temperature, while improved, still shows some deviations in the mid to upper levels. For humidity, the MSE in standard deviation remains relatively high compared to the other variables (MSE increases slightly to 8.78×10^{-10}). This could indicate that while the model is performing well overall, certain aspects of the humidity and temperature profiles are more challenging to capture accurately.

One possible reason the model, even after 100,000 training steps, fails to achieve a sample mean of 0 could be due to the statistical limitations arising from the relatively small number of samples generated. Although the training dataset consists of 5,184,000 points, only 10,000 samples were generated for evaluation, which may not be fully representative of the entire data distribution. This could introduce bias, preventing the model from accurately capturing the full variability and achieving a mean of 0. To test this theory, we ran the model multiple times with 100,000 training steps, each time generating 10,000 samples. While the distributions varied slightly between runs, they all showed similar discrepancies when compared to the target data. To further investigate whether the issue was related to the number of generated samples, we compared the results of generating a smaller number of samples from the same model weights, but the distribution did not change significantly when less samples were generated. This suggests that the discrepancy may not purely be a result of statistical sampling error but could also point to a potential modelling limitation.

5.2 Conditional Model

We assess the performance of our conditional model using the same methods. Detailed results are available in Appendix A.1. The conditional model, configured to split data based on land-sea mask, presence of temperature inversions, and orography, was analysed after 100,000 training steps. Initial observations from our plots indicate that this model captures data variations effectively, similar to the unconditional model. This observation is reflected by the data in Table 5.2, where the MSE for both mean and standard deviation across all classes generally aligns with those observed in the unconditional model. Specifically, the MSE for mean humidity was lower in all classes except Class 8, while Classes 1, 2, 5, and 6 reported lower MSE for mean pressure. Only Class 6 showed a lower MSE for mean temperature. Notably, in sea samples (Classes 1 and 6),

the variations were minimal, and the general trends were well captured, evidenced by the lowest MSE in mean values. Figures 5.6 and 5.7 display the model outputs for Class 1 (sea samples without temperature inversions) and Class 6 (sea samples with temperature inversions), respectively.

Table 5.2: MSE for Humidity, Pressure, and Temperature across Classes

Class	**Humidity (kg/kg)**		**Pressure (Pa)**		**Temperature** (°C)	
	MSE Mean	**MSE Std Dev**	**MSE Mean**	**MSE Std Dev**	**MSE Mean**	**MSE Std Dev**
Class 1	2.62×10^{-11}	6.87×10^{-10}	210	694	0.0037	0.0215
Class 2	3.20×10^{-11}	3.36×10^{-10}	418	2650	0.0043	0.014
Class 3	4.92×10^{-11}	1.42×10^{-9}	6990	8065	0.00638	0.0253
Class 4	1.90×10^{-10}	1.65×10^{-9}	19070	36900	0.0193	0.10095
Class 5	8.23×10^{-11}	1.37×10^{-9}	587	5440	0.00226	0.0837
Class 6	8.23×10^{-12}	2.22×10^{-9}	486	4430	0.000286	0.0517
Class 7	1.26×10^{-10}	3.80×10^{-9}	3560	24500	0.00447	0.103
Class 8	6.69×10^{-10}	1.97×10^{-9}	2930	8400	0.0102	0.0393
Class 9	1.036×10^{-10}	1.92×10^{-9}	2032	2690	0.00838	0.0396
Class 10	6.17×10^{-11}	4.086×10^{-9}	54700	1840	0.0214	0.0911

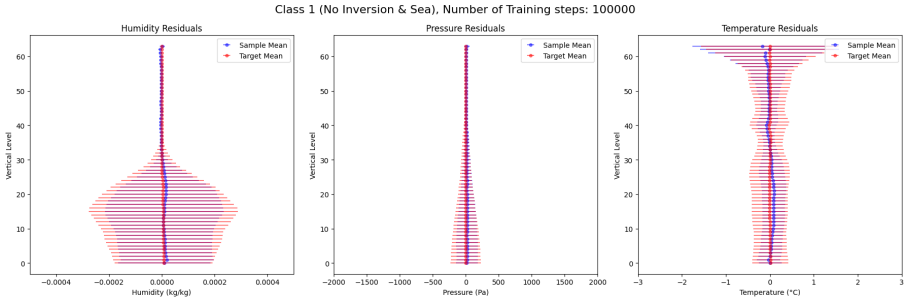

Figure 5.6: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 1 (Sea samples where no temperature inversions are present)

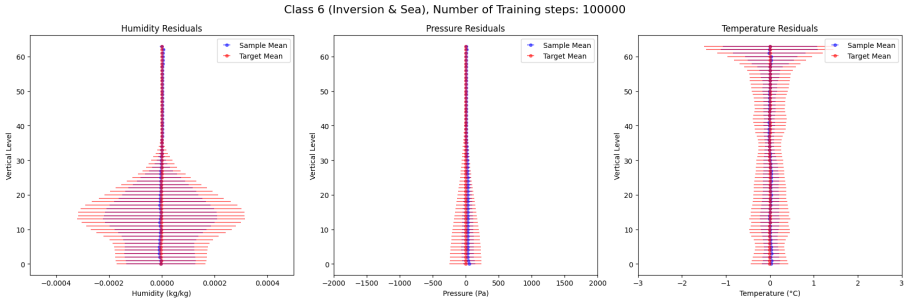

Figure 5.7: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 6 (Sea samples where temperature inversions are present)

In examining the land classes, we observe broader variations in pressure and temperature compared to those in sea classes. Specifically, Class 7 (Land samples where temperature inversions are present and Orography <240m) and Class 8 (Land samples where

temperature inversions are present and Orography >240m), as depicted in Figures 5.8 and 5.9, display notable differences. It is apparent that the pressure and temperature variations are more pronounced in samples with higher orography. Although the model successfully captures the full range of variability in Class 8, it faces challenges in Class 7, where the variations are more constrained. This pattern also appears in coastal samples, indicating that the model struggles more with representing the vertical structure in environments with less variation. This is shown in Figures 5.10 and 5.11, showing the model outputs from Class 4 (Coastal samples where no temperature inversions are present and Orography <120m) and 5 (Coastal samples where no temperature inversions are present and Orography >120m) respectively. Across nearly all classes, the MSE for standard deviation exceeds that of the unconditional model, highlighting the model's difficulty in fully learning the vertical variations when data is segmented into distinct classes.

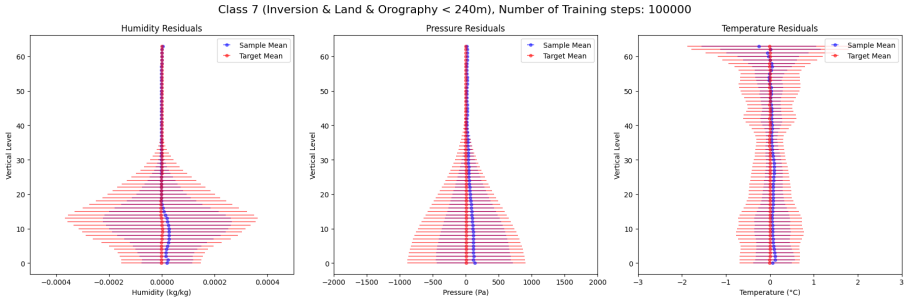

Figure 5.8: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 7 (Land samples where temperature inversions are present and Orography <240m)

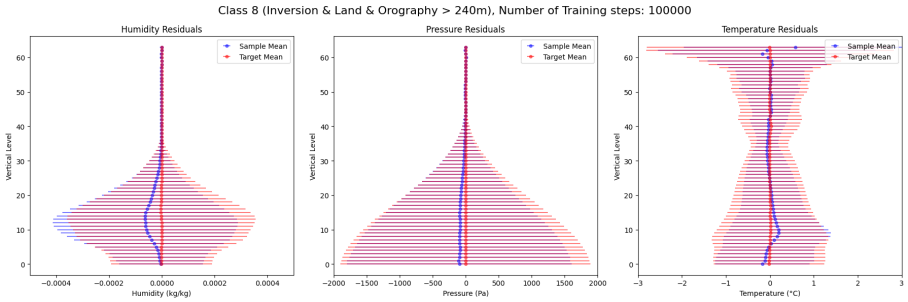

Figure 5.9: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 8 (Land samples where temperature inversions are present and Orography >240m)

The MSE for the mean across all profiles is comparable or generally lower, indicating that while the conditional model has difficulties capturing variability, it might be more effective at identifying the overall trend. The discrepancy in performance between the conditional and unconditional models may be attributed to the restricted dataset size each conditional class is trained on, sometimes as little as 2% of the entire dataset, amounting to around 100,000 data points compared to 5 million in the full dataset. This limited data may not adequately represent the full variability needed to train models to generalise well across different scenarios. From the analysis of each class, it's evident that the target distributions across the sub-grid profiles vary significantly. As orography increases,

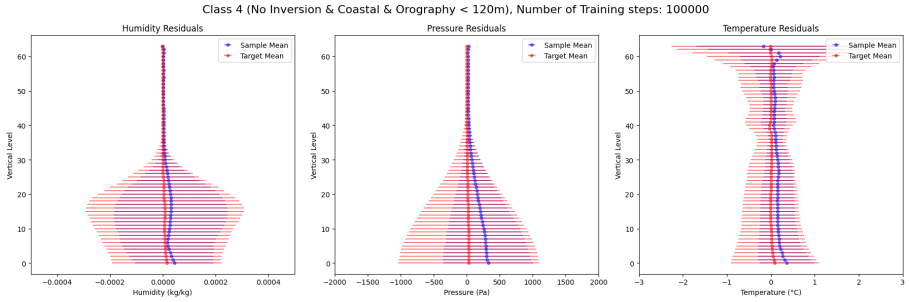

Figure 5.10: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 4 (Coastal samples where no temperature inversions are present and Orography <120m)

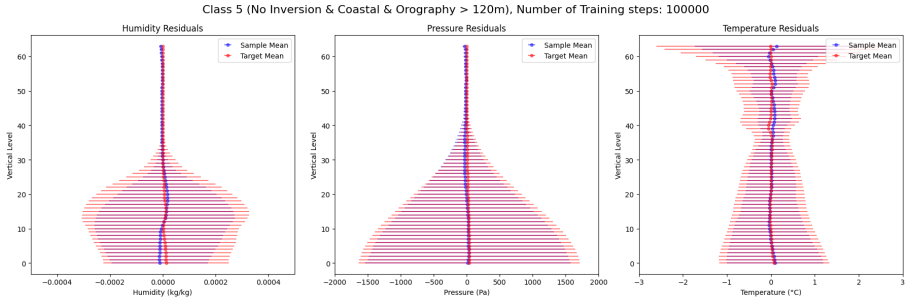

Figure 5.11: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 5 (Coastal samples where no temperature inversions are present and Orography >120m)

we observe a corresponding increase in the range of pressure and temperature values. The first five classes, which have no temperature inversions, display a distinct humidity distribution that appears pentagonal, whereas the latter five classes show a diamond-like shape. Ideally, our model should be able to learn these unique distributions, as they significantly influence the variability in the vertical structure of our sub-grid processes. While the unconditional model performs adequately by providing a general overview, enhancing the performance of our conditional model would be more advantageous. We anticipate that additional subdivisions of the data could reveal even more distinct variations in profiles. However, processing a single configuration of class splits currently takes about 15 hours, with each model requiring approximately 90 minutes to run. Moreover, further splitting the data would result in smaller datasets for each class, which is not ideal as it could limit the model's performance as shown from the results.

6. Discussion & Conclusion

The goal of our project was to learn the variations within sub-grid scale processes in their vertical structure, and as our results show, the model does a reasonably good job of this. As theorised in Section 3.2, a generative model is deemed effective if it can generate examples that convincingly appear as though they were drawn from the data while still producing sufficiently distinct outputs, rather than merely replicating the original observations. Our findings are promising, showing that our model can indeed learn the variations observed in historical sub-grid profiles and generate new samples that adhere to this learned structure. With an increasing number of training steps, there was a notable improvement in accuracy, suggesting that the model does not only capture the general trends but also effectively replicates the specific variations within the vertical structure. Training the model conditionally has also proven beneficial, allowing us to adjust the model's learning process to the diverse distributions observed across different profiles, and yielding results comparable to those of the unconditional model. However, one limitation of our project is determining the best evaluation metric for assessing the model's performance. Do we want our generated samples to exactly follow the full distribution of the data, or is some margin of error acceptable, and if so, to what extent? Ultimately, the true effectiveness of the model may not be fully realised until it is integrated with existing parameterisation schemes and evaluated in real-world industry applications. This step is vital to test whether our approach can enhance the existing schemes, which typically assume a uniform distribution of uncertainty across vertical levels. Our hope is that by addressing these vertical variations more accurately, our method will lead to improvements in the prediction accuracy and reliability of atmospheric models.

A significant computational limitation in this project was the challenge associated with generating a large number of samples, which restricted our evaluation to only 10,000 samples out of a training dataset comprising over 5 million data points. This could introduce a sampling bias, preventing the model from accurately representing the entire distribution of generated data. The restricted number of generated samples could skew the evaluation results, leading to a model that might overfit to the smaller sample set rather than effectively learning from the broader characteristics of the data. This issue could be further compounded by the stochastic nature of the diffusion model, where different runs yield slightly varying results. The computational demands of the project also presented other limitations, particularly in terms of the time required to achieve desirable results. Each run of the model took approximately 90 minutes, which limited our ability to experiment with a diverse range of class configurations. This extended processing time could be impractical for real-time applications. This limitation highlights the need for further optimisation of the model's computational efficiency or the exploration of more advanced computational strategies to reduce processing times. This would make

the model more suitable for real-world applications, where time is a critical factor.

Given the time constraints of this project, significant effort was spent identifying the most suitable model, which limited the opportunity to thoroughly explore and optimise the model's hyperparameters and configurations. These constraints also influenced the structural choices in the model's architecture. Specifically, the U-Net architecture used in this project could only train on data vectors of dimensions that are powers of two (2^n). This limitation was generally manageable for humidity and pressure, where variations in levels above 64 were minimal. However, this imposed a significant challenge for modelling temperature, as the most critical variations were observed between levels 64 and 70. To address this, interpolation methods were initially used to truncate the temperature profiles to 64 levels. Unfortunately, this approach did not yield a smooth representation, impacting the model's ability to accurately simulate temperature at upper levels. Future work should focus on adapting the model structure to accommodate the full range of observed temperature variations. Also, majority of existing research on diffusion models primarily focuses on image data, with extensive applications in generating and manipulating 2D visual content. This focus on images means that most of the established models and optimisations are tailored for handling two-dimensional data arrays rather than the one-dimensional data. Although it works in the same way, it is unknown whether the diffusion model, as currently configured, is truly the best fit for our needs or if there might be a more effective model structure that is better suited to the nuances of atmospheric data.

One of the most significant limitations encountered in our project was related to the size of the dataset available for training. Originally, the dataset encompassed approximately 5TB of data, covering 80 different regions over a month. This offered a diverse representation of global climates by including observations from various geographic locations around the world. However, due to constraints on computational resources, our study was confined to analysing only a single region for a period of 15 days with hourly measurements. This limitation drastically reduced the variety of the data we could use, thereby limiting the scope of our analysis to a very narrow segment of the world's atmospheric conditions. This larger dataset would have still only covered 2% of the global surface area. This compromises the model's ability to generalise to other regions and conditions. Important climate patterns and variations that occur in other parts of the world or at different times may not be represented in the training data, leading to a model that performs well on specific local data but lacks broader applicability. The training dataset's limited geographic and temporal scope might also introduce biases in the model. For example, if the region under study has particular patterns that are not common elsewhere, the model might learn these as more prevalent or normal than they actually are. To mitigate this limitation in future work, it would be advantageous to expand the dataset to include a wider array of regions and longer time periods. Achieving reasonable results in this project with a limited subset of data provides a strong proof of concept, suggesting that increasing the dataset size will likely enhance the model's performance. The approach of manually segmenting data into distinct classes based on specific atmospheric conditions like orography and temperature inversions also introduces limitations. This does not adequately reflect the nature of atmospheric conditions, which are not naturally discrete but rather exist on a continuous spectrum. Future work should focus on implementing these variables as continuous conditions, so that the model could learn to adapt its behaviour based on a scale of input values rather than switching between rigidly de-

fined categories. Again, for this to work appropriately a much larger dataset is necessary.

Despite the limitations discussed, this project has successfully demonstrated the application of generative machine learning to atmospheric modelling, specifically in addressing the uncertainties associated with sub-grid processes in their vertical structure. This opens avenues for incorporating additional variables in future studies. Generative machine learning is particularly advantageous in this context as it captures stochasticity, improving the model's generalisation capabilities. Moreover, as research in generative machine learning is relatively new and continually expanding, its growing relevance across various societal sectors suggests that it will likely play a significant role in organisations like the Met Office too, particularly within their current data science framework [44].

Bibliography

[1] Wood, N., Staniforth, A., White, A., Allen, T., Diamantakis, M., Gross, M., Melvin, T., Smith, C., Vosper, S., Zerroukat, M. and Thuburn, J., 2014. An inherently mass-conserving semi-implicit semi-Lagrangian discretization of the deep-atmosphere global non-hydrostatic equations. Quarterly Journal of the Royal Meteorological Society, 140(682), pp.1505-1520.

[2] Melvin, T., Benacchio, T., Shipway, B., Wood, N., Thuburn, J. and Cotter, C., 2019. A mixed finite-element, finite-volume, semi-implicit discretization for atmospheric dynamics: Cartesian geometry. Quarterly Journal of the Royal Meteorological Society, 145(724), pp.2835-2853.

[3] Sanchez, C., Williams, K.D., Shutts, G. and Collins, M., 2014. Impact of a Stochastic Kinetic Energy Backscatter scheme across time-scales and resolutions. Quarterly Journal of the Royal Meteorological Society, 140(685), pp.2625-2637.

[4] Palmer, T.N., Buizza, R., Doblas-Reyes, F., Jung, T., Leutbecher, M., Shutts, G.J., Steinheimer, M. and Weisheimer, A., 2009. Stochastic parametrization and model uncertainty.

[5] Ollinaho, P., Lock, S.J., Leutbecher, M., Bechtold, P., Beljaars, A., Bozzo, A., Forbes, R.M., Haiden, T., Hogan, R.J. and Sandu, I., 2017. Towards process-level representation of model uncertainties: stochastically perturbed parametrizations in the ECMWF ensemble. Quarterly Journal of the Royal Meteorological Society, 143(702), pp.408-422.

[6] Gentine, P., Pritchard, M., Rasp, S., Reinaudi, G. and Yacalis, G., 2018. Could machine learning break the convection parameterization deadlock?. Geophysical Research Letters, 45(11), pp.5742-5751.

[7] Yuval, J. and O'Gorman, P.A., 2020. Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions. Nature communications, 11(1), p.3295.

[8] Krasnopolsky, V.M., Fox-Rabinovitz, M.S. and Belochitski, A.A., 2013. Using ensemble of neural networks to learn stochastic convection parameterizations for climate and numerical weather prediction models from data simulated by a cloud resolving model. Advances in Artificial Neural Systems, 2013(1), p.485913.

[9] Gagne, D.J., Christensen, H.M., Subramanian, A.C. and Monahan, A.H., 2020. Machine learning for stochastic parameterization: Generative adversarial networks in the Lorenz'96 model. Journal of Advances in Modeling Earth Systems, 12(3), p.e2019MS001896.

[10] Guillaumin, A.P. and Zanna, L., 2021. Stochastic-deep learning parameterization of ocean momentum forcing. Journal of Advances in Modeling Earth Systems, 13(9), p.e2021MS002534.

[11] Towards AI, 2023. GAN Mode Collapse Explanation [Online]. Available from: https://pub.towardsai.net/gan-mode-collapse-explanation-fa5f9124ee73 [Accessed 20 August 2024]

[12] Nvidia, 2022. Improving Diffusion Models as an Alternative To GANs, Part 1 [Online]. Available from: https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/ [Accessed 20 August 2024]

[13] Bubba, T., 2023. *MA50290: Machine Learning 2.* University of Bath. Unpublished.

[14] Rout, Siddharth & Dwivedi, Vikas & Srinivasan, Balaji. (2019). Numerical Approximation in CFD Problems Using Physics Informed Machine Learning. 10.48550/arXiv.2111.02987.

[15] Study Machine Learning, 2021. Mathematics behind the Neural Network [Online]. Available from: https://studymachinelearning.com/mathematics-behind-the-neural-network/ [Accessed 20 August 2024]

[16] Medium 2023. Activation Functions and Loss Functions in Deep Learning: Differences and Overlaps [Online]. Available from: https://python.plainenglish.io/activation-functions-and-loss-functions-in-deep-learning-differences-and-overlaps-e62249de999d [Accessed 20 August 2024]

[17] Microsoft 2024, What is Convolutional Neural Network — CNN (Deep Learning)[Online]. Available from: https://techcommunity.microsoft.com/t5/ai-machine-learning/what-is-convolutional-neural-network-cnn-deep-learning/m-p/4184725 [Accessed 20 August 2024]

[18] Towards Data Science 2022, What Are Transposed Convolutions? [Online]. Available from: https://towardsdatascience.com/what-are-transposed-convolutions-2d43ac1a0771 [Accessed 20 August 2024]

[19] Aggarwal, C.C., 2018. Neural Networks and Deep Learning.

[20] Santry, D.J., 2023. Demystifying Deep Learning: An Introduction to the Mathematics of Neural Networks [Online]. Wiley-Blackwell. Available from: https://doi.org/10.1002/9781394205639.

[21] Bengio, Y., Goodfellow, I. and Courville, A., 2017. Deep learning (Vol. 1). Cambridge, MA, USA: MIT press.

[22] M.P. Deisenroth, A.A. Faisal and C.S. Ong, Mathematics for Machine Learning, 2020

[23] Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18 (pp. 234-241). Springer International Publishing.

[24] Tomczak, J.M., 2022. Deep generative modeling. Cham, Switzerland: Springer.

[25] Foster, D., 2022. Generative deep learning. " O'Reilly Media, Inc.".

[26] Yang Song, 2021. Generative Modeling by Estimating Gradients of the Data Distribution [Online]. Available from: https://yang-song.net/blog/2021/score/ [Accessed 20 August 2024]

[27] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N. and Ganguli, S., 2015, June. Deep unsupervised learning using nonequilibrium thermodynamics. In International conference on machine learning (pp. 2256-2265). PMLR.

[28] Ho, J., Jain, A. and Abbeel, P., 2020. Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, pp.6840-6851.

[29] Hugging Face, 2022. The Annotated Diffusion Model [Online]. Available from: https://huggingface.co/blog/annotated-diffusion [Accessed 17 August 2024]

[30] Lil'Log, 2021. What are Diffusion Models? [Online]. Available from: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/ [Accessed 19 August 2024]

[31] Medium, 2023. A friendly Introduction to Denoising Diffusion Probabilistic Models [Online]. Available from: https://medium.com/@gitau_am/afriendlyintroductiontodenoisingdiffusionprobabilisticmodelscc76b8abef25 [Accessed 19 August 2024]

[32] Webster, S., Uddstrom, M., Oliver, H. and Vosper, S., 2008. A high-resolution modelling case study of a severe weather event over New Zealand. Atmospheric Science Letters, 9(3), pp.119-128.

[33] Walters, D., Boutle, I., Brooks, M., Melvin, T., Stratton, R., Vosper, S., Wells, H., Williams, K., Wood, N., Allen, T. and Bushell, A., 2017. The Met Office unified model global atmosphere 6.0/6.1 and JULES global land 6.0/6.1 configurations. Geoscientific Model Development, 10(4), pp.1487-1520.

[34] Bush, M., Boutle, I., Edwards, J., Finnenkoetter, A., Franklin, C., Hanley, K., Jayakumar, A., Lewis, H., Lock, A., Mittermaier, M. and Mohandas, S., 2023. The second Met Office Unified Model–JULES regional atmosphere and land configuration, RAL2. Geoscientific Model Development, 16(6), pp.1713-1734.

[35] Donlon, C.J., Martin, M., Stark, J., Roberts-Jones, J., Fiedler, E. and Wimmer, W., 2012. The operational sea surface temperature and sea ice analysis (OSTIA) system. Remote Sensing of Environment, 116, pp.140-158.

[36] American Meteorological, 2024. Glossary of Meteorogology [Online]. Available from: https://glossary.ametsoc.org/wiki/Welcome [Accessed 12 August 2024]

[37] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[38] Vaswani, A., 2017. Attention is all you need. Advances in Neural Information Processing Systems.

[39] Zagoruyko, S., 2016. Wide residual networks. arXiv preprint arXiv:1605.07146.

[40] Shen, Z., Zhang, M., Zhao, H., Yi, S. and Li, H., 2021. Efficient attention: Attention with linear complexities. In Proceedings of the IEEE/CVF winter conference on applications of computer vision (pp. 3531-3539).

[41] Wu, Y. and He, K., 2018. Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).

[42] Nichol, A.Q. and Dhariwal, P., 2021, July. Improved denoising diffusion probabilistic models. In International conference on machine learning (pp. 8162-8171). PMLR.

[43] Ding, X., Wang, Y., Zhang, K. and Wang, Z.J., 2024. CCDM: Continuous Conditional Diffusion Models for Image Generation. arXiv preprint arXiv:2405.03546.

[44] Met Office, 2022. Embedding machine learning and artificial intelligence in weather and climate science and services [Online]. Available from: https://www.metoffice.gov.uk/binaries/content/assets/metofficegovuk/pdf/research/foundation-science/data-science-framework-2022-2027.pdf [Accessed 08 September 2024]

A. Results

A.1 Unconditional Model

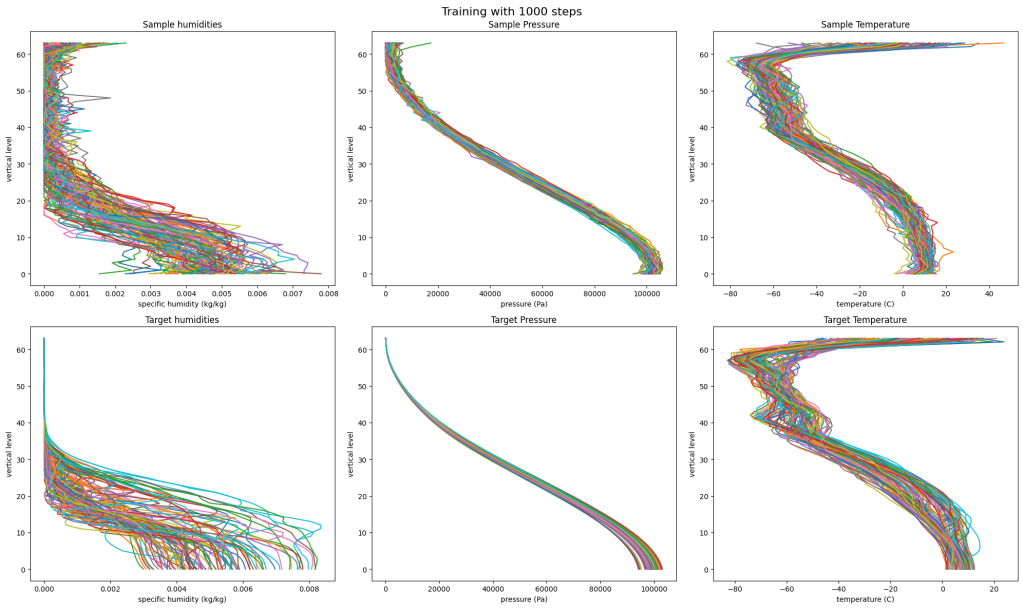

Figure A.1: Top row: 100 generated samples from training over 1,000 steps, of humidity, pressure and temperature profiles across 64 levels. Bottom row: 100 random samples from our training data.

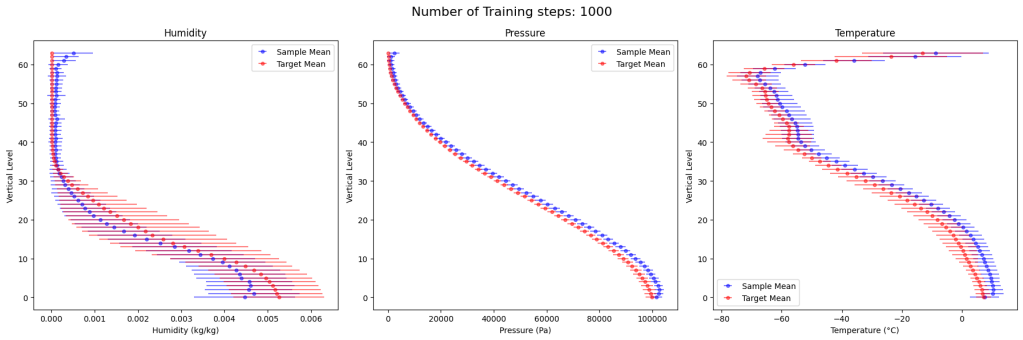

Figure A.2: Mean and Standard Deviations of the vertical profiles for humidity, pressure, and temperature at 1,000 training steps.

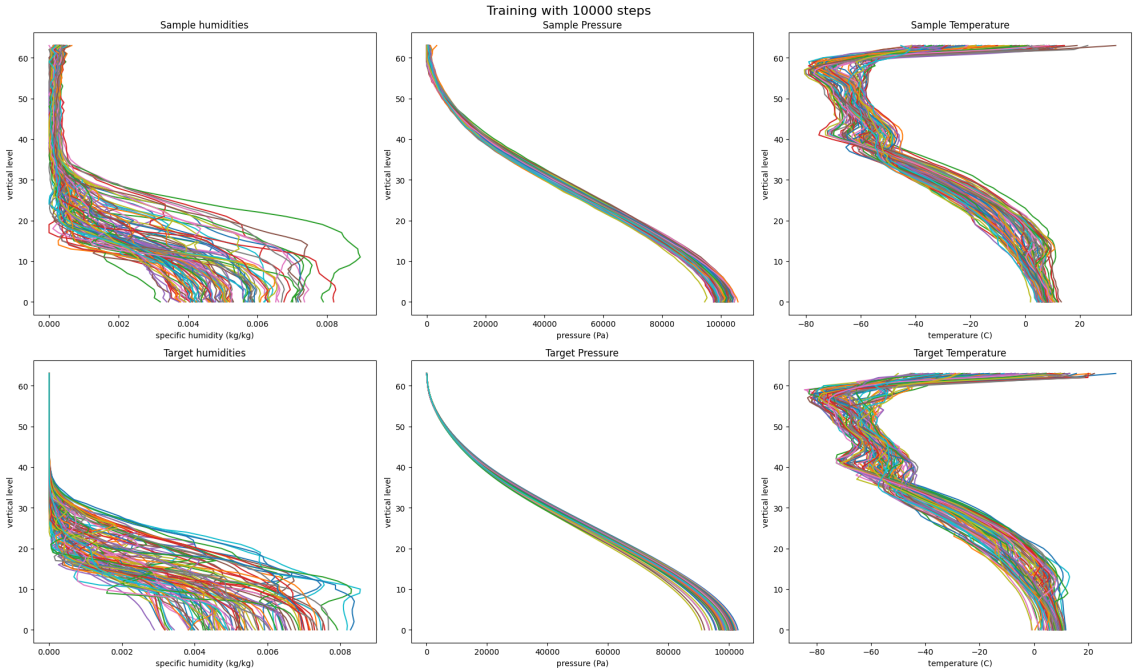

Figure A.3: Top row: 100 generated samples from training over 10,000 steps, of humidity, pressure and temperature profiles across 64 levels. Bottom row: 100 random samples from our training data.

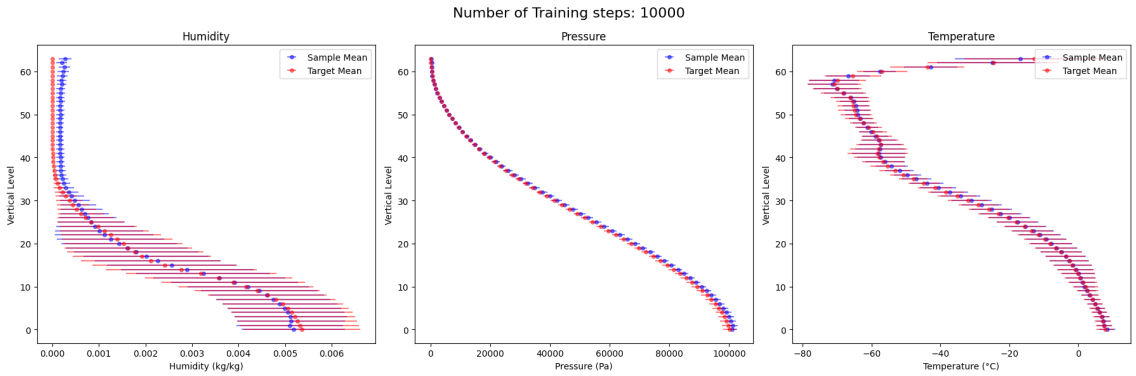

Figure A.4: Mean and Standard Deviations of the vertical profiles for humidity, pressure, and temperature at 10,000 training steps.

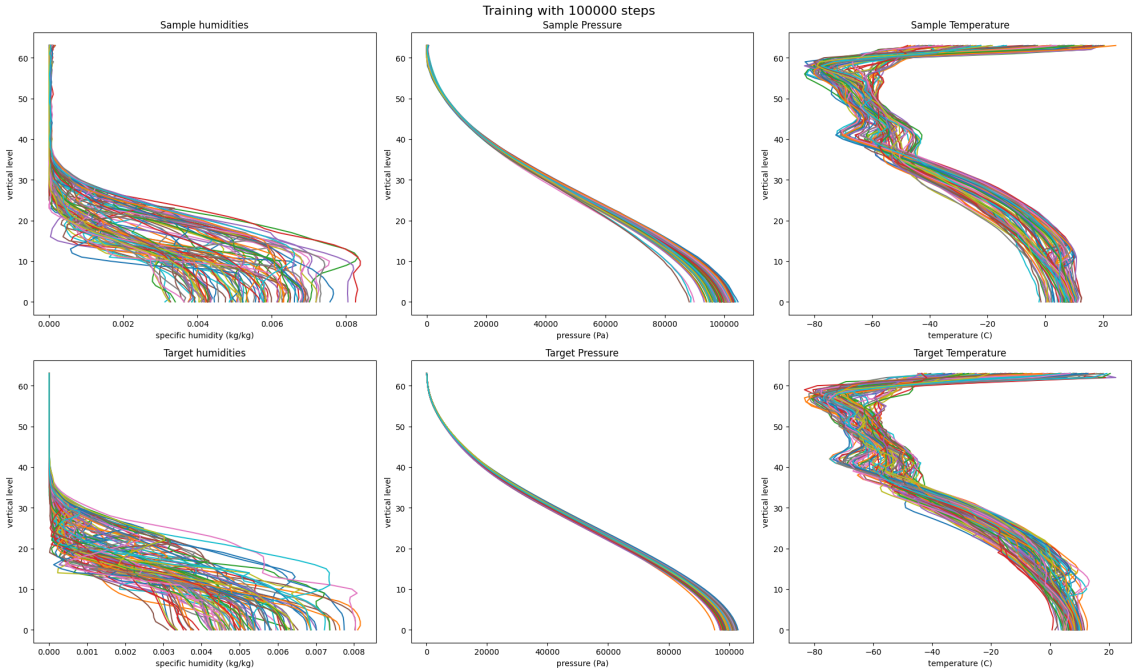

Figure A.5: Top row: 100 generated samples from training over 10,000 steps, of humidity, pressure and temperature profiles across 64 levels. Bottom row: 100 random samples from our training data.

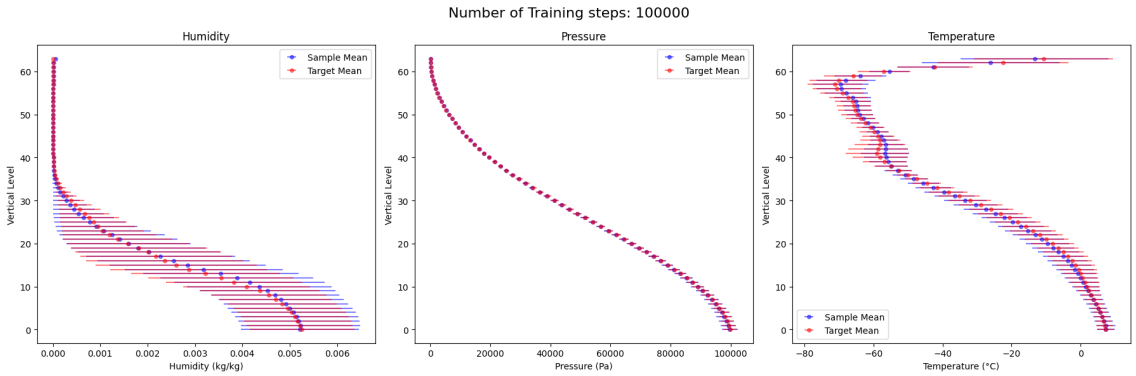

Figure A.6: Mean and Standard Deviations of the vertical profiles for humidity, pressure, and temperature at 100,000 training steps.

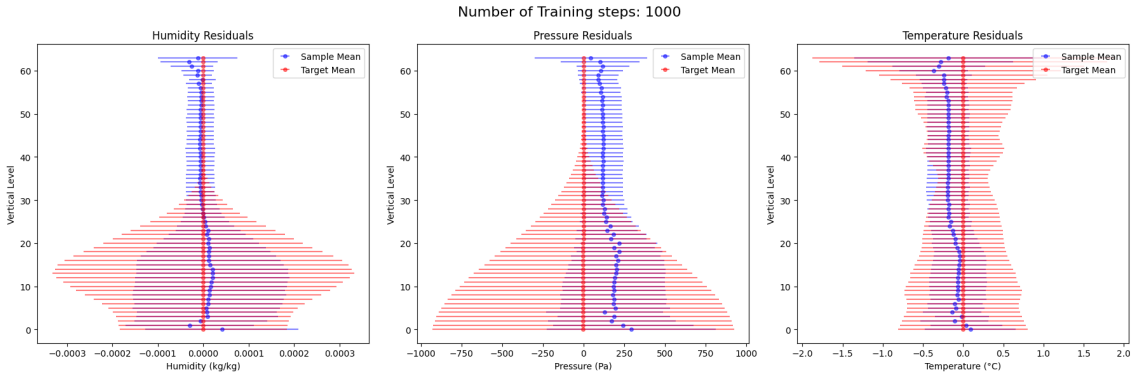

Figure A.7: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 1,000 training steps.

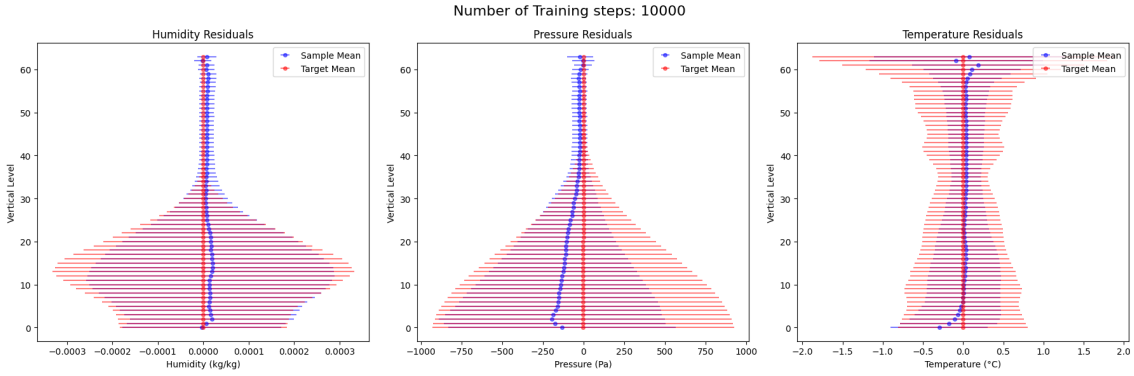

Figure A.8: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps.

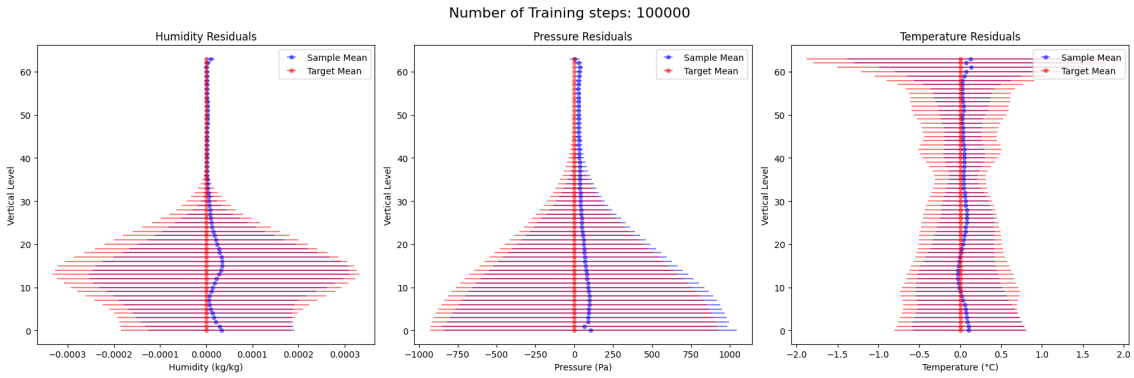

Figure A.9: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps.

A.2 Conditional Model

Table A.1: MSE for Humidity, Pressure, and Temperature across Classes split by land-sea mask, the presence of temperature inversions and orography over 100,000 training steps

Class	**Humidity (kg/kg)**		**Pressure (Pa)**		**Temperature** (°C)	
	MSE Mean	**MSE Std Dev**	**MSE Mean**	**MSE Std Dev**	**MSE Mean**	**MSE Std Dev**
Class 1	2.62×10^{-11}	6.87×10^{-10}	210	694	0.0037	0.0215
Class 2	3.20×10^{-11}	3.36×10^{-10}	418	2650	0.0043	0.014
Class 3	4.92×10^{-11}	1.42×10^{-9}	6990	8065	0.00638	0.0253
Class 4	1.90×10^{-10}	1.65×10^{-9}	19070	36900	0.0193	0.10095
Class 5	8.23×10^{-11}	1.37×10^{-9}	587	5440	0.00226	0.0837
Class 6	8.23×10^{-12}	2.22×10^{-9}	486	4430	0.000286	0.0517
Class 7	1.26×10^{-10}	3.80×10^{-9}	3560	24500	0.00447	0.103
Class 8	6.69×10^{-10}	1.97×10^{-9}	2930	8400	0.0102	0.0393
Class 9	1.036×10^{-10}	1.92×10^{-9}	2032	2690	0.00838	0.0396
Class 10	6.17×10^{-11}	4.086×10^{-9}	54700	1840	0.0214	0.0911

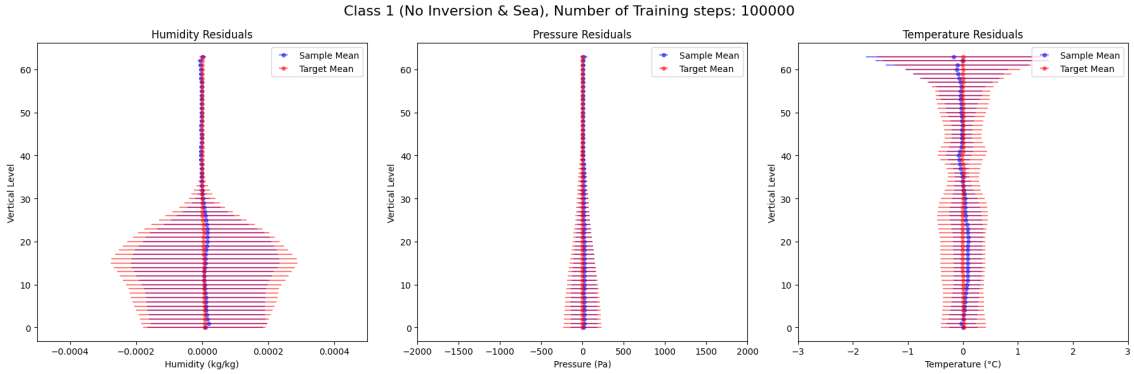

Figure A.10: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 1 (Sea samples where no temperature inversions are present)

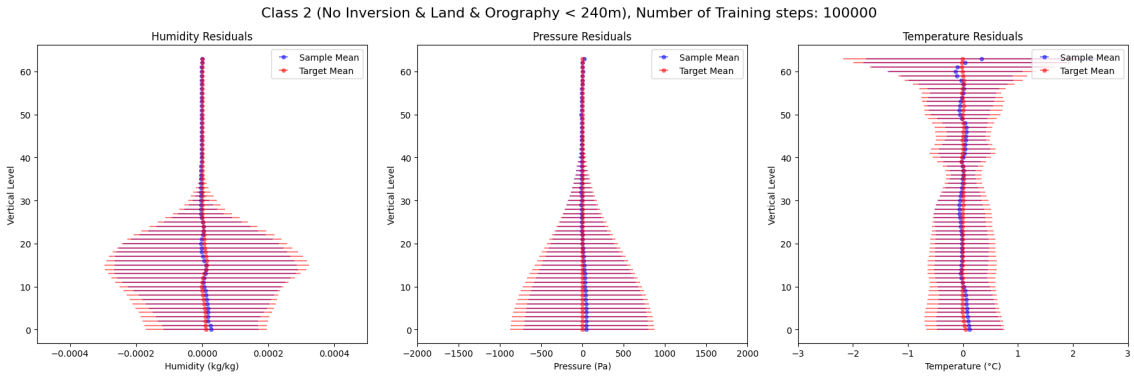

Figure A.11: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 2 (Land samples where no temperature inversions are present and Orography <240m)

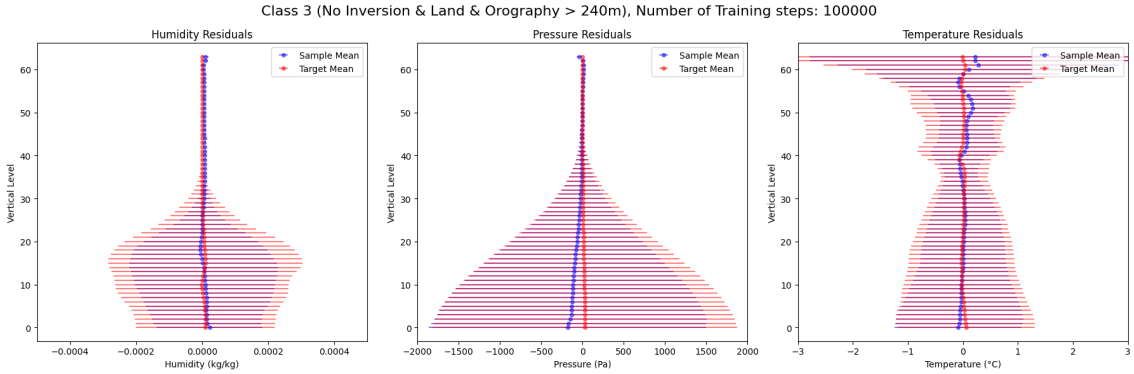

Figure A.12: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 3 (Land samples where no temperature inversions are present and Orography >240m)

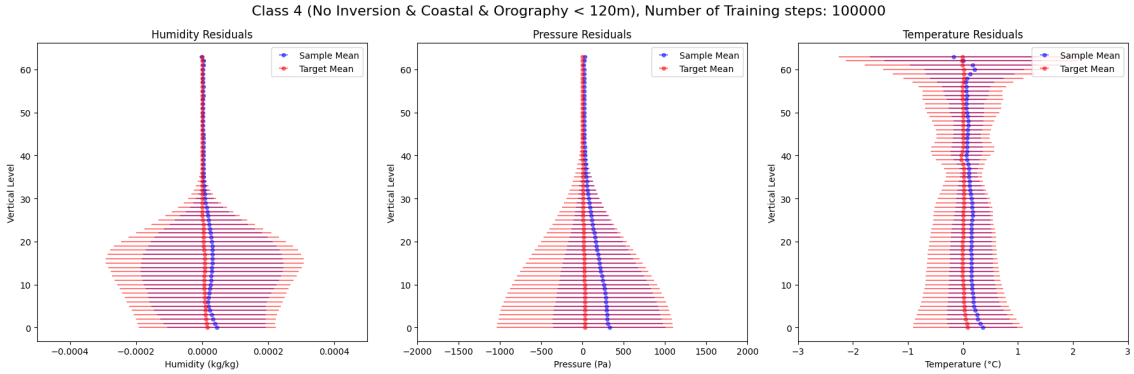

Figure A.13: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 4 (Coastal samples where no temperature inversions are present and Orography <120m)

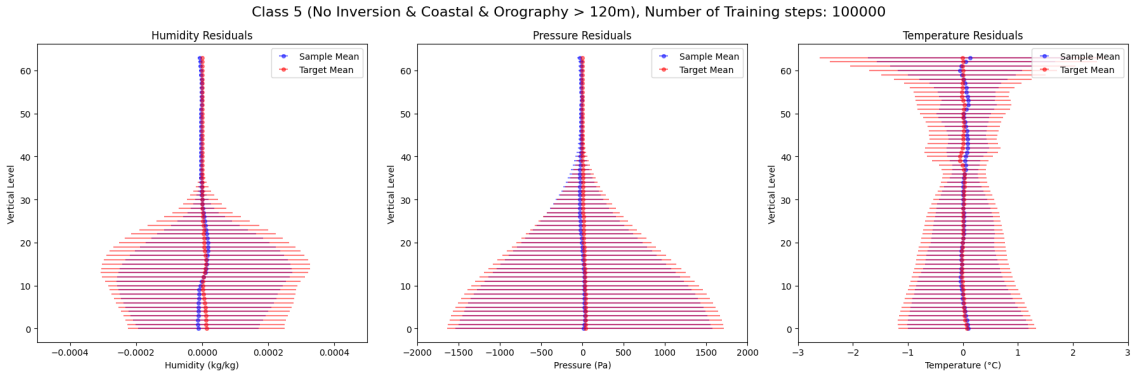

Figure A.14: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 5 (Coastal samples where no temperature inversions are present and Orography >120m)

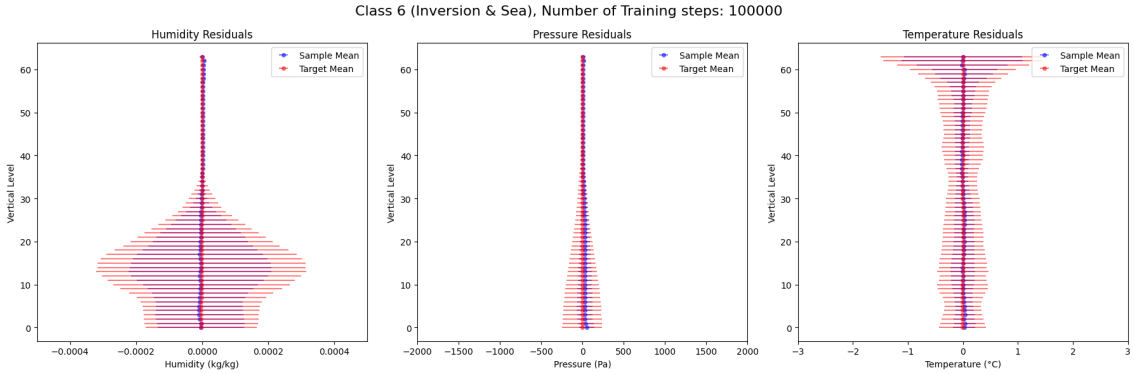

Figure A.15: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 6 (Sea samples where temperature inversions are present)

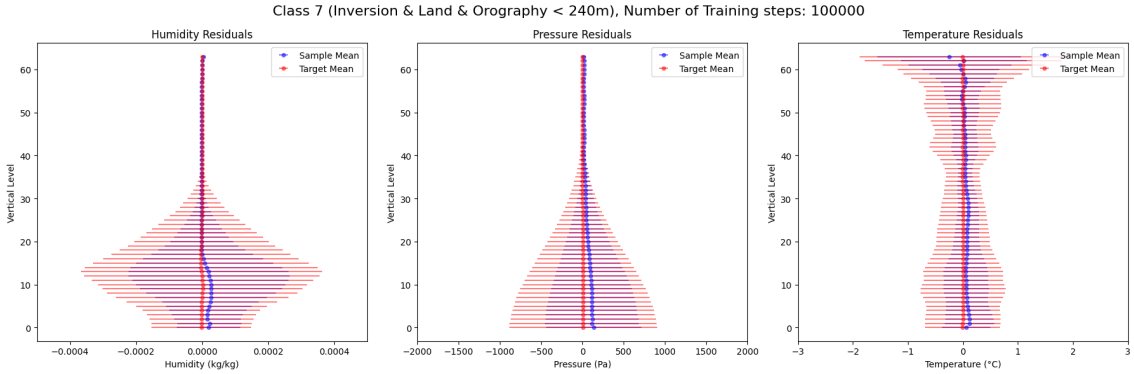

Figure A.16: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 7 (Land samples where temperature inversions are present and Orography <240m)

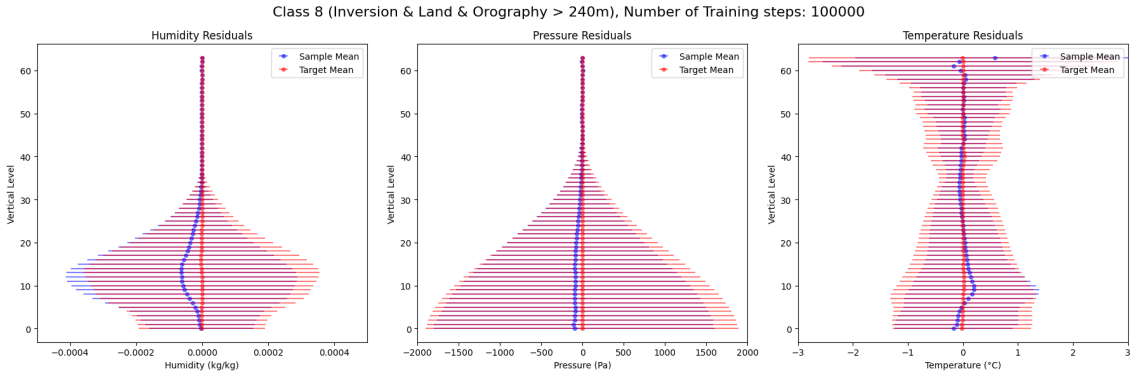

Figure A.17: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 8 (Land samples where temperature inversions are present and Orography >240m)

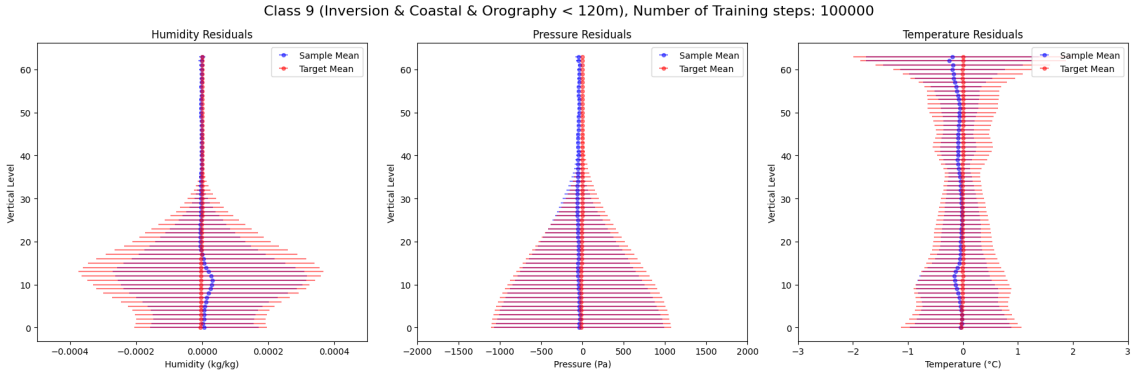

Figure A.18: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 9 (Coastal samples where temperature inversions are present and Orography <120m)

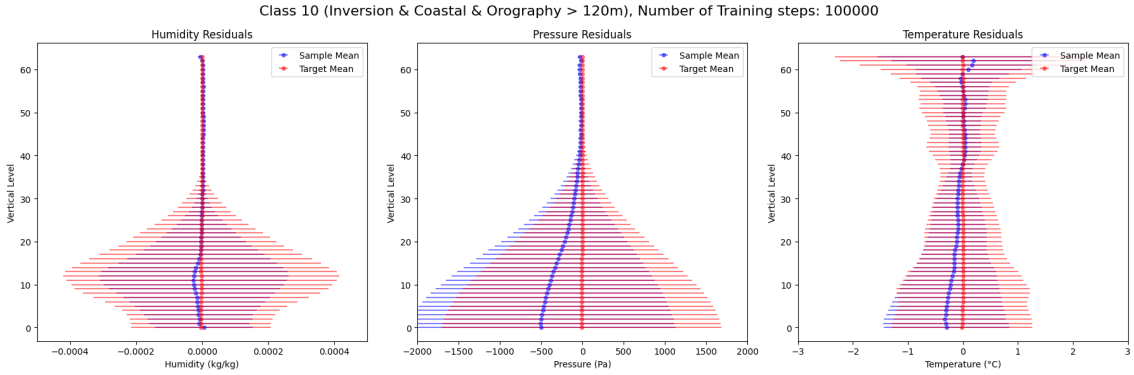

Figure A.19: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 100,000 training steps for Class 10 (Coastal samples where temperature inversions are present and Orography >120m)

Table A.2: MSE for Humidity, Pressure, and Temperature across Classes split by land-sea mask, the presence of temperature inversions and standard deviation of orography, over 10,000 training steps

Class	**Humidity (kg/kg)**		**Pressure (Pa)**		**Temperature** (°C)	
	MSE Mean	**MSE Std Dev**	**MSE Mean**	**MSE Std Dev**	**MSE Mean**	**MSE Std Dev**
Class 1	1.98×10^{-10}	2.56×10^{-9}	579	6860	0.0171	0.0979
Class 2	3.49×10^{-10}	1.18×10^{-10}	3043	1901	0.00967	0.0421
Class 3	3.27×10^{-10}	9.86×10^{-10}	4880	33800	0.00122	0.0732
Class 4	2.44×10^{-10}	1.45×10^{-10}	10630	2290	0.00345	0.0505
Class 5	4.65×10^{-10}	5.27×10^{-9}	15900	73040	0.0246	0.0851
Class 6	1.07×10^{-10}	2.16×10^{-9}	150	4570	0.0123	0.0236
Class 7	9.363×10^{-10}	2.543×10^{-9}	1120	3190	0.00193	0.0357
Class 8	2.895×10^{-10}	1.072×10^{-9}	38050	29310	0.00739	0.176
Class 9	1.84×10^{-10}	1.43×10^{-9}	20600	1530	0.0192	0.0154
Class 10	4.152×10^{-10}	2.38×10^{-10}	148000	2280	0.0347	0.629

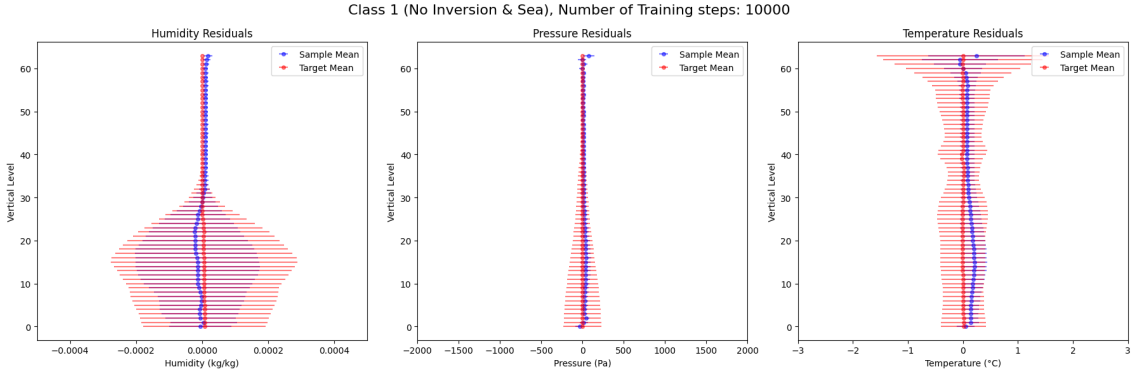

Figure A.20: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 1 (Sea samples where no temperature inversions are present)

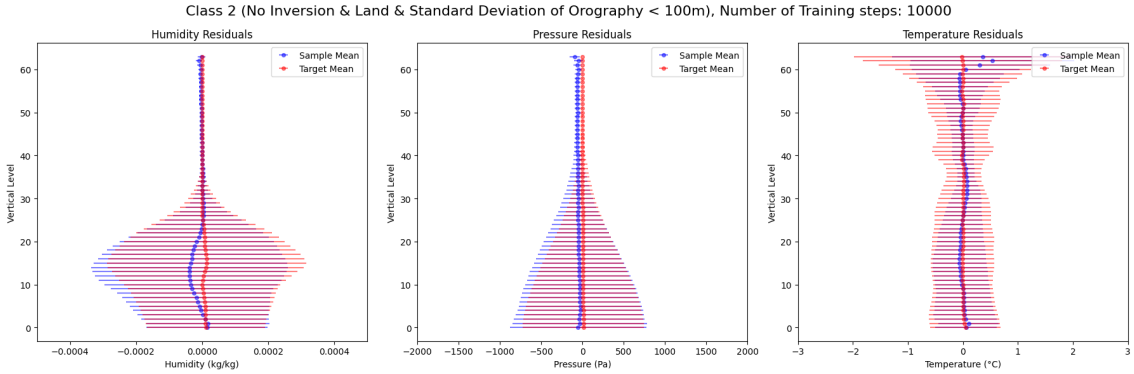

Figure A.21: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 2 (Land samples where no temperature inversions are present and Standard Deviation of Orography <100m)

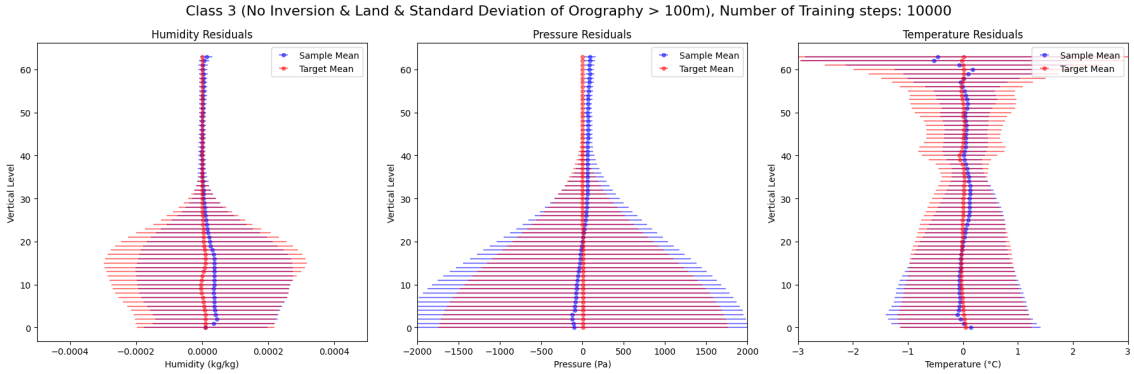

Figure A.22: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 3 (Land samples where no temperature inversions are present and Standard Deviation of Orography >100m)

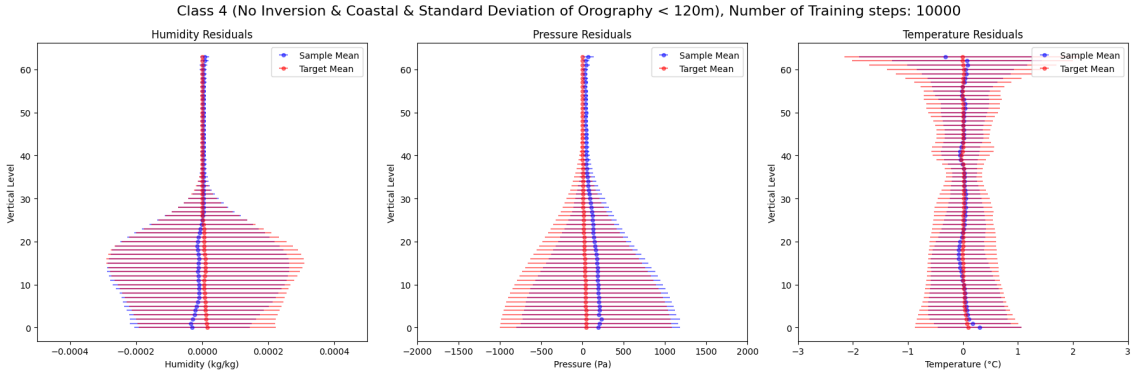

Figure A.23: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 4 (Coastal samples where no temperature inversions are present and Standard Deviation of Orography <120m)

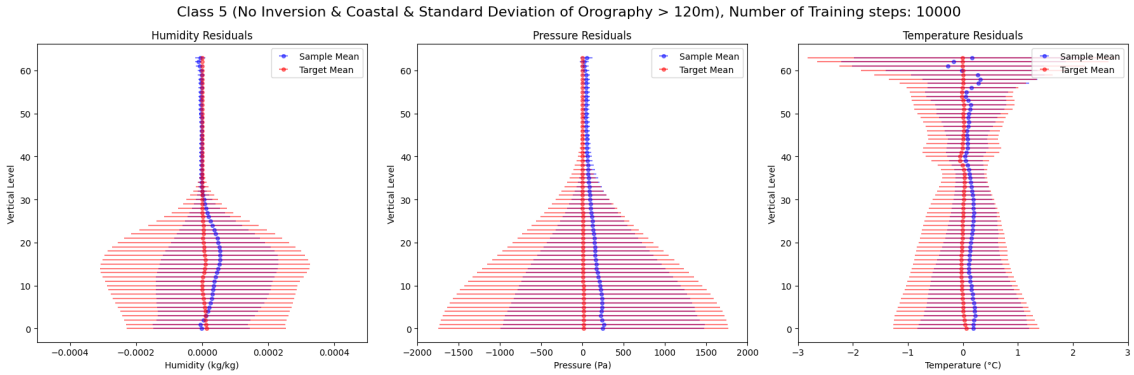

Figure A.24: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 5 (Coastal samples where no temperature inversions are present and Standard Deviation of Orography >120m)

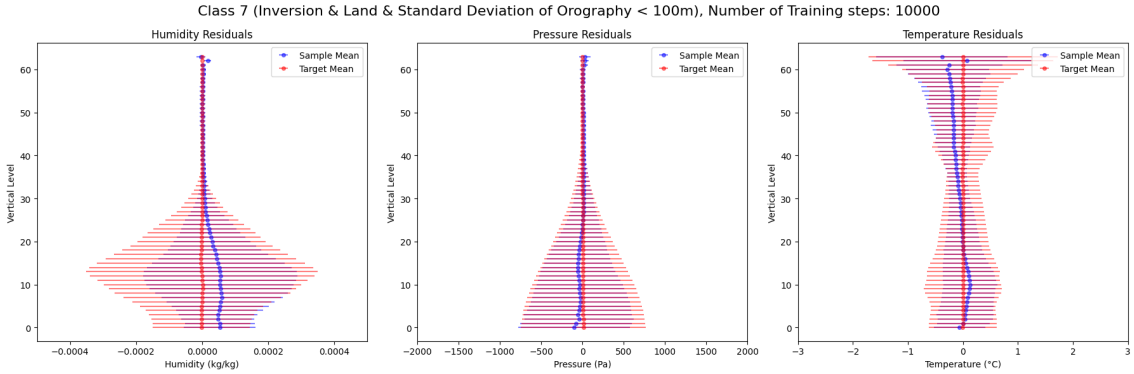

Figure A.25: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 6 (Sea samples where temperature inversions are present)

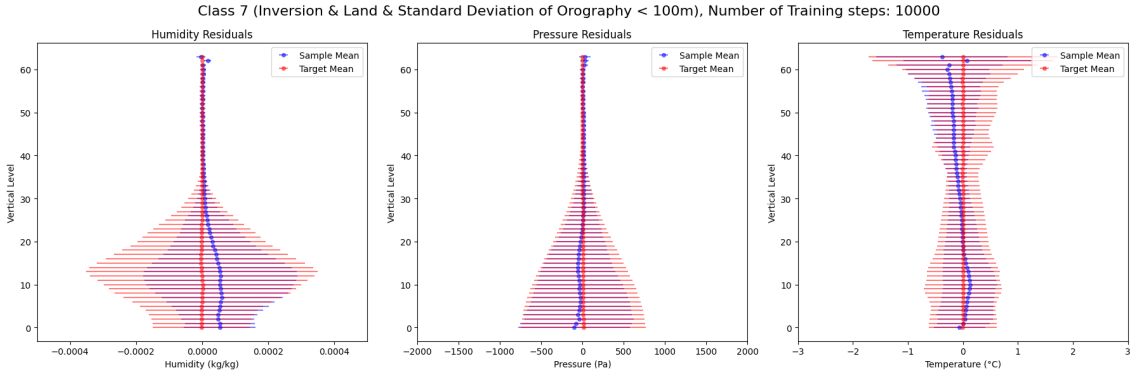

Figure A.26: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 7 (Land samples where temperature inversions are present and Standard Deviation of Orography <100m)

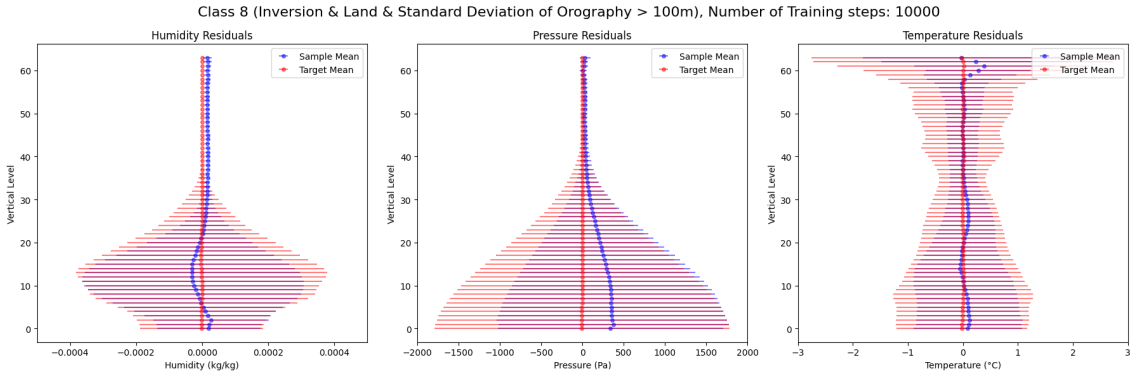

Figure A.27: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 8 (Land samples where temperature inversions are present and Standard Deviation of Orography >100m)

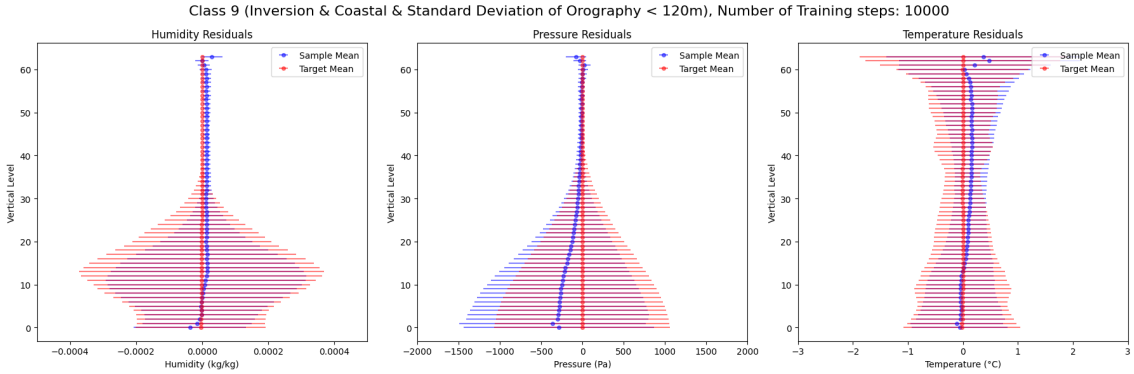

Figure A.28: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 9 (Coastal samples where temperature inversions are present and Standard Deviation of Orography <120m)

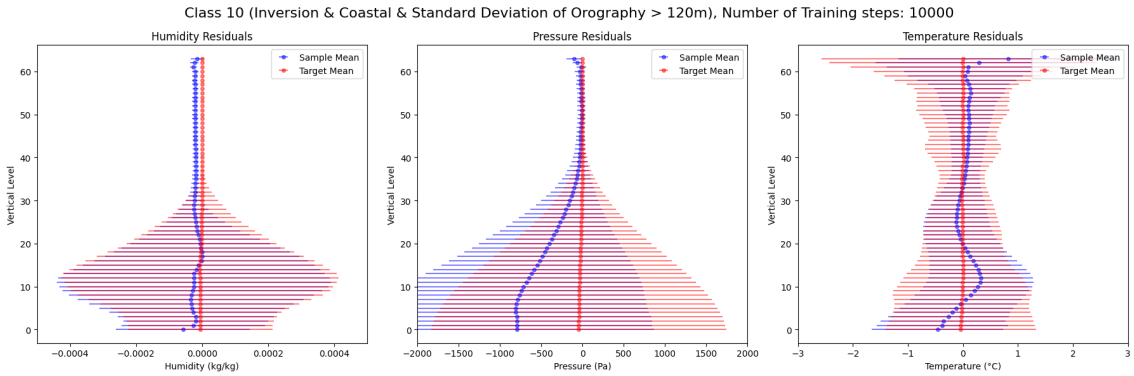

Figure A.29: Mean and standard deviation of the vertical residuals for humidity, pressure, and temperature at 10,000 training steps for Class 10 (Coastal samples where temperature inversions are present and Standard Deviation of Orography >120m)