UNIVERSITÀ
DI TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Artificial Intelligence Systems

Final Dissertation

A Foundation Model Approach for Tropical Cyclones Prediction

Supervisor
Sandro Luigi Fiore

Student
Gabriele Padovani

Co-Supervisor
Valentine Anantharaj

Academic year 2022/2023

Acknowledgements

I would like to express my sincere gratitude to all those who contributed to the completion of this project. Their valuable support, guidance, and encouragement were essential throughout this endeavour. I would like to firstly extend my gratitude to my supervisor, Prof. Sandro Luigi Fiore, and my co-supervisor, Valentine Anantharaj, for their invaluable expertise, patience, and support. A sincere thanks goes to my colleagues and peers, for their valuable discussions and feedback, which enhanced the quality of this project. Lastly, I am thankful to my friends and family for their consistent encouragement and understanding throughout this journey.

This work would not have been possible without the collective efforts from all of you, thank you.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

Contents

Abstract

In recent decades, tropical cyclones have remained a major concern to coastal communities, economies, and ecosystems worldwide. Objective evidence demonstrates the pressing requirement for precise and timely predictions, in order to support proactive disaster management and mitigation strategies. The severe impact of these high-impact storms emphasizes the urgency of this task. While traditional meteorological models have made significant progress in predicting tropical cyclones, the intricacies inherent in these systems require more advanced methods. This is where Artificial Intelligence emerges as a promising tool for revolutionizing tropical cyclone prediction. By utilizing the potential of deep learning, meteorologists can potentially improve the accuracy, lead time, and comprehension of these storm's behavior.

Machine learning techniques have been widely developed and applied, resulting in significant advancements in forecasting and managing of such events. Nevertheless, the models used in these techniques rely on selected and optimized versions of climate datasets, making them domain-specific, uniform and often inflexible to keep pace with the rapidly evolving climate landscape. By utilizing a novel approach for deep neural networks, the foundational model structure has been modified to resolve tasks related to the climate, while retaining the versatility and knowledge-based approach of transformers and large language models. The aforementioned model can be applied to solve various climate-related issues, which include specific aspects such as spatio-temporal disparities, data variable scopes, and dataset resolution, among others. The source code for this project is accessible at `https://github.com/HPCI-Lab/HackatonMachineLearning`.

1 Introduction

1.1 Document structure

In this section, we will delve into the structure of this thesis, the rationale behind the chosen sections, and how each chapter contributes to the overall objectives. The document revolves around six main chapters: an introductory one [Chapter 1] describing the context in which this work is positioned, the entities facilitating operations and the main topics of this project.

Then a theoretical background section [Chapter 2] is developed, where the current state of the art for foundation models, and more in detail how these approaches are applied to the field of weather prediction is explored. This section also introduces the main distributed machine learning approaches, which are crucial in dealing with problems caused by the extreme scale of these architectures.

In the implementation chapter [Chapter 3], the main design and development decision are explained, detailing how each module's addition influences the dimensionality of the encoding space, as well as which type of dataset is utilized for each stage.

In the results section [Chapter 4], statistics for how accurate the model is are presented. Several modalities are tested, spanning the number and resolution of the variables, lead-time of prediction, and scalability of the training in a multi-worker environment.

Finally, some further development points are discussed in the future work chapter [Chapter 6].

1.2 AI for weather prediction

Before discussing the work designed for this project, it is important to define some key concepts related to Artificial Intelligence (AI). Firstly, AI is able to perceive, calculate and adapt to its surroundings and current state, demonstrating intelligence. It is worth noting that abbreviations like "AI" should be clearly explained when first used. In contrast to a regular computer program, an AI-based program may deduce the correct solution even without explicit programming for that specific task. Real-world applications in this domain are, for example, autonomous planning robots and self-driving cars. It is essential to define a subset of algorithms used in these applications, namely, machine learning. These are statistical models that can infer a solution probabilistically when presented with a certain input, which is often unseen by the program. Examples of such applications include object recognition in images and intent classification in text-based tasks. Finally, the third definition is that of Deep Learning, which denotes a subset of machine learning techniques where neural networks are utilized to recognise patterns within an input dataset, with the aim of achieving high performance on structurally similar, previously unseen data. The term "Deep" distinguishes these models from regular neural networks, as they often exceed the threshold of millions of parameters, requiring extensive amounts of data and large computational resources to complete the training phase in a reasonable time.

As for the development of such applications, in [MCS$^+$23], the authors provide a set of guidelines for the use of AI in weather prediction, with the aim of providing a framework for the creation of AI-based weather forecasting systems. It is emphasized that climate forecasting requires different approaches and models at different prediction timescales, for instance, regional numerical weather prediction (NWP) models are used to forecast a few days into the future, while models with global coverage are used for longer-term predictions, such as ten or sixteen days into the future.

The simplest approach to AI-based weather forecasting is using techniques from computer vision, such as CNNs, to treat the problem as an image classification task, where the input is a satellite image and the output is a prediction of the weather. This approach allows for the propagation of simple features, such as cloud patterns, a certain amount of pixels into the future, but it is not able to capture the complex dynamics of weather systems. This causes issues when the model is able to freely make predictions about the climate, but often disregards atmospheric dynamics and physics.

Another issue with these models, is the fact that they are restricted in terms of their spatial capabilities. This is due to the fact that the input data is often provided by satellites, which have a limited resolution, and the fact that the computational cost of training a model increases with the number of pixels in the input image. This means that the model is unable to capture small-scale features, such as the eye of a hurricane, which can be crucial for accurate predictions. One way to avoid this issue is ”super-resolution”, which consists of using another model to increase the resolution of the input image, often by interpolation, this however can be computationally expensive and may not be feasible for real-time predictions.

While these basic machine learning approaches can be viable for some weather related applications, foundation modeling has emerged as a promising new approach for weather forecasting. Benefits include flexibility and adaptability to different tasks, improved generalization capability, the ability to be trained in a self-supervised manner, mitigating the need for expensive labeled data, and the integration with multi-modal data sources, such as satellite images and NWP models.

1.3 Foundation models

In a time when the scene of deep learning is ever evolving, with novel approaches and architectures being developed every day, one technique has emerged as a promising candidate for taking the field to the next level. Foundation models are large machine learning models that are often used as the backbone for other predictive models, featuring a number of parameters which is often in the millions if not billions. These architectures are typically trained on large amounts of data and are designed to capture general patterns, features and representations present in the data. This initial training phase, often termed ”pre-training”, is followed by a ”fine-tuning” phase, where the model is adapted to a specific task, requiring a much smaller dataset. The aim of this model is not necessarily to surpass a simpler machine learning approach in terms of efficiency. Instead, its primary advantage lies in its flexibility, for use in different applications. Since the initial pre-training phase is performed on a large dataset, of several generally scoped variables, the model can be adapted to excel in a wide range of more specific tasks, called ”downstream tasks”, without the need for extensive training data. A more detailed explanation is provided in section 2.1.

Made famous by ChatGPT, these architectures have become increasingly popular, particularly in natural language processing, with models such as GPT and BERT, and in computer vision, with VGG and transformers. These are, however, only a few examples of the many applications of these models, which have also been used in medicine, biology and more. While the initial emphasis of these models was on text generation and understanding, and consequently the prominent breakthroughs were in the field of natural language processing, the potential of these architectures has been explored in other domains as well.

One promising new field of application is that of weather and climate science, where the ability to predict and understand the behavior of complex systems is of great importance. Several papers have been published in recent years, demonstrating the potential of these models in this field, such as [NBK+23] and [BXZ+22].

1.4 Foundation models for climate

While traditional forecasting methods can be effective, they often struggle with the intricate dynamics and rapid changes displayed by complex weather systems. In recent times, the convergence of plentiful data sources, augmented computing capability, and progressions in AI algorithms has created fresh possibilities for improving prediction models. In addition, the emergence of foundation models offers a promising way for significantly enhancing tropical cyclone prediction.

Several efforts have been made to apply these models to climate science, with the aim of improving the accuracy and lead time of predictions. In particular, the European Centre for Medium-Range Weather Forecasts (ECMWF) has recently published a roadmap for the use of machine learning in weather forecasting, which includes the use of foundation models. In this document [DMG+21], the authors lay out a plan for the use of these techniques in the field of weather prediction, with the aim of exploring novel approaches, foster inter-domain collaboration, as well as provide a state of the art framework for the use of deep learning in climate tasks at scale. The document also provides a list of milestones, such as the standardization of machine learning tools and benchmarks, and culminates with the end goal of fully integrating both supervised, but also unsupervised learning approaches into the ECMWF forecasting system.

This thesis investigates the incorporation of AI methods in tropical cyclone prediction, with the goal of examining their ability to transform the manner in which we predict and react to such powerful natural occurrences. In particular, the focus will be on a specific architecture known as the vision transformer, an attention-based model adapted for images instead of text. The analysis will determine whether a foundation model is suitable for weather forecasting and can be fine-tuned to predict large-scale catastrophic events, such as tropical cyclones.

1.5 Digital Twins

The adoption of digital twin technology [ES18] rose during the digitalization of manufacturing systems and machinery in the early 2000s. This idea has a wide range of applications across various technologies and is expected to cause significant disruptions across industries beyond manufacturing. This can be achieved by enabling the ability to create digital models and simulate real-time experiences. Through the seamless transmission of data between the physical and virtual worlds, digital twins will provide the means to monitor, comprehend and efficiently optimize all physical entities' functions. Digital replicas of humans could also allow for the gathering and assessment of physical, physiological, and contextual data to enhance quality of life and well-being.

Digital twins would have distinct characteristics, including a singular identifier for communication with their counterpart, employment of deep learning techniques for swift decision-making on behalf of their real-life counterpart. They should be capable of near real-time interaction with the environment, real-world counterparts and other digital twins.

An example of a digital twin implemented for the use case of global forecasting is the Destination Earth project [NMC21] [Ulm21]. It has the potential to create useful insights to tackle global challenges, aid in the transition to sustainability, and support nations' green goals.

1.5.1 The Destination Earth project

The European Destination Earth project (DestinE) is an initiative supported by funding from the EU Digital Europe Program and aligns with the EU Green Deal and digital strategies [HBS+23]. Its objective is to facilitate informed decision-making by developing an interactive digital information system that employs advanced technologies to create a digital twin of the Earth. Currently, the available information systems attend to various needs and have evolved correspondingly to the public operational monitoring and prediction systems that are accessible. DestinE aims to accelerate information systems through advanced digital technologies in areas where progress must exceed expectations.

The key developments to support climate change adaptation involve enhancing the spatial resolution of climate models to scales that will resolve real-life systems in the Earth's system, instead of just approximations. This is coupled with the improvement of the interaction between models of physical and societal behaviours, thus linking science and adaptation. Digital twins must possess the capacity for scenario testing, and must be effectively designed to manage a complex system at an appropriate scale. DestinE intends to achieve the required level of quality in the aforementioned innovation areas within a timeframe of 7-10 years.

The project's use cases are divided into three categories [CCNC20] based on the progress stage of DestinE. Mature use cases involve improving disaster risk management, considering the rising anxiety over the resilience of developmental undertakings amid growing susceptibility and exposure to disasters. Another use case in this category is climate change adaptation, which is to be considered crucial for industries like agriculture, which rely on external variables, such as weather patterns, and their capacity to adjust to changing circumstances. These changes typically occur over a 10-year timescale, prompting the need to investigate new tools and research for climate predictions that aid climate change adaptation. Finally, climate modelling is an essential tool that is continually expanding through the addition of newer and more advanced representations of crucial biochemical processes. These models are commonly known as Earth System Models, and they are the most dependable sources for long-term projections regarding global climate change.

An important application which is central to this project is the medium-term seasonal forecasts for hurricanes and typhoons. These forecasts are crucial due to the significant impact tropical cyclones have on both developed and developing countries, strengthened by the effects of climate change. This is due to both the strengthening of storms and the increase in exposure of people and infrastructure to their effects in coastal areas.

Less mature use cases tend to concentrate on oceanic models, which include the development of a marine modelling framework, freshwater and rising sea-level simulation, and possible uses for renewable marine energy production and carbon modelling. The previous models aim to replicate marine ecosystem conditions under external stress factors, while enhancing predictability during extended lead times. This limitation presents a significant social vulnerability. The aim of these applications is to improve companies' understanding of the advantages and disadvantages of establishing renewable energy test sites.

Finally, there are use cases that have yet to be expanded upon, such as the development of a digital twin for local and urban areas. This could facilitate the creation of city adaptation plans and simulations of emergency response operations.

1.5.2 Earth System Digital Twins

The Earth System Digital Twins project [BSH21], initiated by NASA, aims to advance new technologies for integrating Earth and human models. The goal is to provide a universal representation for forecasting and decision-making through interconnected, large-scale, multi-domain modeling. The three primary components are to be assessed.

- **Digital Replica**: Developing a series of frameworks that offer uninterrupted and precise depictions of systems as they evolve over time;

- **Forecasting model**: Utilizes machine learning and statistical methods to generate future states in real-time or near real-time;

- **Impact Assessment**: Utilizes the aforementioned two points to better comprehend the future effects for prolonged periods. The intention is to address uncertainty quantification, advanced computation, and visualisation for efficiently running a substantial number of simulated predictions.

The project has multiple objectives. Some include, for example, establishing a framework that guarantees the dependable depiction of systems as they develop, emulating the earth science model to anticipate how the earth reacts to particular incidents, and creating tools for investigating theoretical scenarios.

1.6 High performance computing

HPC systems are designed to handle massive amounts of data and run computations in parallel, enabling researchers, scientists, engineers and organizations to solve complex problems that would be impossible or extremely time-consuming on regular computers. These systems use multiple processing units, such as processors or cores, to perform tasks simultaneously, and are equipped with significant memory and storage capacity. These cores are typically organized into collections of interconnected computing nodes known as clusters or supercomputers. High performance computing centers are extremely useful for a wide range of applications, some of which are physics simulations, quantum computing, and deep learning. In later years, as distributed training techniques for large machine learning models have become more and more optimized, the importance of these centers for the progress of artificial intelligence research has become crucial.

Having access to high-performance computing hardware that competes for the top spots in the world's most performant clusters, this project aims at analyzing the scalability of a machine learning model's training phase in a multi-node and multi-GPU environment. To achieve the fastest training times and maximal resource utilization, the limitations of GPU peak performance and the maximum bandwidth of the file system will also be considered.

1.6.1 Oak Ridge National Laboratory

ORNL [Mas12] is one of the leading research institutions specializing in HPC, Big Data and AI. It focuses on climate change, clean energy, and materials and computational sciences. The center has several super computing architectures, some of which, such as Frontier and Summit, are among the world's most powerful clusters. For this project, the laboratory provided access to two supercomputers: a commodity supercomputer called Andes and a more advanced supercomputer called Summit.

1.6.2 Hackaton

As explained in [AHP$^+$22], a multi-petabyte dataset will be made available for open science via a data hackathon. To alleviate the problem of transferring this large amount of information, all participants will be given access to the Oak Ridge National Laboratory file system and computing facilities. This hackathon, developed in collaboration with the European Centre for Medium-Range Weather Forecasts, aims at enabling a more in-depth research and improve numerical weather and climate prediction, using a dataset that is more detailed and higher resolution than any other currently available.

Projects launched using this dataset cover a range of topics: severe weather forecasting using artificial intelligence; improved understanding of topography; innovative visualisation techniques; and training the next generation of climate scientists through a summer school.

All partecipants have been granted access to a commodity cluster present at OLCF called Andes, to allow for scaling of their research. It is also the case for this work. In case a project may require more processing power, it is also possible to require access to a larger supercomputer, such as Summit, which is at the time of the writing of this report, in the Top 5 of most performing machines in the world.

1.6.3 Andes

Andes is a small cluster often used for data pre-processing and analysis of simulations run on larger architectures. It has two partitions, the first one with 704 cpu nodes, each containing two 16-core 3.0 GHz AMD EPYC 7302 processors and the second one with 9 GPU nodes, with 1TB of memory and two NVIDIA K80 GPUs in addition to two 14-core 2.30 GHz Intel Xeon processors each. Andes also uses 8 login nodes, which function as destinations where users can log in. Environment modules on Andes are provided by lmod, a system for modifying the software available in a shell configuration without the risk of creating package version combinations that cannot coexist in a single environment. it is aware of module compatibility and actively modifies the environment to protect against conflicts. It is also possible to define custom module collections to quickly set up a working Python environment. To schedule jobs, Andes, like most legacy OLCF hardware, uses the Slurm batch scheduler. The most common way to start a job is to create a scheduling script, which is just a bash script with SBATCH options that the program will follow. The sbatch command is used to queue the script to wait for the necessary resources, and the srun command is used to start execution on the newly available nodes. Slurm also allows you to easily reserve resources for a period of time, making it possible to create interactive job scheduling environments.

1.6.4 Summit

Summit, on the other hand, is much more modern and powerful, and remains one of the top five supercomputers in the world. it has approximately 4,600 compute nodes, each equipped with two IBM POWER9 processors and six NVIDIA Tesla V100 accelerators In addition to the compute nodes, there are two other types of compute unit in Summit, Login nodes, which can be used to edit and run code, and are the platforms onto which the user logs in; and Launch nodes, which are used to start a job once scheduled by the user. A small number of nodes are also configured as high memory, these contain larger DDR memories, as well as more GPU RAM. While being connected to the center-wide NFS-based file system, Summit is also connected to an IBM filesystem with 250PB of storage capacity with a peak write speed of 2.5 TB/s. The environment modules can be loaded in a similar manner to Andes, utilizing lmod. However, one distinction is the presence of a pre-existing conda environment named Open-CE, which stands for Open Cognitive Environment. This bundle provides an extensive range of machine learning and deep learning frameworks, in addition to nearly all needed libraries for this project. The standard method for initiating tasks on Summit involves utilizing jsrun, an ad-hoc utility crafted for Oak Ridges computers by IBM. It is the default method for executing such jobs. It schedules a program on compute nodes, allocating resources through the LSF batch scheduler. Unlike Andes, when a batch script or interactive job is run, the resulting shell runs on a launch node, whereas compute nodes are accessed using the jsrun command. This command should also only be called from a launch node, i.e. from a batch script or interactive shell. Similar to scheduling scripts on Andes, the script is simply a bash file containing LSF options that are interpreted to decide the amount of resources to allocate. One important feature of the jsrun scheduler is resource sets, which provide greater flexibility by enabling users to determine how a set of nodes will appear to the program. The jsrun command line arguments enable specification of the number of resource sets and the number of GPUs, CPUs, and tasks available to each resource set. The allocation of resources can be guided by additional arguments, such as sequential task assignment.

2 Theoretical Background

This chapter presents the fundamental theoretical framework for this research on tropical cyclone prediction. We explore significant meteorological principles, historical advancements, and modern forecasting methodologies that lay the groundwork for our investigation. It is regarded to as essential to comprehend this theoretical foundation to understand the context and technicalities employed in this study.

2.1 Foundation Models

Foundation models [ZLL+23] are large machine learning models that are often used as the backbone for other predictive models. These models are typically trained on large amounts of data and are designed to capture general patterns, features and representations present in this information. Once trained, they can be fine-tuned or adapted for specific tasks, reducing the need for extensive training data and computational resources. In recent years, these architectures have become increasingly popular, particularly in natural language processing, with models such as GPT [Dal21] and BERT, and in computer vision, with VGG and transformers. While different implementations may have modified the underlying behavior, foundation models work by gaining a general understanding of the data they are fed with in a phase called pre-training, which can then be used to optimize any downstream task with a much smaller number of samples [SPG22].

The aim of these models is not necessarily to surpass a simpler machine learning approaches in terms of efficiency. Instead, the primary advantage lies in its flexibility for use in different applications. One example of this is the implementation of several foundation models in medicine, where a significant amount of knowledge is necessary to ensure the best decision for a patient. Here, the model can undergo pre-training using a vast medical record dataset, then fine-tuned to accomplish a particular task, like disease prediction or diagnosis. While this approach may seem more intuitive, these architectures have also been put to the test on what is referred to as generalist medical AI. In this scenario, the model is expected to perform a range of tasks with minimal or no task-specific labelled data [MBA+23]. The models will function similarly to a medicine-focused chatbot, offering thorough explanations via text, spoken suggestions, or annotated images that depict the decision-making process of the model. This is a crucial feature in the medical field, where it is imperative to understand the reasoning behind each decision, especially when dealing with matters of human life. This also offers a significant advantage over traditional machine learning models, often called "black boxes", as they cannot provide a rationale for their decision-making.

2.1.1 Popularity of LLMs in later years

It's clear that in later years, models like the GPT-3 became much more popular. The ability to train these models with 175 billion parameters has allowed the creation of architectures with ever-increasing learning capabilities and versatility in performing a wide range of natural language processing tasks. Examples range from text generation and language translation to sentence completion and question answering. Apart from the well-known example of ChatGPT, various models have been deployed over the years with excellent results. GitHub Copilot boosts programmer productivity by reducing the amount of manually written boilerplate code, while BERT, a foundation model for context understanding, is used by numerous platforms, including Netflix and YouTube [BHA+22], to offer content recommendations based on user preferences.

In addition, the two-phase nature of the basic models, with pre-training being the more computationally expensive, allows small companies that do not necessarily have this computational capability to use a freely available pre-trained version and fine-tune it for their task. This not only requires smaller datasets, but still allows for competitive performance not achievable with other machine learning approaches.

However, researching and improving these models poses several challenges. Firstly, due to the emergent nature of foundation models, in-context learning functionalities are only achievable in large-scale models. Hence, access to sufficient computing capability is fundamental, and not all academic laboratories possess it yet. Community initiatives like Hugging Face are working towards training large foundational models, but the disparity between the private models that industry can train and those accessible to the public will probably continue to exist and could even widen. Large technology companies operate on a distinct level in terms of their resources, particularly with regards to their infrastructure, user base, and data coming from their market position.

Reproducibility is a significant challenge for these models as well. Private companies often own architectures that lack transparency with no exhaustive explanation of the development and decision-making processes, rendering them not open source. The challenge is exacerbated because the models take in an extensive set of information, with limited datasets shared publicly, and many not even shareable due to their vast size.

2.1.2 Foundation models applications and advantages

Other applications, aside from the one discussed in Section 2.1, comprise protein design [FH22], general computation engines, and remote homology detection [LGAM21].

The former refers to a preliminary transformer prototype that is trained for protein generation in a comparable manner to text generation. Developing generative text models to create innovative proteins is a highly promising and under-explored area that would greatly benefit from the creation of an ad-hoc language model. The latter, on the other hand, is a transformer-based model designed to predict the protein family of a given sequence. This task is challenging because the sequences of proteins belonging to the same family can differ immensely. Consequently, the model must capture the underlying patterns and representations that define each category.

This work is also part of a wider discussion on the use of transformers as universal computing engines, which have the ability to perform diverse tasks without requiring extensive fine-tuning after being pre-trained on general natural language data. Other tasks this model has undergone include numerical computation, such as XOR and bit memorisation, and image classification, where the model achieved a competitive accuracy in classifying images from the CIFAR-10 and MNIST datasets. Experimentation has been conducted by freezing all self-attention layers in the transformer, and only training the final prediction head. This demonstrates the architecture's ability to transfer its knowledge to another modality, thereby proving the hypothesis that foundation models can perform computation-agnostic learning.

2.1.3 Transformer

The Transformer architecture, first introduced in [VSP$^+$17], is a deep learning model designed to handle sequential data, such as sentences, by capturing long-range dependencies and relationships within them. The key innovation in Transformer is a self-attention mechanism that allows the model to weigh the importance of different elements in a sequence when processing each element. This mechanism allows the context and relationships between words or tokens to be taken into account, leading to improved performance in tasks such as machine translation and sentiment analysis.

Most sequence models employ an encoder-decoder structure to link an input symbol with an output symbol. However, this approach is limited in that it can only deal with one token at a time. The transformer approach follows a comparable method by repeatedly stacking up self-attention, point-wise and fully connected layers, as shown in Figure 2.1. The benefit lies in the fact that encoder and decoder blocks can be used independently.

The encoder module comprises six identically stacked layers. Each layer contains a multi-head attention mechanism, a position-wise fully connected layer, a residual connection, and layer normalization to prevent gradient explosion. To facilitate the learning process and given the need for repeated blocks, each sub-layer has a hidden dimension of 512, resulting in a consistent latent space. The structure of the decoder module is symmetrical to that of the encoder, but includes a third multi-head attention layer for the output of the encoder. Additionally, the first multi-head attention layer in every block is masked, and the input is shifted by one to ensure each token is dependent solely on the preceding inputs.

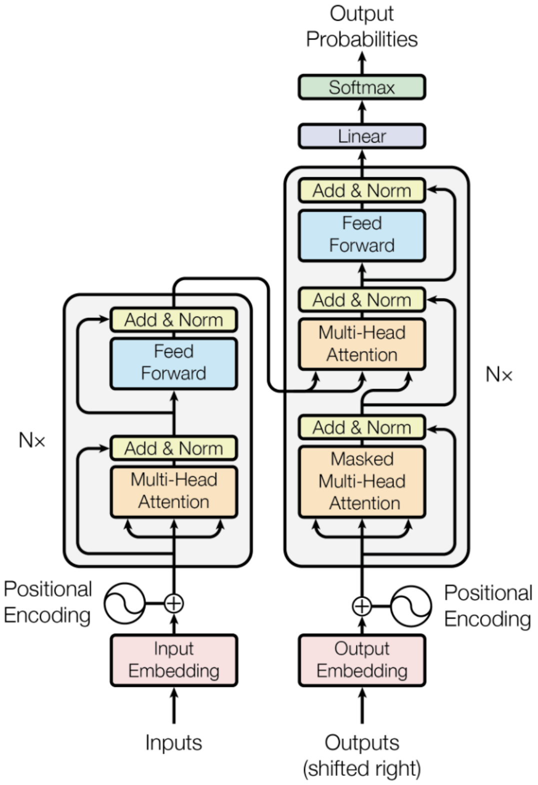


Figure 2.1: The transformer model architecture - Source: ”Attention Is All You Need”

$$\text{Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}}) \tag{2.1}$$

Further exploring the attention mechanism, it revolves around mapping a query array to a collection of key-value pairs, resulting in an output vector. The outcome equals a weighted sum of values, where the weights derive from the similarity assessment between the query and value vectors. To avoid gradient explosion caused by several consecutive dot products, equation 2.1 divides by a $\sqrt{d_k}$ factor, which identifies the dimensionality of the querys and keys. This problem may not have a significant impact on regular calculations, but it can cause considerable deviation in results when performing multiple dot products.

The difference between a single attention layer and a multi-head attention layer comes from the dimensionality of keys, values and queries, where the three components are linearly projected h times. Each of these vectors is then passed in parallel through an attention module, concatenated and again linearly projected, as shown in equation 2.2. The advantage of this technique is that the model learns jointly from different representation sub-spaces at different positions, resulting in a more accurate encoding of information.

$$\begin{aligned}\text{MultiHeadAttention}(Q, K, V) &= \text{Concat}(H_1, H_2, ...)W^O \\ whereH_n &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned} \tag{2.2}$$

The position-wise feed forward layer comprises of two linear layers separated by a ReLU activation function, resulting in equation 2.3.

$$\text{FFN}(x) = max(0, xW1 + b1)W2 + b \tag{2.3}$$

To incorporate token sequencing information, a positional encoding block is applied prior to inputting to the architecture, and again before processing the previous output token.

2.1.4 Vision Transformer

The Vision Transformer [DBK+20] is an adaptation of the Transformer architecture for computer vision. It applies the principles of self-attention to images, treating them as a sequence of patches that are then processed by the Transformer layers. Each patch is treated as an input token, and the self-attention mechanism captures the interactions between different patches. This approach has demonstrated competitive performance against traditional convolutional neural networks in image classification tasks.

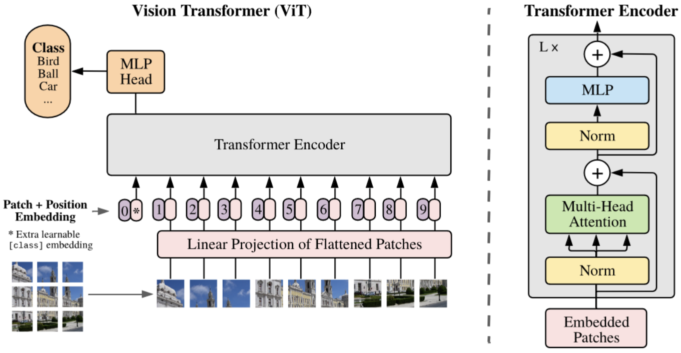


Figure 2.2: The vision transformer architecture - Source: ”An image is worth 16x16 words”

While the conventional Transformer accepts a 1D sequence of token embeddings as an input, to process 2D images, they are reformatted into a sequence of flattened 2D patches. Positional embedding is then added to the patch embeddings for retaining the positional information. A one-dimensional learnable position embedding is utilised as there was no noteworthy improvement in performance when implementing a more sophisticated two-dimensional encoding within the model.

The vision transformer, much like the transformer architecture, is composed of several blocks featuring alternating multi-layer attention, layer normalization, feed forward layers and residual connections. Refer to Figure 2.2 for illustration. While there are several benefits to using this architecture, two key advantages can be identified. The initial advantage of this type of model is its consistent lack of inductive bias when compared to other convolution-based architectures. This implies that the model is considerably less likely to overfit on the training data, and that the positional information is utilized solely at the beginning, yielding generalized information. The second advantage lies in the input’s flexibility, which can be passed to the model without necessarily conforming to any specific format or structure. It doesn’t have to only consist of image patches. Instead, it can comprise any collection of data, such as the feature map from a convolutional layer.

2.1.5 Pre-train and Fine-tune

The pre-training phase constitutes the initial training stage, where a neural network model of immense scale is trained on an extensive corpus to extract general patterns, characteristics and representations present in the data. Its objective is to obtain a model that can capture meaningful information from the data and serve as a foundation for a broad range of subsequent tasks. In this phase, the model undergoes training on a vast corpus of textual data, often obtained through web scraping. One example of this is ChatGPT, which learnt from GitHub code, as well as Wikipedia and other websites’ information. Other pre-training techniques comprise data masking, in which a fraction of the image or text is concealed, prompting the model to guess about the absent parts, and de-noising, where pseudo-random noise is applied onto the image to boost the network’s reconstruction ability. After this stage, the output is a model that has acquired a sophisticated language representations. These representations include semantic meaning, syntax and contextual data and can be viewed as ”knowledge” of the domain acquired from the extensive data used to train it. These pre-trained models can be fine-tuned for distinctive downstream objectives through limited amounts of task-specific data.

The fine-tuning phase consists of taking a pre-trained model, which has learned general patterns and representations from a large dataset, and adapting it to a specific downstream task. The process involves taking a pre-trained model, removing the prediction head, which is often the last linear layer in this type of architecture, and replacing it with another. This newly added layer set is then trained on a much smaller dataset compared to the pre-training one, while all other blocks are frozen. This results in faster execution times, as the backpropagation phase only needs to be brought forward on the final layers.

2.2 Cyclones physics

A tropical cyclone is a storm system that rotates rapidly, featuring a low-pressure center, intense winds, and an organized series of thunderstorms that cause intense rain and sudden gusts [Gra98]. The term tropical refers to the geographical origin of these systems, which form almost exclusively over tropical seas, while cyclone refers to their winds moving in a circle, around a central eye, with surface winds blowing counterclockwise in the Northern Hemisphere and clockwise in the Southern one. Tropical storms usually develop over vast warm bodies of water. They obtain their power from its evaporation from the sea's surface, which results in clouds and rainfall once humid air rises and cools due to saturation. This energy source differs from other cyclonic storms, which may be powered primarily by horizontal temperature contrasts. These cyclones have a diameter most often found between 100 and 2,000 km. The powerful swirling winds of a tropical cyclone, as the ones shown in Figure 2.3, arise due to the Earth's rotation imparting angular momentum as air moves towards the axis of rotation. These storms are generally most severe when over or near water and quickly lose intensity when moving over land. Damage can result from strong winds, rain, high waves, and storm surges, all of which are phenomena of rising water caused by high-speed winds pushing water towards the coast. Tropical cyclones draw in air from large areas and concentrate the water content into precipitation over a much smaller radius. This phenomena may cause multi-hour or multi-day heavy rains, even as far as 40 km from the coast, exceeding the amount of water that the atmosphere can hold. This can cause both river and overland flooding, as well as overwhelming local water control structures throughout a broad region.

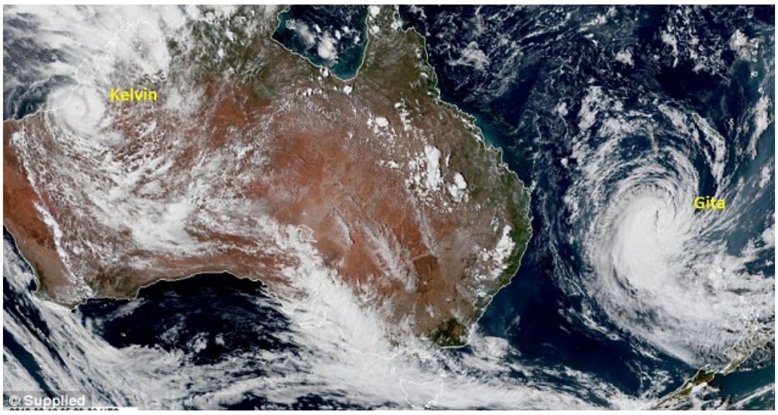


Figure 2.3: Cyclone Gita on New Zeland - Source: https://www.dailymail.co.uk/news/article-5412605/New-Zealand-state-emergency-Cyclone-Gita.html

These tropical storms are low-pressure regions in the troposphere. The pressure is the lowest near the surface, while at the center of these storms sea level pressures are among the lowest ever observed. These systems are called ”warm core” because the environment near their center is warmer than the ambient temperature at all heights. The wind field near the surface of a tropical cyclone involves air rotating rapidly around a center of circulation and also flowing radially inwards. At the periphery of the storm, the air may be nearly calm; however, because of the Earth's rotation, the air possesses non-zero absolute angular momentum. As the air flows radially inwards, it starts rotating cyclonically (counter-clockwise in the Northern Hemisphere, clockwise in the Southern Hemisphere) so as to conserve angular momentum effectively. At a certain distance from the centre of the storm, air starts moving upwards towards the top of the troposphere. This distance is generally the same as the inner radius of the storm's eyewall, which is where the storm's strongest winds occur, also known as the radius of maximum winds. The air, once lifted, moves away from the storm's centre and forms a layer of high clouds called "cirrus clouds". These processes ultimately create a wind field that is almost symmetrical around the storm's centre. Wind speeds are low at the centre, increase moving outwards towards the radius of maximum winds and then decay more gradually with radius.

As for the significance of detecting these events, it is important to note that tropical cyclones are one of the most devastating natural disasters, resulting in billions of pounds in damages and thousands of fatalities each year. Although climate models largely continue to predict a future decrease in the number of global tropical cyclones [WMK+16], it is becoming increasingly important to optimize the accuracy and lead time of predictions of such events. This is necessary to minimize the damage these storms can cause since the same projections indicate an increase in the intensity of the strongest storms and higher rates of rainfall. On the other hand, the inevitable impacts of climate change, such as an increase in sea levels, will also amplify the impact of tropical cyclones. The surge of the storm will be able to reach further inland, causing more damage to coastal communities.

2.3 ClimaX

In [NBK+23], "ClimaX: A Foundation Model for Weather and Climate" the authors introduce a deep learning model for weather and climate science that is both flexible and generalizable. It can be trained using heterogeneous datasets that span different variables. ClimaX extends the Transformer architecture through the use of novel encoding and aggregation blocks that allow effective use of available compute while maintaining general utility. It has been trained using CMIP6 data and can be refined to tackle various weather-related tasks. A crucial benefit of this approach is its efficacy when fine-tuned on any dataset, including ones containing previously unseen atmospheric variables and spatio-temporal scales during the pre-training period.

2.3.1 ClimaX Design

ClimaX is a foundation model designed to be pre-trained on heterogeneous data sources and then fine-tuned to solve various downstream weather and climate problems. The set of climate and weather variables is extremely broad, and predictions may be required for regional or even spatially incomplete data, even at different resolutions. Current CNN-based architectures are not applicable in these scenarios, as they require the input to be perfectly gridded, contain a fixed set of variables, and have a fixed spatial resolution. resolution. Transformer-based architectures, on the other hand, offer much greater flexibility by treating the image-like data as a set of tokens. As a consequence, the backbone architecture chosen is a Vision Transformer to provide greater flexibility. Two significant changes to this model were implemented, and are shown in Figure 2.4. The first change involved variable tokenization, which includes separating each variable into its own channel and tokenizing the input into a sequence of patches. The second change was variable aggregation, introduced to speed up computation by reducing the dimensionality of the input data and to aid in distinguishing between different variables, thereby enhancing attention-based training.

After combining variables, the vision transformer block can produce output tokens that are then processed through a linear prediction head to recreate the original image. During the pre-training phase, a latitude-weighted reconstruction error is used to keep into account the location of the current patch. For fine-tuning, the ClimaX modules can be frozen, allowing for training only on the intended part of the architecture. In fact, often only the final prediction head and variable coding modules need retraining. This model has undergone testing for several downstream tasks, including global and regional forecasting and prediction for unseen climate tasks.

2.3.2 Other weather forecast approaches

As opposed to the ClimaX approach, several other works have been developed to tackle the same problem, each with its own benefits and drawbacks. In this section, the more well known techniques will be briefly discussed, with the aim of providing a comparison with the approach taken in this project.

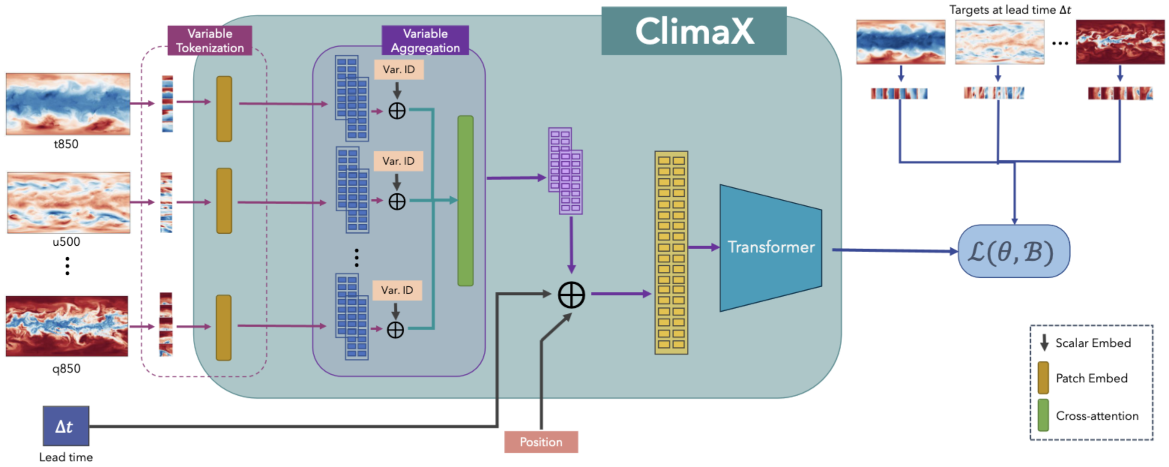


Figure 2.4: ClimaX architecture - Source: ”ClimaX: A foundation model for weather and climate”

Pangu Weather Pangu Weather [BXZ+22] is a transformer architecture trained on three dimensional weather variables, as opposed to Climax, where all data was two dimensional. The lead time is also handled differently, with the model being trained to predict the weather at a certain time in the future, as opposed to the approach taken in the ClimaX work, where the lead time is passed as a parameter during the training phase. The former approach is more similar to the one used in this project, where the simplicity of the dataset allows for a more straightforward implementation of the lead time, sacrificing some flexibility in the process. Finally, the Pangu weather model features some advanced techniques which separate it from all other competitors, namely the use of two different resolutions for the encoding of each variable, allowing the model to capture both large scale and small scale features, and use the attention mechanism to focus on different parts of the input data at the same time. To achieve these two resolution, an encoder-decoder approach is used, where the encoder is tasked with the downscaling of input variables, and the decoder is tasked with the upscaling of the output. All transformer blocks are then applied to the output of the encoder, taking as input both the low and high resolution information.

FourcastNet FourcastNet [PSH+22] is an architecture based on the Adaptive Fourier Neural Operator [GML+21], which is a neural network model designed for high-resolution inputs, fused with a vision transformer backbone. The Fourier Neural Operator is a neural network architecture that uses a Fourier basis to represent the input data, allowing for the efficient computation of convolutions in the Fourier domain. The use of this module allows to have a very small footprint in GPU memory, which is crucial for the training of large models. For instance, the base model used in [PSH+22] is around 10Gb in size, while analogue models with similar number of parameters have a size of around eight times as large.

Graphcast Graphcast [LSGW+23] is a graph neural network architecture with an encoder-decoder configuration. The graph neural network is used to encode unstructured input data into a graph representation. As opposed to, for instance, convolutional layers where neighbouring information is encoded in a structured grid, graph layers use message passing between nodes to capture the relationships between different parts of the input data. This allows for the encoding of different kind of information, not necessarily restricted to a grid configuration.

One important hyperparamter to be set in this kind of architectures is the number of hops the messages containing neighbouring information are allowed to travel. This is crucial for the model to learn from the correct amount of knowledge, and allows for reducing the computational complexity of the model, as the number of hops is directly related to the time required for the model to train.

FuXi In [CZZ+23], an auto-regressive model for weather forecasting is proposed. The model is based on the U-transformer architecture, and is able to recurrently produce a prediction for the next timestep, given the previous predictions. To generate a 15 days forecast, it is estimated it takes the model around 60 iterations, with a lead time of 6 hours. The loss utilized is multi-step, meaning it takes into account several timesteps at once, minimizing the error for each of them. This is in contrast with the approach taken in this project, where the loss is computed for each timestep individually. The U-transformer takes as input around 70 variables, for the current timestep, as well as the the preceding frame. All the variables used for this model are however restricted to two dimensions, ignoring any height layer. This architecture is a variation of the vanilla transformer model, and as opposed to the latter, before passing the encoded information to the self attention blocks, it downscales partially the input.

FenWu In [CHG+23], a cross modal fuser transformer is applied to the problem. This variation of the transformer architecture is able to extract feature embeddings for each variable in an independent manner. Another important novel approach used in this work, is the proposal of a custom uncertainty loss function, which automatically learns the weights for each weather variable. This is crucial in a problem of the magnitude of weather forecasting, as well as for large predictive models, as not all variables learnt are influencing the final prediction in the same way.

MetNet Finally the MetNet [SEH+20] architecture is an LSTM-based neural network, which uses convolutions for downscaling the input variables, and a spatial aggregator module consisting in several self-attention layers. The aim of this model is to be able to forecast precipitation up to 8 hours into the future, using an iterative approach. This means that the model is trained to predict the weather for the next timestep, and then the output is fed back into the model, which is then trained to predict the next frame, and so on. As input, the model takes the current timestep, as well as the previous time frame of data, and is trained to predict the next step in the process. The lead time used for this architecture is 90 minutes, meaning the model takes as input the current timestep `t` and the previous one `t-90`, and outputs the same variable at timestep `t+90`.

2.3.3 Cyclone prediction comparison

In [BFD+22] four distinct algorithms for detecting and tracking cyclones were compared. The study found that on average, the trackers detected cyclones with an 80% success rate. However, the false alarm rates (FARs) differed significantly amongst the four trackers, and in some cases exceeded the number of genuine cyclones identified. Many of these false alerts (FAs) can be classified as extra-tropical cyclones. To address this issue, two commonly used filtering methods can be applied. These post-treatments significantly reduce the false alert rates, which range from 9% to 36% in the final TC track catalogue.

The first algorithm taken into consideration was created by Ullrich and Zarzycki [UZM+21] as a command-line tool that allows for quick and adaptable execution of TC trackers. The TempestExtremes program has two functions:

1. **DetectNodes** identifies "nodes" that correspond to local extreme of a given variable and optionally meet additional criteria.

2. **StitchNodes** connect possible matches within a designated proximity and combine them into a track.

The thresholds were fine-tuned by conducting sensitivity analysis on various metrics and the data from four reanalysis products. The process involves two phases: first locating cyclone points, then connecting them to track their accurate position. To detect candidates, the initial step involves identifying local minima of SLP, It establishes a series of potential points, with only those satisfying two closed-contour standards being kept in the second stage. The first condition confirms that the low-pressure zone is big enough and consistent, while the second criterion confirms the presence of warm air at higher altitudes that is connected to the nearby area of low pressure. For stitching

TC Subsequent potentials are connected if their angular distance is within 8 degrees, only a 24-hour maximum gap is permitted in a track.

The second algorithm is based on evaluating the Obuko-Weiss-Zeta (OWZ) quantity [TDD+13], which is named after its founders and defined according to equation 2.4.

$$OWZ = max(OW_{norm}, 0) \times \eta \times sign(f) \tag{2.4}$$

where η represents the absolute vorticity, which is the sum of the relative vorticity ζ and the Coriolis parameter f, whereas OW_{norm} denotes the Obuko-Weiss parameter normalized.

$$OW_{norm} = \frac{\zeta^2 - (E^2 + F^2)}{\zeta^2} \tag{2.5}$$

in which E and F are the stretching and the shearing deformation.

The initial step of identifying potential candidates involves pinpointing the local maxima of OWZ at 850 hPa. Candidates with a higher OWZ maximum within a 5 degrees are eliminated. Then, TC tracks are stitched. When consecutive TC points are within a maximum distance of 5 degrees, they are stitched together, with a maximum 24-hour gap allowed, in a similar way to the previous algorithm.

The TRACK algorithm [Hod94] is derived from an extra-tropical cyclone tracking algorithm and aims to track all vorticity disturbances. It does not embed a warm core criterion in its initial basic detection; the warm core test is performed only in the last step, independently of the tracking.

Finally, in the CNRM algorithm [CRD06], candidate points are first tracked according to a set of strict criteria, and this detection step is followed by a stitching procedure adapted from [Hod94]. Tracks shorter than one day are then eliminated.

The results for the four trackers are shown in Table 2.1. The POD is defined as the ratio of the number of tracks overlapped by IBTrACS, to the total number of tracks, while the FAR is defined as the number of false alarms divided by the total number of tracks detected by the tracker. Overall, these figures illustrate the trade-off between FAs and misses: improvements in POD tend to come at the cost of an increase in FAR.

Tracker	UZ	OWZ	TRACK	CNRM
POD	74 %	76 %	84%	74%
FAR	9 %	25 %	36%	15%

Table 2.1: Probability of detection (POD) and false alarm rate (FAR) of the four trackers used in this paper with respect to IB-TS. - Source: "Intercomparison of four algorithms for detecting tropical cyclones using ERA5" [BFD+22]

However, these figures are global averages and mask significant regional variability. Further analysis of the results shows that by comparing the first and last dates of detected and observed tracks, the duration of the tracks is homogeneous across all oceanic basins.

In general, TRACK detects the longest TC life cycle: 50% of its tracks start 4 d or more before the first IBTrACS record and end 2 d or more after the last IBTrACS record. On the other hand, three quarters of the OWZ tracks start before IBTrACS, but show a reduced ability to follow a track after its re-curvature.

2.4 Pytorch and Pytorch lightning

PyTorch [PGM+19] is a Python package for deep learning and machine learning. It is a framework that enables the development of large-scale deep learning models and offers GPU acceleration. It provides the fundamental components for building neural networks and a range of standard tools that may be beneficial for establishing and evaluating these designs. PyTorch Lightning [FT19], on the other hand, is a framework constructed on PyTorch that standardizes the development of any model and offers automatic management of the training, validation, and testing stages, without compromising design flexibility. Furthermore, it provides a simple interface for parallelizing the training of large models, offering access to several techniques.

2.4.1 Other libraries and tools

Other Python libraries utilized in this project comprise Timm [Wig19], an abbreviation for Pytorch Image Models. Timm encompasses contemporary computer vision models, layers, and other pivotal tools essential for constructing machine learning structures. The previous sections of this thesis mostly made use of Dask [Das16] for data analysis. Dask enables easy implementation of Python parallelization, which is especially beneficial when used alongside Pandas [pdt20], NumPy [HMVDW$^+$20], Xarrays [HH17] and other mathematical libraries in Python.

On the other hand, some tools were also used to manage file conversion, the main one being Climate Data Operators (CDO) [SKQ19], which was mainly used to convert files in GRIB format to NetCDF4. The former is a self-contained data format, mostly used in meteorology, which offers a better compression ratio for weather data. The latter, on the other hand, is an open standard for array-oriented scientific data, developed with the idea of allowing simple and efficient subsetting.

2.5 Distributed Deep learning

Distributed training is the process of subdividing the training workload of, for example, a large neural network across multiple processors. These processors are often referred to as workers, and they are tasked to compute in parallel to speed up the training process. There are two approaches to parallelism: data and model. In data parallelism, the full training set is divided between all the workers, where a copy of the model is also kept. Training is done either synchronously, where all the workers wait for each other, synchronize the gradients, and only then perform the backward step; or asynchronously, where a selected worker is tasked with keeping an updated version of the weights, and all the others can read and write from this worker, often called a ”parameter server”. Using the latter procedure means that all resources are used at the same time, without any delay. However, it also means that only one worker at a time is training with the latest version of the weights. In large clusters the centralized nature of this approach can also create bottlenecks. Model parallelism, on the other hand, divides the model either horizontally, i.e. node-wise, or vertically, i.e. layer-wise, between several workers who are allowed to run at the same time. This approach also reduces the footprint of the model in each worker, making it lighter on the GPU's memory.

2.5.1 DDP and FSDP

Distributed Data Parallel is a method of data parallelism that enables a program to operate on numerous machines simultaneously. Applications utilizing DDP generate numerous processes and initialize one DDP instance for each process. In the case of machine learning, this consists in generating a neural network copy for each thread, and utilizing the communication methods provided by `torch.distributed` to synchronize gradients and buffers. With PyTorch Lightning, this data parallelism technique is even simpler to enable, and it enables training times to be reduced almost linearly.

In some cases, it may not be possible to create a duplicate of the model for every process. In these instances, Fully Sharded Data Parallel may be utilized, where the optimiser states, gradients, and parameters of the model are subdivided across all DDP ranks. In this case, the neural network is divided into smaller sub-models, each of which represents a portion of the parameters of the overall model. This approach allows different parts of the model to be processed simultaneously by different processing units, and can be used in conjunction with a data-parallel approach that splits the training set to achieve even faster processing times. This results in a program that has less impact on GPU memory, thus reducing execution times.

Due to a problem with the Oak Ridge file system, FSDP cannot be used. However, it is possible to complete this stage using another successful paradigm, called DeepSpeed, which has trained architectures with billions of parameters.

2.5.2 DeepSpeed

DeepSpeed [ARZ+22] is a deep learning optimisation suite that enables efficient scalability and faster execution times for both training and inference of large machine learning models. It was developed by Microsoft and claims to offer a 15x speedup over other state-of-the-art parallelization techniques. It provides memory efficient data parallelism and enables training without model parallelism through a novel solution called Zero Redundancy Optimizer. Unlike basic data parallelism, where memory states are replicated across data-parallel processes, ZeRO partitions model states and gradients to save significant memory. Several other memory optimisation techniques are also used, such as Constant Buffer Optimisation, which combines all communication-based operations into a single operand, and Contiguous Memory Optimisation, which reduces fragmentation during training.

2.6 Data sources

Several high-quality datasets are accessible on the ORNL high-performance storage system. For the specific aim of tropical cyclone prediction, two primary datasets are selected. The first is ERA5, which is preferred due to its wide time window of data that provides ample information on the infrequent but expansive events. The second is the IBTrACS archive, which records tracking data for all previous cyclones.

2.6.1 ERA5

The European Centre for Medium-Range Weather Forecasting's (ECMWF) ERA5 reanalysis archive [MSDAP+21] is the primary data source for learning and benchmarking weather forecasting systems. The ERA5 reanalysis will provide a detailed record of the global atmosphere, land surface and ocean waves since 1950. The currently available ERA5 reanalysis data combines the state of the art ECMWF Integrated Forecast System forecast model with available observations to provide the best estimate of the state of the atmosphere, ocean waves and land surface at any point in time. The integrated forecasting system is utilized to fill in missing data, whether these are patches of information, or entire temporal gaps in weather variables.

Several versions of this dataset are available, differing by the set of variables contained, or by the temporal or spatial resolution. While the full ERA5 dataset version is available on the OLCF storage system, it was opted to work on a collection of variable explicitly curated for deep learning tasks at scale, called WeatherBench [RDS+20]. One of the main advantages of this collection, is the fact that all variables present are curated for medium-range weather forecast in mind, so with a lead time of around three to five days. Since the full resolution ERA5 grid is really large, the dataset was re-gridded to several lower resolutions, and only 13 of the 37 initial height levels are kept. On the OLCF file system, the ones available are the 0.25, 1.40625, 5.625 and 2.8125 degrees versions. Since the 1.40625 version results in images already too small to be able to clearly distinguish the features of tropical cyclones, it was opted for experimenting with the two larger versions of this dataset.

The former consists of over 40 years of hourly data, from 1979 to 2022, on a $0.25° \times 0.25°$ global latitude-longitude grid of the Earth's sphere, with several different climate variables. The grid contains a total of 721×1440 grid points for latitude and longitude, respectively, and 13 levels of height. The latter, on the other hand, consists in the equivalent temporal dimension, but the spatial one has been down-scaled. The resulting image consists in `256`×`128` grid points, as well as the usual 13 layers of height.

2.6.2 Coupled Model Intercomparison Project Phase 6

The Coupled Model Inter-comparison Project (CMIP) [EBM+16] is an international effort that brings together different individual climate modeling groups to compare and evaluate their global climate models. Delving deeper, 49 groups are involved with their experiments covering a wide range of climate variables from as early as 1850. To allow for easier experimentation, the latest data from their experimental runs are easily accessible in the CMIP6 archive, a wide range of climate variables and processes, including temperature, precipitation, atmospheric circulation, ocean currents, sea ice extent, and more. These projects aim to establish a framework for comparing diverse climate models and their simulations in various scenarios. This facilitates researchers to evaluate potential future climate changes. The main goal of CMIP is to improve the understanding of the Earth's climate system, by identifying the areas of agreement and disagreement between models, and the accuracy of its simulations, to have an high confidence forecasting system for future earth climate.

Another important topic CMIP sets to tackle, is to develop Informing IPCC assessments, especially on the matter of climate change. These will contribute to the assessment reports of the Intergovernmental Panel on Climate Change, which provide comprehensive summaries of the state of climate science for policymakers and the public.

2.6.3 International Best Track Archive for Climate Stewardship

IBTrACS [KN10] is a worldwide repository of historical data on tropical cyclones, which records their intensity, tracks, landfall times, and other factors. The database incorporates information from multiple agencies and time basins, providing comprehensive global coverage. The quality of the data is closely monitored to guarantee its consistency, reliability, and adherence to a standardized format across all variables.

This dataset contains several data points including storm position in an oceanic basin, which may be the North or South Atlantic, Eastern North Pacific (including the Central Pacific region), Western North Pacific, South Pacific and South or North Indian. It also includes a temporal dimension to facilitate access to sets of storms, enabling further subdivision of points into well-defined subsets.

The dataset is accessible in two formats, CSV and NetCDF4. CSV is a text file with comma-separated values, while NetCDF4 is binary and includes metadata information about the variables. Although NetCDF4 would be better for the project due to its efficient storage, CSV was chosen for its ease of use and the fact that not all fields are essential. For this reason, and the need to reduce the size of the file, only the latitude, longitude, date and time fields were kept. This is sufficient to reconstruct the trajectory of the cyclone, as well as to extract the position of the storm at a given time from the ERA5 dataset.

3 Implementation

Before embarking on the development of a machine learning model, some preliminary assessments and tasks need to be resolved. In particular, since this specific project will involve multi-phase training, it will be crucial to have several datasets of consistent size, since a basic model requires a large amount of data to be pre-trained. To this end, several resolutions of ERA5 have been used, as well as other auxiliary information such as the position and timestamp of tropical cyclones.

Another problem that frequently arises during the creation of a comprehensive foundation model design is what is commonly known as the critical model size [Bow23]. This refers to the minimum size of the model that can assimilate the characteristics within the dataset, and learn the underlying patterns and representations. This is particularly striking in the case of large language models, where billions of parameters are often required to accurately comprehend the intricacies of the language. Examples of this include GPT-3, with 175 billion parameters, and BERT, with 340 million parameters. It is worth noting that due to the complexity of training such a large architecture, there is now a division between companies that can afford to train these models and those that cannot. This creates what has been referred to as the "Parameter gap" [VSB+22] in the literature, where there seems to be a shortage of machine learning publications featuring models with a parameter count between 20 and 70 billion.

Several could be the motivations for the presence of this gap, although the most likely being the divide between the number of parameters present in expensive models, such as GPT-3, and smaller ones. The former architectures in fact, feature what is termed "Critical model size", which is the minimum number of parameters necessary for the model to start learning all features present in the dataset. This hypothesis is also reinforced by the fact that almost all architectures above the parameter gap are language models, requiring an extremely high number of weights for satisfactory accuracy in text based tasks.

While the critical model size for a model is strongly task dependant, and it is difficult to know as no precise literature studies are present, it is possible to position this threshold at around 100 million parameters for most language understanding tasks. On the other hand, for climate variable prediction it seems a smaller model can be enough, as there are examples, such as Google's Graphcast, achieving competitive results with much smaller architecture size.

3.1 Dataset analysis with Dask

A crucial aspect necessary for the correct learning process of the basic model is to ensure that all the data used for training is normalised. To ensure this, a transformation is applied when each patch is extracted from the dataset, according to equation 3.1.

$$\mathrm{norm}(V) = \frac{V - \mathrm{mean}(V)}{\mathrm{std}(V)} \tag{3.1}$$

Given that a program would have to iterate over the entire dataset to calculate both the mean and the standard deviation, and given the extremely large size of all the datasets used in this project, it is simply not possible to have a sequential script deal with this problem.

However, with access to the resources provided by the Oak Ridge National Laboratory, it is possible to write a multi-node, multi-process program using libraries such as Dask and Horovod to compute only the mean and variance of a subset, and then reduce all outputs to the complete final result.

Looking at the details, since the fast nature of this task is unlikely to require more than a single node, Dask was chosen. This library requires several steps to run on a cluster such as Andes. The first requirement is to activate a conda environment to ensure that all the necessary libraries don't have any compatibility issues. The second step is to create a dask scheduler using the command:

```
>>> dask scheduler --interface ib0
                   --scheduler-file $scheduler_file
```

And then running:

```
>>> srun dask worker --scheduler-file $scheduler_file
                     --interface ib0
                     --nworkers $nworkers
```

This node will be responsible for the scheduling and the execution of tasks within a Dask computation graph. Dask breaks down complex computations into smaller, manageable tasks organized as directed acyclic graphs. The scheduler determines when and where to execute these tasks, considering dependencies and available resources. In this project, multiple computational nodes are present. The scheduler assigns tasks to worker nodes in the cluster, guaranteeing efficient and parallel execution.

Once the former command has completed, it is possible to run the Python source code, with the command:

```
>>> python3 script.py $scheduler_file $SLURM_NTASKS
```

where $scheduler_file is the path to the scheduler configuration file, and $SLURM_NTASKS is the number of workers which are requested by the program. The script has now the functionality to create a Dask client, request an arbitrary number of worker nodes, and schedule functions to run in a distributed manner, all with very minimal code changes.

In the python program, the first step is to create a dask client, which will handle communication between all workers, and execute the `client.wait_for_workers` command, which will register all connected processes.

Once these steps are completed, it is possible to submit distributed functions to each worker, for example, in the case of finding the mean and standard deviation of a dataset, the procedure will calculate these values on a subset of the data. This operation is repeated for all the datasets used in this project, for all the variables used in the foundation model, and for both the full and half resolution variants used in the experiments. The machine used in this case was Andes, as this process does not require any GPU acceleration, and the high memory queue was not necessary as the memory footprint of the program is extremely small. A run with one node and four tasks was sufficient to complete the task in about 10 minutes, for one variable and one resolution. Since each node in Andes has 32 CPUs, this means that the program was able to use 128 CPUs in parallel, which is a significant improvement over a sequential script.

3.2 Datasets

Four datasets were created for this project. The ERA5 dataset is used for both pre-training and fine-tuning phases, and holds most of the data used by the model. The Mixed, Sequence, and Ibtracs datasets, on the other hand, are only used for fine-tuning or testing the pre-trained model's forecasting accuracy.

3.2.1 ERA5 Dataset

The first data set implemented is based on ERA5, which was already available on the Oak Ridge High Performance Storage System in a weather benchmark collection. This set contains several weather variables such as vorticity, humidity, temperature, geopotential, wind speed, and mean sea level pressure. Considering that the final task of this project is the prediction of tropical cyclones, only the variables that influence their formation are needed. After a primary selection process, those considered to be influential for this type of large scale events are: humidity, temperature, wind speed and mean sea level pressure.

The dataset consists of hourly data, divided into separate files according to the year of collection. Each file then consists of `1440×720` images and 12 elevation levels. The Pytorch Dataset class iterates over all the files, i.e. annual data, over all the time steps in each file, totalling 8760 hours in a year, and divides each `1440×720×12` into `32×32×4` patches, selecting the height levels closer to the ground.

An important exception is the mean sea level pressure variable, which, unlike all the others, is not three-dimensional, but simply a flat image. In this case the approach taken was to make it a three dimensional variable by repeating the same values on the height axis. This is a simpler approach than ensuring that the variable embedding and aggregation systems standardize the size of the latent space, but will inevitably lead to more repeated data being processed.

As for the design decisions, it is important to note some inconsistencies in this collection. The first one is the difference in variable dimensionality, as mean sea level pressure is a two-dimensional variable, while all others include a height dimension. This is solved by repeating the same values on the height axis, as previously mentioned. The second inconsistency is the difference in naming schemes for datasets at different resolution. One example of this is the height dimension, which in the full resolution version is called ”lev”, while in the half resolution version it is called ”plev”. This is solved by using a dictionary to map the names of the variables to the correct ones for each resolution.

3.2.2 IBTrACS

The IBTrACS dataset was first downloaded in CSV format from the NOAA website[1]. As the only component required for this project is the time stamp (`ISO_TIME`) and the latitude and longitude of the cyclone, all other fields in the file were removed. These included the basin and sub-basin information, the type of climate, wind information, and tracking information. By cycling through all the elements, the dataset looks through all the elements in the IBTrACS file and opens the corresponding ERA5 file, taking the correct image patch containing the cyclone. This patch is image-aligned and not centered on the center of the storm, to avoid the model always predicting maximum probability in the middle of the patch. Once the input patch has been extracted, the ground truth `32×32` image is created. This patch contains the distance between all cells and the center of the cyclone, up to a maximum distance of 16. To ensure that the model learns the probability of the presence of the storm, this distance is then inverted and normalized between 0 and 1. Although this dataset has only been used in conjunction with the full-scale ERA5, its adaptability allows it to be used with the half-scale version for fine-tuning in other applications, such as long-range cyclone tracking.

3.2.3 Mixed

The Mixed dataset class is simply used as a merged dataset that randomly samples either a cyclone patch from an IBTrACS dataset or a non-cyclone patch from an ERA5 dataset. The latter also includes an option to filter out cyclone patches, as this can lead to a data imbalance problem where the model predicts too many cyclones because they are extremely common in the dataset it was fine-tuned on. This mechanism works by checking if the current timestep is present in the IBTrACS dataset, and if so, it skips the current iteration and moves on to the next one. As a result, it is possible to select accurately the amount of cyclone patches to be included in the dataset, and to avoid the problem of data imbalance. All experiments have been run with a ratio of cyclones per normal patch of one to 30, which is to be considered still a very high ratio, as cyclones are extremely rare events. It is however not possible to reduce this ratio further, as the dataset would not have enough cyclone patches to be able to train the model correctly.

3.2.4 Sequence

The final dataset created is a variation of the initial ERA5 one, where the input matrix is kept the same, while the output is a sequence of matrices corresponding to the next N timesteps of the same patch. Although ERA5 contains hourly data, it is also possible to specify, through a parameter, the number of hours to skip between each item in the sequence.

This version of dataset, differently from all others, is not used for pre-train or fine-tuning purposes, but for judging the model's whether prediction capabilities, by testing each item in the target sequence against a prediction made from the model, which results from its output at the previous timestep.

The dataset takes the initial input variable and the corresponding target information, and appends to the latter the next N timesteps of target variables. This allows for the comparison between the output of the model at timestep 0, generated using the input data, and the target information (Gt_t). Once this first comparison is done, the process can be repeated using the output at timestep 0 as

[1] https://www.ncei.noaa.gov/products/international-best-track-archive

input and checking the result with the target at timestep 1. The resulting loss is shown in equation 3.2.

$$\begin{aligned} Out_t &= \text{Model}(Out_{t-1}) \\ \text{Loss}(t) &= \text{MSE}(Out_t, Gt_t) \end{aligned} \tag{3.2}$$

3.2.5 Dimensionality of input data

For all four types of dataset, the input is essentially the same, with the exception of the number of variables V used. Each input patch is a five dimensional vector `B, V, H, Lat, Lon` where B is the batch size, V is the number of variables, H is the amount of height layers, and Lat and Lon are the patch dimension. In most experiments, the dimensions used are reported in Table 3.1.

Dimension	Size
B	16
V	3, 4 or 5
H	4
Lat	32
Lon	32

Table 3.1: Input dimensionality for all experiments run in this project.

It has to be noted that, when dealing with a distributed training system, the ideal performance is obtained with a batch size of 1. In this project it was decided to keep it at 16 as it is theoretically easier for the model to converge, as well as it helps when testing the scalability of the training, as with less workers it is still beneficial to have a large batch size.

When using these measurements, the final input dimension corresponds to `16×4×4×32×32` yielding 262.000 floats or 1.048.000 bytes sent to the GPU for each batch. Several experiments have also been run using the high-memory queue, where it was possible to increase the batch size to 32 without running into errors due to not enough RAM available.

As for the target dimensionality, this depends on the type of dataset. For example, in the ERA5 dataset it is equivalent to the input, on the other hand in the Sequence dataset, this one is N times the size of the input, where N corresponds to the number of sequence items used for the prediction. As N has been consistently set to 12 in most experiments, the sequence dataset generates an input of standard size, which consumes over a megabyte of memory. Moreover, it results in a target list that adds another twelve megabytes of data.

As all pre-training phases utilise the Latitude-Weighted Mean Squared Error, an additional element is required in the form of a two-dimensional array that normalises the encoding of the latitude coordinates for the current patch. As a single patch contains three or more variables, which are restricted to the same portion of the globe, a flat array is sufficient to encode this information. The same tensor can then be used for all loss calculations. Nonetheless, this additional component results in another `32×32` array of floating points, resulting in an additional 4096 bytes for each sample.

Finally for the IBTrACS dataset, the required target is a simple `<Lat×Lon>` probability map, where dimensionality is almost always `32×32`.

To make matters worse, the model size tends to be very large, with parameter counts in the range of 100 million for the more common experiments and even 600 million for the larger ones. This means that between 1.5GB and 10GB of GPU RAM is taken up by the model, leaving little space for loading the dataset.

3.3 Foundation model design

The foundational model developed in this project draws heavily from the one presented in the work [NBK+23] and comprises four distinct components. Firstly, there is variable encoding, which involves transforming input data into a latent space for transformer blocks to leverage. Secondly, there is variable aggregation, which reduces the dimensionality of input data to minimize both computational cost and GPU usage. The third component comprises transformer blocks responsible for capturing connections between distinct data patches. Lastly, the prediction head restores the original data from the transformer block outputs. Two different types of prediction heads have been employed: one for reconstruction, which aims to reconstruct the original input, and another for prediction, which attempts to anticipate whether the input has cyclone-related features or is closely related to them.

On the topic of variations, the reference model has been streamlined by removing the Latitude-weighted reconstruction loss, deemed superfluous for this project. Instead, it has been substituted with a straightforward Mean Squared Error loss. Additionally, the original model utilised a complex technique for merging the patches into a single image. This aspect has been simplified because it is believed that the model can learn this transformation independently without using a complicated up-sampling and concatenation system. Additionally, the input used in this project consists of three-dimensional patches, so the model has been adjusted to accept this type of input. The original model was designed for two-dimensional images. This involved redesigning the patch embedding system to accommodate the height dimension using a 3D convolutional layer.

Similarly to the ClimaX foundation model, this architecture can forecast input variables for a specific lead time. The key difference between the two approaches is their method of encoding into the model. In this approach, the architecture solely learns to predict variables with the chosen lead time. Therefore, a pretrained model that has learned to forecast with a 6-hour lead time cannot predict an hour ahead. This is achieved by comparing the model's output to the target variable taken at the appropriate time interval.

3.3.1 Variable Encoding

The initial aspect of the fundamental model is variable encoding, which converts the input data into an alike latent space that can be utilised by the transformer blocks. To delve deeper into this stage, the input data initially undergoes a convolutional layer that is utilised to cut down the number of channels to a specified amount. The said convolutional layer is designated in an ad-hoc module referred to as the PatchEmbed3D, which is a modest wrapper encircling the Pytorch Conv3D module. This is the first distinction between this model and the one presented in ClimaX, as the original model was intended for two-dimensional images, while this one is intended for three-dimensional patches. To ensure specialisation in the embedding space of the model, one PatchEmbed3D module is utilised for each variable in the input data. This implies that the number of channels in the output of this module corresponds to the number of variables in the input. This ensures that every module only learns the unique characteristics pertaining to their respective variable and allows the model to attain the optimum representation for each one. The input for this phase is divided for each variable, sent through the respective PatchEmbed3D module, then concatenated along the channel axis. The ultimate shape comes out to be `<B, V, P, E>`, where B represents the batch size, V denotes the number of variables, P is the total number of patches, and E stands for embedding size. When passing through the patch embedding module, the input undergoes patch division by a convolutional layer with a kernel size and stride of 2, effectively reducing the input size by half. The use of 2 as both stride and kernel size is pivotal since it ensures the division of the input into non-overlapping patches while allowing for the preservation of the power-of-2 dimension.

The second component of the foundational model is variable aggregation, which aims to decrease the input data's dimensionality and reduce both computational costs and GPU usage. The first step involves appending a new parameter named variable encoding to the current representation, which incorporates the patch embeddings. This process is achieved via several measures in succession. Variable encoding is a straightforward `<V, E>` vector, where V denotes the number of variables in the input data and E denotes the embedding dimension. This parameter serves as a bias term and is essential for the model to learn the optimal representation for each variable. The second step involves interchanging the patch dimension P and the variable dimension V, which is a preliminary step to flatten the first two dimensions. After this stage, the input takes shape `<B×P, V, E>`, where B represents the batch size, P represents the number of patches, V represents the variable count, and E represents the embedding size. This sets the foundation for the subsequent step, which involves applying an initial Multi-Head attention layer. Although this layer is not yet part of the transformer blocks, it is implemented to enable the model to learn the best representation for each variable. The query generated for this attention block is derived by replicating a different parameter, with a shape of `<1, 1, E>`, as many times as the current representation's size. However, the key and values applied are the current representation itself.

The outcome vector gains a shape of `<B×V, 1, E>` and is then flattened to a `B×V, E` shape. It is then un-flattened to form a `<B, L, E>` shape, where L denotes the fixed size latent space dimension. Finally, an additional parameter known as positional embedding is incorporated into the existing representation, followed by the passage of the vector through a dropout layer to restrict over-fitting.

3.3.2 Transformer blocks

The third component of the foundational model is the transformer blocks, which are responsible for capturing connections between distinct data patches. During the pre-training phase, the model learns the relationships between variables contained in each patch, and identifies the best performing latent space which allows for a correct reconstruction of the initial input.

The library employed during the development phase of this project is timm, which offers a diverse array of pre-fabricated transformer architectures, including the Vision Transformer employed. Each transformer block comprises a custom attention layer, a feed-forward layer, LayerScale blocks (comprising a Parameter layer multiplied by the current input), DropPath blocks (which act like residual blocks, skipping the connection depending on a given probability), and an MLP block that passes through either a convolutional or linear layer twice, along with a dropout layer. The input is initially processed by the attention layer which consists of a 16 head multi-head attention block, the LayerScale and the DropPath block. This procedure is subsequently replicated for the second segment of the transformer block, comprising of the MLP layer, a LayerScale block and a DropPath block. This process is replicated for the second stage of the transformer block, comprising the MLP layer, a LayerScale block, and a DropPath block. The outcome of this second stage is then combined with the output of the first stage, resulting in the final product of this phase. Finally, the transformer block's output is passed through a layer normalization layer to normalize the tensor on a layer-wise basis. This step is crucial as it prevents both gradient explosion after passing data through multiple computational blocks and internal covariate shift, which is the phenomenon of the input data distribution changing between layers.

3.3.3 Prediction head

The foundational model's last part is the prediction head, which restores the original data from the transformer block outputs. The prediction head constitutes a simple linear layer that reshapes the input from embedding dimension size to the original input size. In pre-training, the prediction head is tasked with restoring the original input data, which in this case is `<V, H, Lat, Lon>`. Once this vector is obtained, which will have a shape of `<B, L, V×H×Lat×Lon>`, it undergoes an intermediate reshaping operation to facilitate the concatenation of patches into the original image. This is accomplished by reshaping the vector to `<B, H, Lat, Lon, p, p, p, V>`, after which it is flattened to `<B, V, H×p, Lat×p, Lon×p>`, where p indicates the patch size of the embedding module.

3.4 Pre-training

During pre-training, the model is trained to reconstruct the input variables shifted by a specific amount of lead time. This establishes that the architecture learns the accurate relationship between diverse inputs and allows for a direct comparison between the model's output and the ground truth. The models have been trained utilizing early stopping with a 5-epoch patience and a maximum of 100 epochs to avoid overfitting. Two different types of loss function have been used, the first tests were carried out with a simple Mean Squared Error loss, checking the difference between the model output and the ground truth, as shown in equation 3.3. However, this process is flawed when the images are taken from a spherical projection, as the points on a sphere are not evenly distributed. An improved approach is then adopted, using Latitude-Weighted Mean Squared Error, which weights the loss according to the latitude of the current pixel. This is done to ensure that the model is not biased towards the equator, where points are more densely packed, and to ensure that the model learns the correct representation of variables in all parts of the globe.

$$\text{Loss} = \frac{1}{H \times \text{Lat} \times \text{Lon}} \sum_{h=0}^{H} \sum_{la=0}^{Lat} \sum_{lo=0}^{Lon} L(la)(\hat{X}_{t+\delta t}^{h,la,lo} - X_{t+\delta t}^{h,la,lo})^2 \tag{3.3}$$

It has to be noted that, in cases of small patches such as this one, the difference between the two loss functions is minimal, as the points are not too far apart from each other. The latter approach is still however preferred, as it is considered a more correct approach to the problem.

As for the optimiser, Adam [KB14] is used for all experiments, with a learning rate that is adjusted according to the model size, the experiment, and whether pre-training or fine-tuning is performed. It was decided to use Adam instead of AdamW because the convergence times for this model are already particularly slow and the built-in learning rate scheduler, which is used to progressively reduce this hyper-parameter during training, could have further increased the number of training epochs required.

3.4.1 Pre-training Evaluation Prediction head

In an effort to evaluate the pre-tranined model's performance in a way to isolate the forecasting ability of a single variable, another prediction head has been developed. This block is used to convert the transformer's output into a `<B, H, Lat, Lon>` tensor, where the only variable trained to be predicted is temperature. Once the tensor is obtained, the target temperature channel from the ERA5 dataset is compared with the predicted one, using a Mean Squared Error loss function. This allows for a direct comparison between the model's prediction and the ground truth, and can be used to evaluate the model's performance in a way that is independent from all other variables.

This is an analogous analysis to the one conducted in [NBK$^+$23]. In this case, the losses for four distinct variables were isolated: geopotential at 500hPa, temperature at 850hPa, temperature at 2 metres from the ground, and zonal wind speed at 10 metres from the ground. It was then tested how the error increased on medium to long-range prediction, from one day up to a month of lead time.

Although the temperature variable is identical to that used for pre-training and evaluation in this project, the differing lead time prediction scope makes comparison challenging. However, it can be noted that the calculated mean squared error is significantly lower than the one reported in the ClimaX paper, as the shorter lead time makes forecasting much simpler.

3.5 Fine Tuning

Moving to the fine-tuning phase, the prediction head is replaced with a new one that fulfils a new distinct purpose. For this project, the objective is to forecast if the input patch holds characteristics related to cyclones or not. As the transformer model is pre-trained to forecast the input variables displaced by a duration of time, the fine-tuning head should predict future cyclone occurrences with respect to the present input image. The process of swapping the prediction head involves freezing the gradients for all transformer blocks, along with the variable encoding and aggregation, to prevent the model from forgetting the previously acquired knowledge during pre-training and to avoid catastrophic forgetting. Performing this operation before attaching the new head to the model enables loading of all weights from the pre-trained mode. Once the pre-training weights have been loaded, replace the prediction head by swapping the last linear layer with a new one. The new layer comprises a simple linear layer with a GELU [HG23] activation function, which outputs a `32×32` tensor representing the probability of a cyclone's existence for each pixel.

The difference between a classic ReLU activation function and a GELU one is also shown in Figure 3.1, where the latter has a smoother transition between positive and negative values, and seems to perform slightly better on larger architectures, and is often preferred in models such as LLMs and transformers.

One crucial concern with this approach is the reduction of dimensionality in the latent space within the transformer blocks. To elaborate on this issue, several introductory statements are necessary. Primarily, the model is pre-trained to forecast the input variables, producing a significant amount of data as output. This may pose a challenge as the architecture may find it difficult to differentiate which information is essential for our task. On the contrary, the forecasting components decrease the latent space's size quickly, producing a much tinier output with shape `<B, E, 2>`, where only the last

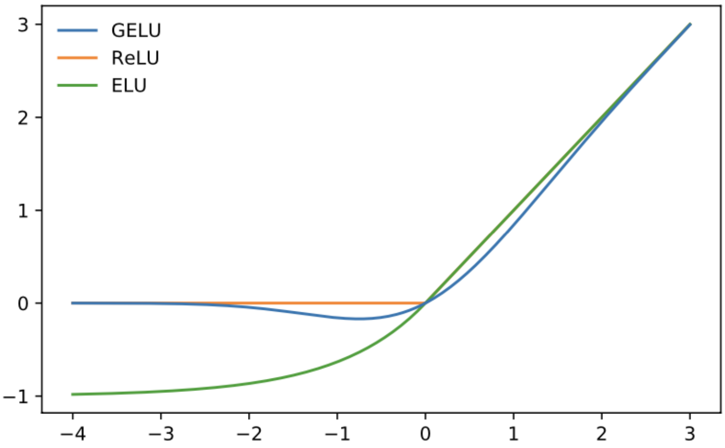


Figure 3.1: Differences between activation functions - Source: "Gaussian Error Linear Units (GELUS)"

dimension is modifiable by the linear layers. This causes not enough information to be used for the final reshape, resulting in slow and ineffective learning. The issue was solved by reshaping the output of the previous linear layer to a `<B, E×2>` shape. The reshaped output is then passed to a secondary prediction head which fine-tunes it further. These last two layers consist of a linear layer with a GELU activation function, used to output a tensor of size `32×32`, indicating the probability of a cyclone's presence for each pixel.

$$\text{Loss} = \frac{1}{W \times H} \sum_{w=0}^{W} \sum_{h=0}^{H} (\hat{X}_{t+\delta t}^{w,h} - X_{t+\delta t}^{w,h})^2 \tag{3.4}$$

The loss used during this phase is a simple Mean Squared Error, checking the per-pixel difference between the ground truth probability map and the output of the model, as shown in equation 3.4.

3.5.1 Variable encoding re-training and full retraining

During the fine-tuning phase, it is important to prevent the model from forgetting the knowledge obtained during pre-training. To achieve this, the gradients for all blocks that are not part of the new prediction head should be frozen. This ensures that catastrophic forgetting is avoided, while also allowing the prediction head to learn the optimal transformation from the transformer's latent space to the final output. This method is beneficial for many foundational model applications, enabling the model to be adjusted to a new dataset while retaining prior knowledge. However in select cases, such as when the new dataset's variables differ in resolution from the pre-training ones, it may be required to retrain the variable encoding and aggregation blocks. This is applicable for certain experiments in this project, wherein the model is pre-trained on the half-resolution rendition of ERA5, and subsequently fine-tuned on the full-resolution version. In said situation, the variable encoding and aggregation blocks are also fine-tuned to allow the model to learn encoding of new variables at higher resolutions. While generally unnecessary, fine-tuning these blocks can sometimes slightly improve performance by allowing the model to better learn new variables' representations.

In situations where the fine-tuning stage requires additional variables that are not already included in the pre-trained model, a complete fine-tuning of the model is necessary. This involves loading all weights from the pre-trained model and then beginning the training process on the smaller dataset containing the new variables without freezing any gradients. This process is expected to result in poorer performance for tasks that require the newly-trained variables since the finetuning dataset is often smaller and less reflective of real-world situations.

3.6 Differences between ClimaX and current implementation

While the ClimaX model was used as a valid reference for the aim of this project, several differences distinguish the two works. These are both due to the purpose of this model, which is not analogous to ClimaX's, and to the development decision and time available. Following are some of the main differences between the ClimaX implementation and this project:

- The first difference is how the lead time is handled. In the ClimaX architecture it is encoded during pre-training, so essentially creating a lead-time-conditioned model, and it is passed as a parameter to the model during the training phase, whereas in this project it is handled by the dataset class, which returns the correct lead time for each patch. This results in a more flexible training process for the ClimaX model, which is able to learn how different lead times affect the prediction, whereas the dataset class approach is more mono-dimensional, as it only allows a single lead time to be used for each training run. However, it theoretically allows for better performance as the model can focus on a single modality without taking into account external factors such as lead time.
- The second difference is the method of variable embedding. In the ClimaX paper, the authors use an approach that allows a variable to be embedded in a latent space, which is then concatenated with the patch embedding. This is done to ensure the same dimensionality for each type of variable given as input, but it can cause the model to learn a representation that is biased due to over-compression of the data. In this project, on the other hand, all variables are three-dimensional, which simplifies the embedding process, although the patchification layer has to be modified to account for the different number of channels. In particular, since the ClimaX latent space was two-dimensional, a simple 2D convolutional layer could be used, whereas this project requires a 3D convolutional layer.
- The third difference is in the dataset resolution used. In the ClimaX paper, the authors used a resolution of 1.40625 and 5.625 degrees in order to favour a larger scope for fine-tuning tasks, whereas in this project the full resolution of 0.25625 degrees was used. This allows for a more detailed representation of the data, which can be beneficial for the model to learn more complex patterns, such as the large scale events that this project revolves around. However, it also means that the model will require more computational resources to train, as the number of patches will be much larger.
- A notable distinction of the ClimaX implementation is the magnitude of the model created. The computational potential of Summit, allows for greater scope in model size expansion in terms of depth (i.e. layer count) and height (i.e. neuron count per layer) compared to Microsoft architecture, with 100 million parameters. Although most of the experiments were conducted on an architectural scale similar to Climax, some tests have been carried out using a 600 million parameter model, which already challenges better Summit's capabilities.
- The final difference with ClimaX is the range of forecasts for which the model was trained. In the ClimaX paper, the authors focused on long-term forecasts with lead times of up to 30 days. In this project, however, the focus is on short-term forecasts with lead times of 1 and 6 hours, although longer forecasts are possible by iterating over the same process. This is due to the fact that the intentions of the two projects are different, with the ClimaX paper focusing on climate forecasting, whereas this project focuses on tropical cyclone forecasting.

4 Experiments

Several modalities of experiments have been tested, both for the pre-training and fine tuning phases. This section will delve into the differences, difficulties and expected results of each.

4.1 Pre-training

The pre-training of this model was executed with several modalities, which differ in:

- Lead time of prediction, meaning how much time passes between the input and the output. The modalities utilized are one and six hours of forecast time;
- Variables used, since these may have a accuracy effect in forecasting tropical cyclones;
- Dataset resolution, full scale ERA5 and 1.4 degrees;
- Architecture size, both in size and depth of the transformer blocks. The sizes chosen are 1024 and 2048 for the hidden width, and 8 and 10 layers for the depth.

These modalities each target a specific purpose in this project, with the architecture size being used to test the parallelization performance, both from a node stand point (as the more workers are used the faster the training will be) and from an algorithmic point of view, as using a simple DDP approach should be theoretically slower than a very advanced model parallelization technique such as DeepSpeed. On the other hand, the difference in lead time and variables used has purpose of understanding the best modalities for tropical cyclone forecasting, as well as the differences in accuracy when predicting on a larger time window.

4.1.1 Different Lead times

Two different lead times have been tested, the first with one hour between input and target variables, the other with 6 hours. The intent of these experiments are determining what is the difference in prediction when a larger lead time is used. What is expected is a visible reduction in the detail present in the 6 hours prediction, while a very accurate forecast is to be expected in the other situation. Differently to ClimaX, where the lead time is passed as an input parameter, the model presented in this work is pre-trained with a input and output always at a specific distance in time. The former method is extremely more flexible, but it requires larger train times and considering the short time available for the development of this thesis, it was opted for the second option.

The ClimaX architecture was also tested with larger lead times, from six hours all the way up to a month. To reach lead times this large, the solution implemented in this project was to iterate over the prediction at time t, using it as input for the next forecast (at time $t+1$). This approach is explained in depth in Section 4.2

4.1.2 Different variables

Another variation in the experiments comes from the type of variables used for pre-training and the subsequent fine-tuning phase.

The ones considered for this project are

- Temperature: instantaneous temperature in Kelvin on 12 levels through the atmosphere;
- Humidity: mass of water vapour per kilogram of humid air. This variable is calculated as the sum of dry air, water vapour, cloud liquid, cloud ice, rain and falling snow in the atmosphere;
- u and v Wind speed: instantaneous horizontal wind speed of air moving east and north respectively, in metres per second. A negative sign indicates air moving in the opposite direction;

- Mean Sea Level Pressure: the pressure (force per unit area) of the atmosphere at mean sea level. The unit of measurement is Pascal or $hPa(=100Pa)$. It is a measure of the weight that all the air in a column vertically above the area of the Earth's surface at that point would have if the point were at mean sea level;

- Vorticity: a measure of the rotation of the air in the horizontal plane, about a vertical axis, relative to a fixed point on the Earth's surface. Adding the rotation of the Earth, called the Coriolis parameter, to the relative vorticity gives the analogous variable in the absolute form.

The combinations of these variables used are mainly three.

1. Temperature, Humidity, Wind Components;
2. Temperature, Humidity, Vorticity;
3. Temperature, Humidity, Wind Components, Mean Sea Level Pressure;

Most of the experiments which have been run feature the first configuration of variables, the reason is mean sea level pressure and vorticity were added later to the dataset and some pre-training phases were already concluded.

4.1.3 Pre-train and fine-tune on different datasets

The final variation comes from the resolution chosen for fine-tuning and pre-training of the model. In particular, it was decided to test, in addition to both phases being run on the full resolution ERA5 dataset, whether the model was able to learn the underlying representation present in a low resolution image, such as the 1.4 degrees version of ERA5, and transfer it to a high resolution image for a task such as cyclone prediction. This method is a highly desirable benefit of foundation models, which have the ability to adjust their expertise to changing circumstances where a clear link between the pre-training and fine-tuning data sets does not exist. Practical examples of this ability are common, ranging from object recognition, where a man-made or enhanced version of an item is used for pre-training, to any other scenario that calls for an input set to be artificially generated. In such cases, a real-life, noisy setting is used for fine-tuning and testing, particularly when the testing dataset's distribution is not aligned with the training one.

4.1.4 All Combinations

Now that all differences in modality have been explained, it is possible to list what combinations have been selected. These are presented in Table 4.1.

Modality	Lead Time (hrs)	Resolution	Model Depth	Model Width
t,h,u,v	1	Full	8	1024
t,h,u,v	6	Full	8	1024
t,h,u,v	1	Full	10	2048
t,h,u,v	6	Full	10	2048
t,u,v	1	Full	8	1024
t,h,u,v	1	Half	8	1024
t,h,vo	1	Full	8	1024
t,h,u,v,msl	1	Full	8	1024

Table 4.1: Experiment modalities. The first column indicates which variables are taken into consideration. `t` is temperature, `h` is humidity, `u` and `v` are wind speed components, `vo` is vorticity and `msl` is mean sea level pressure.

In particular, two combinations of one and six lead time trainings have been chosen, one with model size equal to ClimaX, the other with a twice the width and increased depth. Additional experiments have also been run, the first to test whether the model was able to learn dependencies in variables in a low resolution dataset and apply it to a higher resolution one during finetuning, the other to test whether the information present in the humidity variable was necessary for the correct forecasting of tropical cyclones.

4.1.5 Scaling over the ClimaX model size

A difference between this work and the one presented in the ClimaX paper is how the transformer encoding section has been structured. As explained in Section 2.3.1, the latter architecture is composed of a variable encoding and aggregation module, followed by several transformer blocks. In the ClimaX definitive model the number of layers used (defined as ”transformer depth”) is eight, while each layer is 1024 neurons wide, resulting in a model size of around 100 million parameters. Since the computational capabilities of Summit allow for training of models with parameters orders of magnitude larger, it was decided to test both the parallelization capabilities of pytorch lightning, with algorithms such as DDP and DeepSpeed, and the improvement in performance resulting from the utilization of a larger architecture. The former analysis is reported in Section 5.6, while results for the latter are presented in Section 5.2.

Although a larger architecture on two modalities has been tested, there are several issues that prevent further development on this topic. Primarily, the larger architecture necessitates more training time in comparison to the smaller one, as well as a more diverse dataset. Given the extensive training times required, with 300,000 samples during the pre-training phase, this considerably impedes the ability to carry out numerous experiments. To compound matters, although Summit's scaling capability would, in theory, allow for the use of more nodes to speed up the process, progress was hindered by lengthy queue times. These caused processes to wait for more than a day and resulted in a significant slowdown.

4.2 Iterative forecasting

As validation for the pre-training process, one possible way to visually determine whether the model is learning the correct effect of each variable is passing the initial input components through the network, and utilize the newly obtained prediction as input for the next iteration. This iterative process, illustrated in Fig.4.1, allows a comparison between each prediction and the target at the corresponding time instance. The mean squared error, in this case, should increase over time, as small errors in prediction eventually add up.

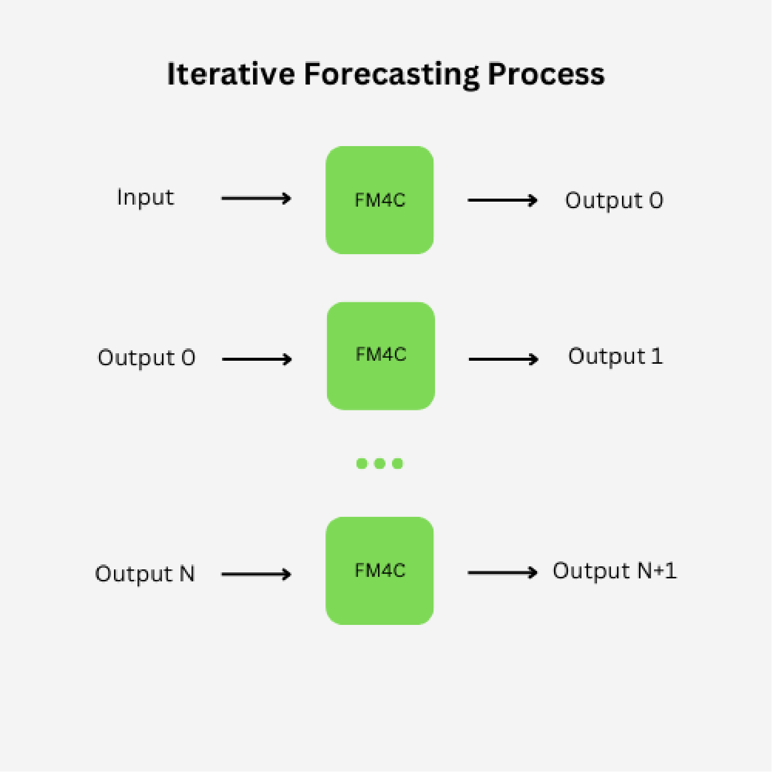


Figure 4.1: Diagram representing the Iterative Forecasting Process

This technique for verifying the model's forecasting abilities cannot be compared directly with the approach outlined in the ClimaX article, where the authors employ an alternative method. They train the model to predict input variables shifted by a particular lead time, and determine the loss based on the model's output and the ground truth. While this approach is valid, and it has been reproduced for the current model in Section 5.3, it fails include information about the repeated prediction process, and how the error accumulates over time.

The iterative benchmark described in this chapter enables a comprehensive analysis of error behaviour, including the isolation of all variables. This guarantees the model's understanding of each set of information independently, even when taking into account very large lead times.

5 Results

This chapter presents and discusses the results of the experiments described in the previous chapter. For commodity, the results are divided into several sections, each corresponding to a different experiment, from transfer learning, to the effect of longer lead times, to training with different variables.

5.1 Visual resemblance

The initial assessment is to visually compare the model output and the target. This is vital to understand whether the model can accurately learn the correct representation of variables and reconstruct the input data. The model used in this case is the same size as the one presented in ClimaX. It is trained on the full-resolution version of ERA5, and the target is derived from the same dataset. The figures 5.1 display the input temperature values on the left, and the model's output in the second picture.

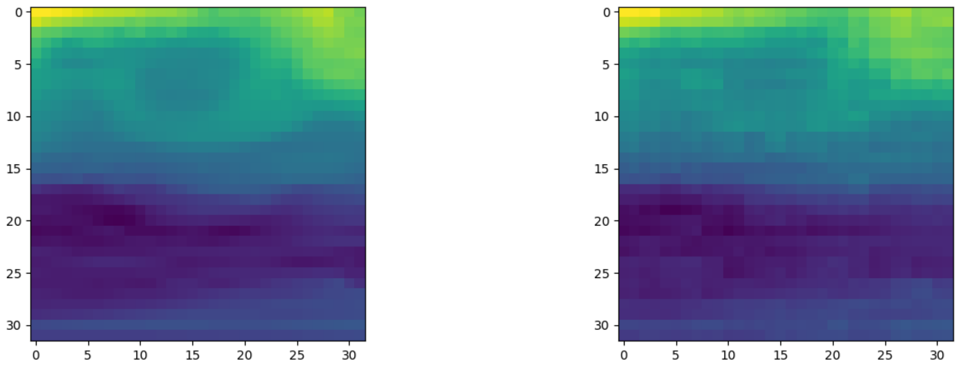


Figure 5.1: Examples of pre-training input (on the left) and reconstruction (on the right).

This initial visualization displays the input and output of the pre-training phase. The similarity between the two images suggests that the model has learned a favourable encoding accurately.

For the second task, in Figure 5.2, the model is fine-tuned on the full-resolution version of ERA5. The input temperature, target probability density function in two dimensions, and output probability from the model are displayed from left to right.

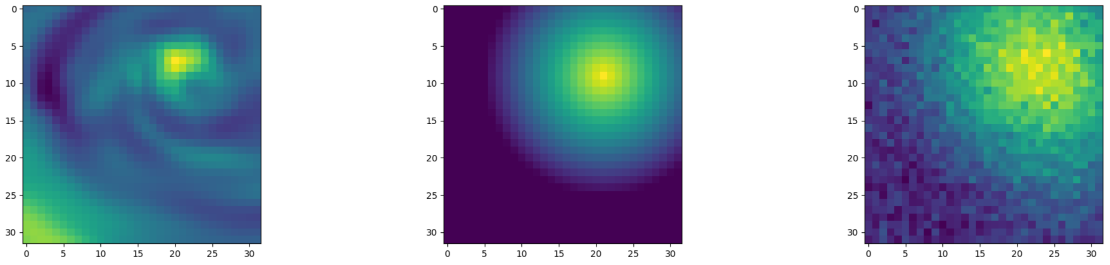


Figure 5.2: Examples of fine-tuning input (on the left), target (in the center) and guess (on the right).

For this stage, the input is a four-dimension vector, with only the atmosphere temperature at ground level (i.e. the lowest available layer) later visualized. The target and output are represented in a simple two-dimensional array, fully displayed in the image.

As reported in the last two images, significant attention was given to ensuring that the cyclone in the patch is offset and not always centred. This simplifies the task for the model, and a prediction would lose significance otherwise.

5.2 Results for pre-training and fine-tuning

Several experimental modalities were run to determine which variables had the greatest impact on the predictive ability of the model. The ones reported in the table 5.1 include two different lead times and three combinations of variables.

Lead-Time	Temp.	Humidity	Wind Components	Vorticity	Mean Sea Level Pressure	Error
1	✓	✓	✓			0.0010 LWMSE
6	✓	✓	✓			0.0121 LWMSE
1	✓	✓	✓	✓	✓	0.0036 LWMSE
6	✓	✓	✓	✓	✓	0.0158 LWMSE
1	✓	✓		✓		0.0041 LWMSE
6	✓	✓		✓		0.0153 LWMSE

Table 5.1: Results when pre-training on different modalities of variables.

The first thing to note is how increasing the lead time directly affects the Latitude Weighted Mean Squared Error. As reported, this metric is consistently one order of magnitude larger when the six hour lead time is used. As for the combination of variables, it seems that using vorticity instead of the wind components significantly reduces the accuracy of the model, while using all available variables gives intermediate results, but this could be due to some data not being correctly represented, perhaps requiring more training. It should also be noted that using a larger model in the latter case could also improve performance, as the architecture may lack the latent space dimensionality to encode all the information correctly.

5.2.1 Losses

Figure 5.3 shows the losses for pre-training on the left and fine-tuning on the right. The curve has the correct shape, with a steep decline at the start followed by a slower one towards the end.

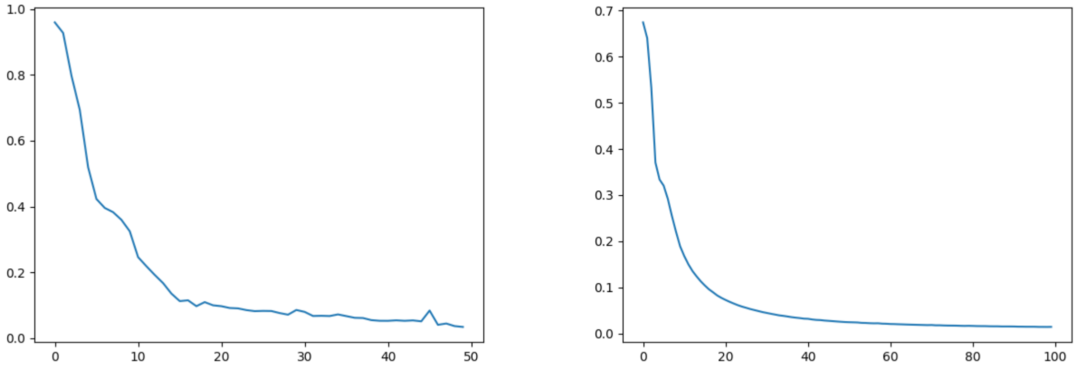


Figure 5.3: Losses for pre-training (on the left) and fine-tuning phases (on the right).

However, it is worth noting that in some situations the pre-training phase has a tendency to get stuck in a local minimum and not improve for several epochs. This issue arises particularly when the model is trained on the full resolution version of ERA5, and it appears to occur randomly regardless of the learning rate. Figure 5.4 illustrates an instance of this behaviour.

Despite the final loss converging to a similar value, the training required more epochs than previous experiments due to this issue. Ultimately, the problem was addressed by increasing the dropout rate

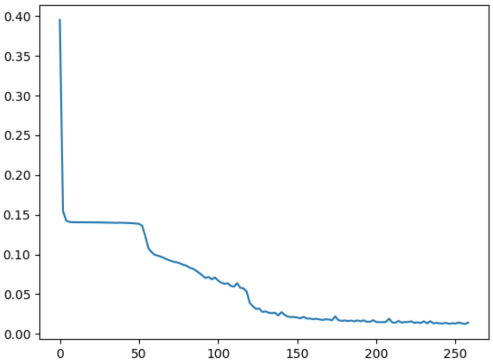


Figure 5.4: Loss where the model's training gets stuck in a local minimum.

for the transformer blocks. Another solution that was tested in some experiments, like the one depicted in the previous image, involved eliminating the early stopping mechanism. This approach prevented the training from being interrupted prematurely, allowing the model to become unstuck.

5.2.2 Results with different lead-times

An important aspect of our experiments concerns the behaviour of the model when tasked with predicting longer lead times. Table 5.2 shows pre-training results that illustrate the architecture's clear difficulties in forecasting the same variables with a lead time of six hours, compared to one hour. This is simply because the model has the same amount of information but with more variability in the possible changes in the behaviour of the variables.

Parameters	Lead Time	Pre-training	Fine-tuning
100M	1 Hour	0.0010 LWMSE	0.0019 MSE
100M	6 Hours	0.0121 LWMSE	0.0030 MSE
600M	1 Hours	0.0023 LWMSE	0.014 MSE
600M	6 Hours	0.068 LWMSE	0.0168 MSE

Table 5.2: Difference in performance when using larger lead time (6 hours).

The use of a larger model should alleviate this effect as it is expected to encode more predictive information than the smaller one. However, training the latter architecture has presented several difficulties, which are discussed in more detail in Section 5.2.4. These problems cause the large model to perform worse than the former one, and this is evident in both pre-training and fine-tuning errors.

Regarding fine-tuning results, a comparable story can be told, with the larger model exhibiting inferior performance to the smaller one by at least one order of magnitude. Furthermore, it should be noted that the fine-tuning error is not a reliable indicator of the model's accuracy since the latter depends directly on the number of cyclones found in the dataset. This is because fewer cyclone images make it easier for the model to consistently predict a flat probability map while achieving low error.

5.2.3 Transfer learning from low to high resolution

An incredibly important application of foundation models is knowledge transfer between datasets or between their low and high resolution versions. The model is pre-trained on the half-resolution version of ERA5 in this project and then fine-tuned on the full-resolution version. Two models of equal size are trained on different datasets, and compared to determine whether the half-resolution pre-trained model outperforms its full-resolution counterpart. Observations show that the half-resolution pre-trained model struggles more to learn variable representations, resulting in inferior performance with Latitude-weighted mean squared error loss. However, upon fine-tuning both models with the same high-resolution ERA5 dataset, the pre-trained model on the half-resolution version slightly outperforms the one trained on the full-resolution version slightly, with a loss of 0.0017 in comparison to 0.0019.

All results are reported in Table 5.3. This outcome demonstrates the model's capability of acquiring the accurate representation of variables at a lower resolution and transferring this knowledge efficiently to a higher resolution version of the same dataset.

Resolution	Pre-training	Fine-tuning
Full	0.0010 LWMSE	0.0019 MSE
Half	0.0021 LWMSE	0.0017 MSE

Table 5.3: Difference in performance when pre-training on a low resolution versus a high resolution dataset

It should be noted that all experiments in this project were run with early stopping, which means that the model is trained until the validation loss stops decreasing for a certain number of epochs. The patience level is set to 5 epochs. This means that there is some variability in the training process, as the model is not always trained for the same number of epochs, and may be stopped too early if there is an unlucky streak of epochs without improvement.

5.2.4 Results after scaling to a larger model

The model used in the previous experiments is a relatively small one, of the same size as that used in the ClimaX paper. It has 8 transformer blocks, each with a hidden size of 1024, for a total parameter size of about 100 million. Several foundation models have been successfully trained in the literature with larger sizes, even in the billions of parameters. The larger model trained in this project has 10 transformer blocks, each with a hidden size of 2048, for a total parameter size of about 600 million. Thanks to the distributed training capabilities of Pytorch Lightning, it is possible to train this model on a cluster of GPUs, in this case Summit, on 128 nodes with 6 GPUs each. This means that the model is trained in parallel on 768 GPUs, resulting in short training times. Using deepspeed as the parallelization technique, it is possible to split the model into layers, so that each GPU is responsible for a different part of the model and can train it in parallel with the others.

One problem that has arisen in training this model is that the architecture is often too large to fit in the memory of all the GPUs. Even with all of DeepSpeeds' optimisations, a model larger than the one described above would cause multiple out-of-memory errors, requiring more workers to partition the layers. This is not a problem with the smaller model, as it can fit into the memory of a single GPU without requiring any layer parallelization technique. To make matters worse, queue times on the Summit are extremely long for jobs requiring more than 128 nodes, meaning that it is not possible to simply increase the number of nodes to solve the problem. This problem could be solved in future work by scaling the project even further, requiring an even larger number of GPUs, but this is not possible with the resources currently available.

Hidden Size	Blocks	Lead-Time	Pre-training	Fine-tuning
1024	8	1	0.0010 LWMSE	0.0019 MSE
1024	8	6	0.0121 LWMSE	0.0030 MSE
2048	10	1	0.0023 LWMSE	0.014 MSE
2048	10	6	0.068 LWMSE	0.0168 MSE

Table 5.4: Difference in accuracy when scaling from a smaller to a larger model.

The results for training this model are similar to the ones obtained with the smaller model, and are reported in Table 5.4.

It appears that increasing the size of the model may not significantly boost performance. This is because of insufficient resources, which results in the dataset being too small to provide a complete understanding of the relationship between the variables. However, increasing the number of samples raises the issue of requiring additional computing power and consequently of longer queue times. Several jobs in this experiment queued for over a day, even when only requiring 128 nodes.

5.3 Accuracy benchmark for temperature

To establish whether the pre-training results are competitive with those from the ClimaX paper, a tailored fine-tuning approach was introduced, incorporating a prediction head with a single variable output. Temperature was selected due to its shared variable with ClimaX. A challenge arises when comparing the results due to the differing forecasting scopes of the two projects, with one focused on lead times of one to six hours, and the other ranging from six hours up to a month. One possible method of comparison is to iterate over the six or one hour prediction of this project multiple times, resulting in forecasting lead times exceeding one day. However, this approach faces several challenges, particularly the accumulated error in the forecasting process that diminishes the accuracy of this method compared to its counterpart. Results for the new fine-tuning approach are reported in Table 5.5.

Lead time	Resolution	Error
1 hour	Full	0.0016 LWMSE
6 hours	Full	0.0047 LWMSE
1 hour	Half	0.0023 LWMSE

Table 5.5: Temperature losses for different dataset and model configurations. `Full` indicates using the ERA5 dataset at 0.25 degrees resolution, `Half` indicates the ERA5 dataset at 1.40625 degrees.

Another important consideration when determining the appropriateness of comparing the two models is that they concentrate on entirely divergent variables and fine-tuning objectives. The ClimaX architecture is designed for a large amount of possible downstream tasks, hence it is pre-trained on more variables to accommodate this wider scope. The model presented in this thesis, however, aimed specifically to predict tropical cyclones, utilising only the variables pertinent to this fine-tuning task. This may result in a more complete understanding of weather and its evolution over time for the former model, which will inevitably lead to a more precise forecasting process.

5.4 Accuracy metric for cyclone prediction

To determine whether the model correctly predicted the presence of a cyclone in a patch, or the absence of one, an ad-hoc metric was developed. This task in fact, is not straight forward. First of all, the model outputs a probability map for the presence of a cyclone, but this representation for one lacks a way to determine whether it thinks one of these large events is present. It could be possible to use a threshold to eliminate solutions which do not exceed a specific level of certainty, but this would ignore situations where a cyclone may be forming, and the probability is mostly uniform. Another drawback of this technique is the fact that it loses information about the center of the cyclone. Assuming that the central position of the cyclone can be calculated with a weighted sum of high certainty pixels, some highly valued outlier points may influence too much the calculation, shifting the center and resulting in an inaccurate metric.

The final metric used, on the other hand, takes into account both the certainty of the prediction, as well as it is able to determine the center of the probability density function, coinciding with the eye of the storm. To determine whether a patch contains a cyclone, the first step is using a threshold to remove low probability pixels. This process will aid the metric in the next phase, in case a cyclone is not present. Once low certainty values are removed, a Shapiro-Wilk test is performed on the 2d grid, determining whether the probability density function present resembles a Gaussian.

By using a low threshold for the probability the Gaussian will also remain mostly intact, and this allows for a more precise calculation of the center of the cyclone.

The resulting accuracy is around 60-70%, this however is strongly impacted by the number of cyclones which are presented in the test set. The distribution used for the final experiments includes one cyclone patch every 29 non-cyclone patches, so the test set should include an analogous ratio of storm images. Ignoring this guideline resulted in a large amount of false positives and negatives respectively. This topic is expanded upon in Section 5.4.1.

A sample of results is shown in Figure 5.5, where the ratio of cyclones has been increased to one every three images, to be able to visualize a correct confusion matrix with a stronger presence

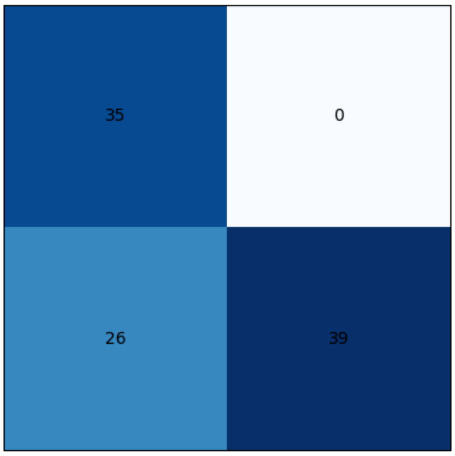


Figure 5.5: Confusion matrix with correctly classified results. Most patches are inserted along the diagonal, so either true positive of true negatives.

of samples along the diagonal, where true positives and true negatives are grouped. The confusion matrix in the Figure represents all samples with a cyclone present on the left cells, and samples without a storm on the right side. These are referred to as ”positive” and ”negative” samples. If a positive sample is classified as not having a cyclone, then it is added in to the bottom left cell, as a false negative (FN). On the other hand if a patch without a cyclone is classified as positive, then it will be allocated in the top right cell as a false positive (FN). In all other cases, the classifier will have correctly recognized the presence or absence of a storm, and the sample will be added to one of the diagonal cells, as a true positive (TP) or a true negative (TN). As reported, there is still a large amount of false positives, as the aforementioned threshold was tweaked to include more samples in this category, at the expense of tilting the classification towards always predicting a cyclone presence.

5.4.1 Data imbalance problem in forecasting of cyclones

The main problem with the cyclone recognition and prediction system is the presence of data imbalance. As tropical cyclones are very rare events, it is not enough to p ass just positive samples in the finetuning process, as the model will always predict a high probability of finding a storm. It is important to achieve a balance between positive and negative samples that reflects the distribution in the real world. In certain rare events, the dataset may lack sufficient positive samples, causing the model to consistently produce negative outcomes. One positive sample will be considered for every 29 negative samples in a dataset of 300,000, resulting in 10,000 cyclone patches. This result still predicts too many events. However, reducing the ratio further may result in insufficient positive samples being included.

As shown in Figure 5.6, a model trained with an incorrectly balanced dataset will predict always or very often a incorrect outcome, leading to a larger number of false positives in the confusion matrix. This is a problem that is present in all experiments, as the number of cyclone patches is extremely small compared to the number of non-cyclone patches.

Another important issue, is that the patch cannot be centered on the cyclone, otherwise the model will learn to always predict a Gaussian probability density function at the center of the image. To avoid this problem, the variable arrays at global scale are patchified by separating each layer into 32×32 sections, ignoring the presence of a storm. This means that in some rare cases, the probability density will be split among two patches, and only the one containing the storm's center will feature a non-zero probability. This inevitably leads to the model learning that some features coming from a cyclone can be ignored, in case these are too close to the edge of a patch. While this issue could have been avoided, all methods ended up encoding the cyclone information in a secondary file, and it

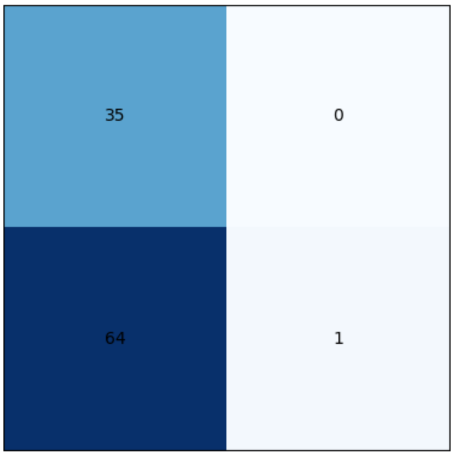


Figure 5.6: Confusion matrix with imbalanced results, all samples are classified as positives, so cyclone patches.

was decided to ignore this problem due to the scarcity of these samples, and the overhead the solution could have caused.

5.5 Accuracy of iterative forecasting process

For the iterative forecasting approach, several modalities of experiments have been run from one to six hours lead time to forecast on lower resolution ERA5 as a benchmark. These consist of sequences of twelve frames with one and six hours lead time respectively, resulting in one modality being able to forecast up to twelve hours and the other up to three days. As hypothesised in section 4.2, the average error for all variables grows over time, as small inaccuracies propagate over time. The losses, both averaged and individual for each variable, are shown in Figure 5.8.

An exception to the first statement is the humidity variable, which appears to decrease with each iteration. The reasons for this behaviour are discussed in the following section.

It should be noted that the sequence in the former figure is taken from the half resolution ERA5 dataset, and there appears to be a difference in the predictive ability of the two datasets. In particular, since the full resolution data encodes more detail, the model has a harder time reproducing each feature with fidelity, resulting in what may appear to be the worst prediction. On the other hand, the half resolution data contains less information in each patch, as each pixel encodes a larger portion of the globe, and the transformer is able to reproduce the image with ease.

Although the Mean Squared Error seems to increase over each iteration, the visual accuracy of the temperature variable forecast, depicted in Figure 5.7, is fairly high, with most weather elements being accurately reconstructed. Moreover, the forecast convincingly indicates the temperature movement direction.

Regarding the disparity between the one hour and six hour lead time forecasts, there appears to be consistent deviation in error, as shown in 5.2.2. This order of magnitude large error however is not that visible in some of the variables, for example for the atmospheric temperature. Therefore, utilising the six hour lead time model for all forecasts may prove advantageous without significant loss of accuracy.

Since these analysis require frame by frame examination, just one has been reported. However more results analogous to the one shown have been saved in the repository containing this project source code. The short gifs merging the sequences have improved visualization convenience. Two versions are accessible: the model's ground truth and the predicted outcome resulting from the iterative process on the input.

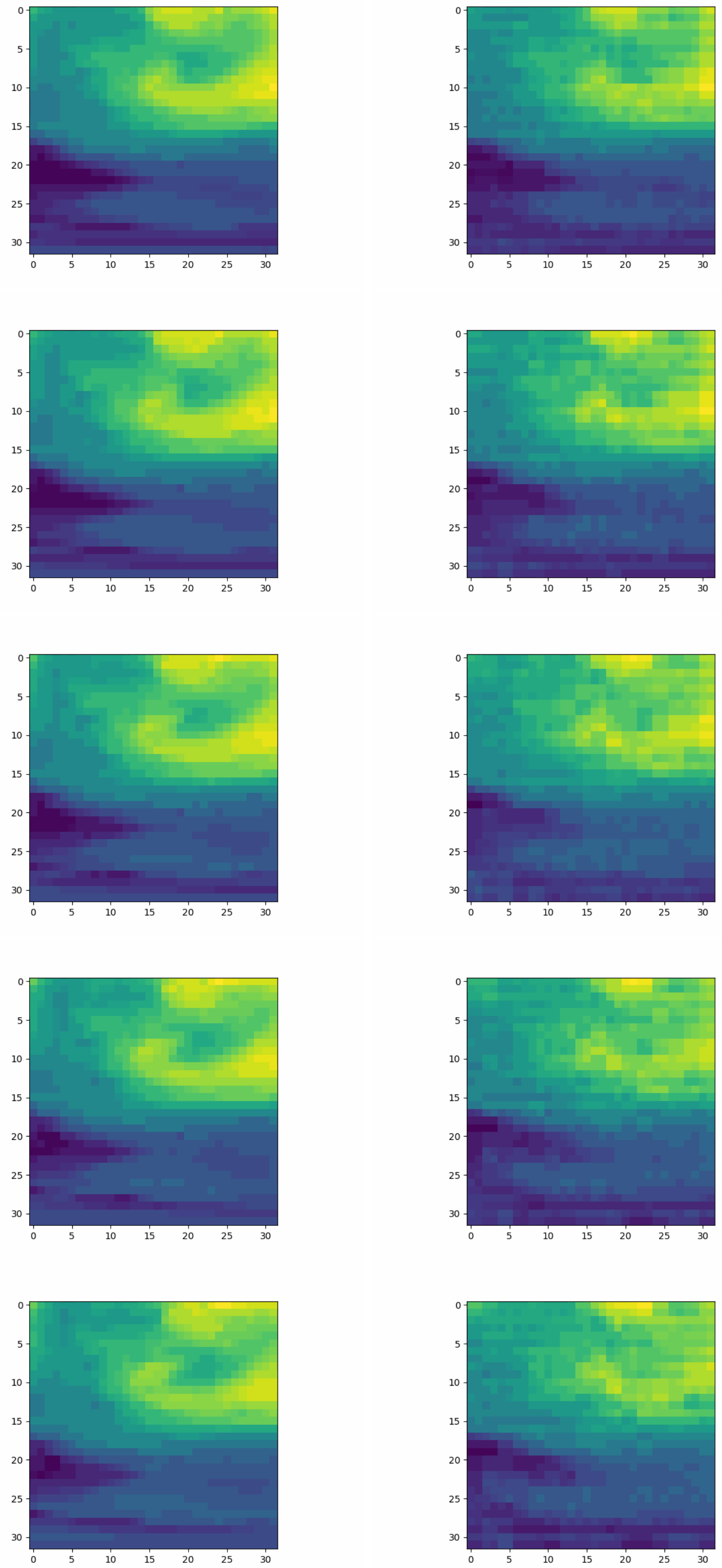


Figure 5.7: Sequence of temperature targets (on the left) and predictions (on the right) over time

5.5.1 Difficulties in humidity prediction

More insight into the prediction process can be gathered by plotting the losses for each iterative forecasting step. Ideally, over time the difference between the target and guess should raise, however, as shown in Figure 5.8, for some variables this is not the case.

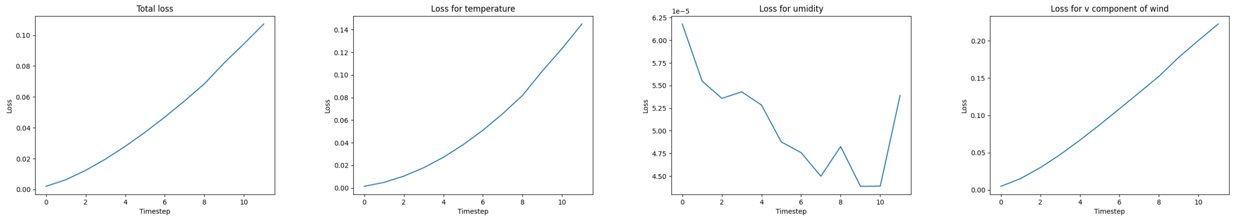


Figure 5.8: Loss in iterative forecasting for temperature, humidity and wind speed, as well as the average of all three. For all but humidity, the loss over time increases, while for the latter it seems to converge to an average value.

Although the overall total loss raises with each iteration, for the humidity variable the difference between target and guess seems to get lower. This is due to a very frequent issue in foundation models, where instead of learning an optimal encoding for a variable, the architecture figures the most frequent encoding and uses it as output, resulting in what can be thought of as the ”best solution on average”.

What can be done to help the model avoid this kind of issue, which was also the approach utilized in ClimaX, is to use a separate loss function for each variable, then sum them all before the back-propagation phase. This guarantees that each variable is optimized independently, and allows for a more flexible approach with different loss functions for different types of data.

5.5.2 Ablation Study: Humidity variable

Since this approach was already used in the ClimaX paper, what was opted for was removing the problematic variable, and test the forecasting ability of the model. The results are shown in Table 5.6.

Humidity	Lead-Time	Pre-train	Finetune
Yes	1	0.0010 LWMSE	0.0019 MSE
Yes	6	0.0121 LWMSE	0.003 MSE
No	1	0.0018 LWMSE	0.003 MSE
No	6	0.0165 LWMSE	0.0029 MSE

Table 5.6: Prediction results with and without humidity, at different lead times.

Although the model appears to not learn humidity accurately, incorporating this variable within the training set leads to a lower error score. The reason for this could be multifaceted, but it is most probable that it provides significant information on how the weather will evolve. Omitting the variable reduces the amount of data available for the architecture to work with. It is also important to note that the last two rows of Table 5.6 omit a variable that is likely to have a large error since it is not properly learned, and yet the model still performs similarly or strictly worse.

Another important aspect to elaborate upon is the fact that during the iterative forecasting process, the loss for humidity seems to get lower over time. The most likely reason for this, is the fact that, if a variable is not learnt correctly, over time the forecast will average to the most likely value for it, and the error will slowly converge.

Regarding fine-tuning, it appears that retaining the humidity information leads to a more precise forecast, particularly for shorter lead times. Removing this variable, which is not well-represented, certainly does not result in better outcomes, despite the possibility that this inadequately learned data may cause confusion for the model. Again, this issue may be a result of the input of additional information, and a more extensive pre-training process could resolve the variable misrepresentation problem to improve the accuracy of predicting tropical cyclones.

5.6 Scalability of pre-training phase on a cluster

With the availability of a large cluster that permits parallelization of the training process, we focused on ensuring scalability during this phase. Various parallelization techniques, including Data Distributed Parallel and DeepSpeed, were taken into account.

The image 5.9 displays the speedup for four different experiment modalities.

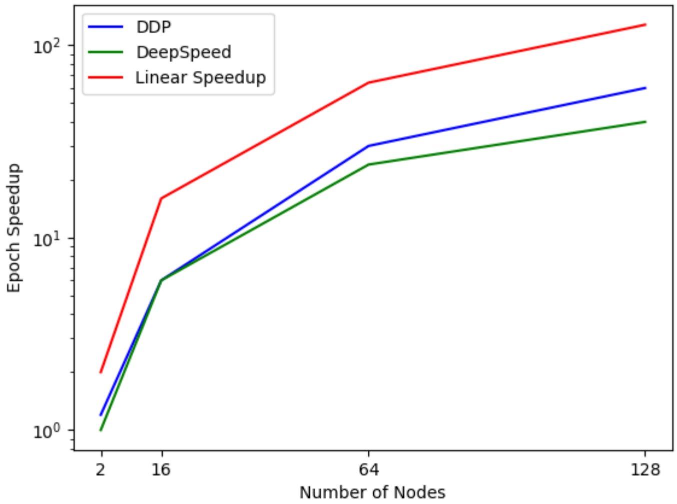


Figure 5.9: Training Speedup for DDP and DeepSpeed, using 2, 16, 64 and 128 nodes with 6 GPUs each.

Speedup measures the relative performance of two systems that process the same problem. It quantifies the improvement in execution speed of a task carried out on two comparable architectures with differing resources. The concept of speedup was introduced by Amdahl's law, which primarily focuses on parallel processing.

Linear speedup with relation to the number of processes can be calculated with Equation 5.1, where p is the number of processors, T_{serial} is the execution time for a program with one CPU, while $T_{parallel}$ is the ideal execution time for the same program, if this were to scale in an optimal manner.

$$T_{parallel} = \frac{T_{serial}}{p} \tag{5.1}$$

In general, the full formula is shown in Equation 5.2 should also take into account the overhead generated from, for example, spawning the extra processes or joining partial results into one.

$$T_{parallel} = \frac{T_{serial}}{p} + T_{overhead} \tag{5.2}$$

Repeating Equation 5.1 for the four processor configurations, the red line can be plotted. The primary difference with relation to the other two lines is that DDP denotes a basic data parallelization method that partitions the dataset among all processes, with each node possessing a replica of the model parameters, and synchronizes the gradient calculation for each model. DeepSpeed, on the other hand, achieves model parallelization by dividing the model into different sections and assigning them to individual workers, resulting in a theoretical speed increase.

Nevertheless, it is worth noting that as shown in Figure 5.9, DeepSpeed does not seem to enhance performance when compared to DDP. This is likely due to the former approach being commonly employed for training models with billions of parameters, with the overhead associated with this approach outweighing the benefits.

Since the hardware present on Summit would allow to scale the architecture even more, a comparison was tried with a 600 million parameter model. The latter however, caused several issues due to its size, as using a simple DDP approach each node's GPU RAM is not enough to store both the parameters as well as the dataset. This renders a comparison between the two approaches unfeasible, as where DeepSpeed would show its benefits, DDP is not applicable.

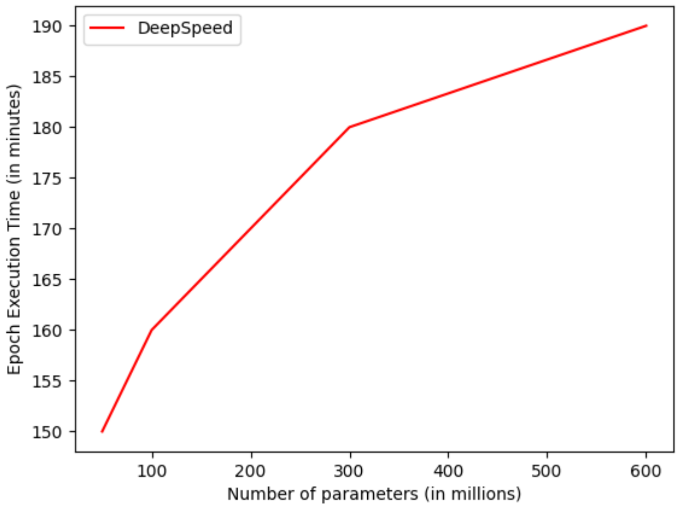


Figure 5.10: Training times when using DeepSpeed on different model sizes, keeping the number of nodes utilized at 128

What can be done to test the performance of DeepSpeed, is explore whether the epoch training times scale in a linear manner with the increment of the model size.

As shown in Figure 5.10, the epoch execution time raises in a steep manner when the model size is still relatively small, and starts to flatten out when approaching the maximum number of parameters tested in this project.

This reinforces the hypothesis that, being a framework for large model training, DeepSpeed does not show its benefits when training trivial architectures, and most of the optimization capabilities come into play when the number of parameters reaches the one billion mark.

6 Future work

6.1 Incremental Probability in Cyclone Prediction

One possible improvement to the presented work arises from the fact that the dataset currently illustrates the likelihood of a cyclone being present in a patch with a simple Gaussian distribution. Nonetheless, as the weather variables increasingly make it more plausible, the probability of a cyclone should raise over time. It may be possible to encode this behaviour by manipulating the standard deviation parameter of the Gaussian, resulting in the probability area progressively expanding over time. This would require determining when a patch is displaying indications of forming a tropical cyclone and identifying the exact central position in the patch.

An example visualizing this approach is shown in Figure 6.2, and it is clearly visible that, during the cyclone intensification in Figure 6.1, the target gets larger and larger.

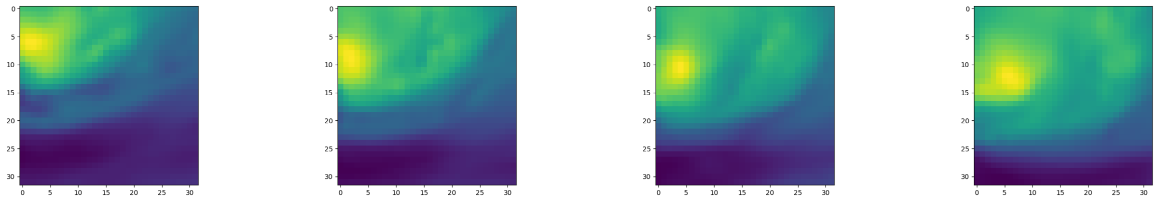


Figure 6.1: Sequence of temperature with cyclone generating

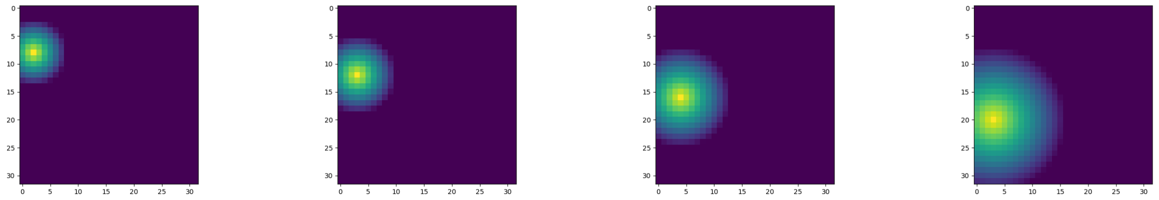


Figure 6.2: Sequence of target with incremental probability

Focusing on this temporal analysis could significantly improve forecasting accuracy and enable the introduction of recurrent architectures, playing into their strengths, resulting in a more accurate iterative process.

Since the analysis presented in this work mainly focused on predicting large-scale events in the near future, we chose not to delve into this subject and instead concentrate on scaling the model and achieving high accuracy in the results. It remains to be said that, in case of a more time-sensitive application, it might be useful to include a temporal dimension in the model, to allow for a more accurate prediction of the cyclone's formation. This would not only improve prediction certainty but also facilitate the visualisation of the formation process as the model would enable step-by-step prediction instead having to wait for the complete storm formation, before the ground truth shows a non-zero probability density.

6.2 Parameter oriented training

One of the main differences between the current implementation of the model and the one presented in the ClimaX paper, is the fact that the latter is trained to predict the input variables shifted by a certain amount of lead time, and the training is repeated for several lead times. This allows the architecture to learn the correct behaviour of each variable over time, and enhances its flexibility by allowing at inference time to take a lead time parameter, and output the correct prediction. This approach has been referenced to as "parameter oriented training", and by its very flexible nature allows for a more general purpose model, which can be used for several different tasks.

Implementing a similar approach in the current work would be one of the most important improvements to be made, as it was preferred to develop a simpler model, and focus on the scaling aspect of the project. The parameter oriented approach would have added complexity to the training phase, as well as to the inference one, making it more time consuming and difficult to implement. This being said, it is still a very important improvement to be made, and would allow for a more diverse and flexible approach.

6.3 Global forecasting system

In a similar way to ClimaX's approach, it may be feasible to merge the image patches and generate a worldwide weather forecast. This would not necessitate any modifications to the current model since the global image normalization is already executed. Additionally, the dataset can be adjusted to share an N-pixel border with adjoining patches and prevent loss of nearby data.

This modification addresses weather variable forecasting and has no impact on this project's finetuning section. Predicting global cyclones may not be necessary since the regions where they most often form are widely recognized, and a regional forecasting approach would be enough.

Another potential method of achieving the same goal may involve utilizing a downsized edition of ERA5 to train the model. This would allow a more contiguous representation of future weather, since patchification would not be necessary on the smaller global image. This approach could potentially compromise the model's ability to forecast tropical cyclones, as the dataset's resolution would be insufficient to provide an accurate depiction of the weather conditions in the region where these phenomena emerge. However, this would necessitate considerable computational resources or the segmentation of the global image, which would defeat the purpose of this approach.

6.4 Time-Series Transformers

Among several advantages of the transformer architecture, the ability to capture long-range dependencies and interactions is particularly central to time series modeling.

A possible improvement to the current work could revolve around increasing the dimensionality of the input data by adding a temporal dimension, and using this information to better predict the future weather. This would imply passing a sequence of time snapshots to the model, and the tensor's dimensions would become `<B, T, V, H, Lat, Lon>`, where T is the number of time snapshots.

This approach has already been tested in the literature, and similar techniques are cited in [WZZ+22], and applications are found in forecasting, anomaly detection, and classification.

As for the modifications necessary to the current architecture, the positional encoding of the transformer has to be changed, to allow for the correct encoding of the temporal dimension. What has been done in similar works, is allow for the embedding to be learned from time series data, and not be fixed as in the current work. A similar approach is to use the timestamp information to influence the training of these embedding layers, and allow for a more accurate representation of the data [CPFL21].

Another important modification to the current architecture revolves around the attention mechanism, which features quadratic complexity in both time and memory due to the length of the time series. A possible solution is to implement a hierarchical approach into the transformer, to take into account the different time scales present in the data [LYL+21].

It has to be noted that it may not be necessary to completely repeat the pre-training phase, as the model is already able to learn the correct representation of the variables, and it may be enough to simply finetune the model, by modifying the variable encoding and aggregation layers to take into account the sequence of time snapshots. This module, as well as the prediction head, which will inevitably have to be modified, should be the only ones to be trained from scratch, as the rest of the model is kept invariant. Futher work on this technique may even include a recurrent transformer block, where the input from the time series is passed iteratively to each block, learning a better encoding of the data.

6.5 Connection to the Intertwin project

Another possible improvement to this work comes from the interconnection between this and Massimiliano Fronza's work [Mas21] on tropical cyclones. This project revolved around the use of Graph Neural Networks [SGT$^+$09] [Kei22] to forecast the presence of tropical cyclones, and was further developed to include unstructured information about sea variables, in an effort to link the former phenomena and ocean eddies.

The project has also been connected to the Intertwin project [Int], which is an EU-funded project with the goal to design and implement a prototype for an interdisciplinary Digital Twin Engine. This open-source platform will provide software components for modelling and simulations to integrate application-specific digital twins.

An effort to join these two works could yield an improvement in our understanding the relationship between these two phenomena, and a consequent raise in accuracy of prediction. In fact, the impact of tropical cyclones on the upper ocean is considered to be restricted to generating inertial waves, while eddies are typically arising from oceanic instabilities. However, in [LWS20] a new hypothesis is proposed, which suggests cyclones directly introduce vorticity, giving rise to these oceanic phenomena.

Furthermore, eddies often have distinct heat and atmospheric conditions from their surroundings, which can significantly affect the path and intensity of tropical cyclones. Research by [ZHZZ23] demonstrates a higher genesis potential index for tropical cyclones that form over anticyclone eddies, as compared to those over cyclone eddies. This study encourages further research into improving predictions of local tropical storms by taking into account the effects of ocean eddies in air-sea models.

One possible mean of unifying the two works could revolve around the use of graph neural network architectures, and integrating these mostly novel approaches into large foundation models. While Fronza's work revolved around graph neural networks, there exist in the literature examples of applications of these kind of models to global weather forecasting [Kei22], which is an essential requirement for developing a foundation model based on GNNs. These techniques have already been tested in the literature, especially with presented in [KKBL$^+$22], where a Graph Transformer is used to generate text, processing both the data with the classical approach, but also building a knowledge graph of the sentence, enhancing the understanding of the context. This approach could be used to build a foundation model which is able to understand the context of the input data, and use this knowledge to improve the prediction ability of the model. In this case, the context could be the current regional weather, and the model could use this information to better predict the future weather.

Another interesting link point between the two works could be the development of a joined architecture, leveraging the information present in both structured and unstructured data. This could be achieved by using the foundation model to process the unstructured data, and then using a GNN encoder for the latter version, the common latent space generated could yield a better representation of the data, and allow for a more accurate prediction.

Examples of this approach are present in the literature, and a similar technique is used in [MKR16], where one of these architecture is used for image matching, a task where two representations of the same object are presented to a neural network, and the latter has to determine whether the two images belong to the same sample. A similar approach could be used to help the foundation model understand the relationship between the different variables presented in the two modalities, and allow for a better prediction of the future weather.

Conclusion

In summary, this project has explored the application of foundation models to the field of weather forecasting, and in particular to tropical cyclone forecasting. By harnessing the power of advanced machine learning techniques and large datasets, the application of foundation models to improve the accuracy and precision of tropical cyclone forecasting has been put into focus. The project's findings demonstrate that foundation models can acquire precise representations of individual variables and effectively reconstruct input data, even across large periods. The versatility of these models can prove invaluable, as the architecture can be trained on a diverse range of variables for weather forecasting and subsequently utilised for a multitude of downstream tasks, including ones significantly different from cyclone prediction.

A system of this kind can effectively transfer knowledge from a low-resolution dataset to a higher resolution one. This is achieved by initially learning the essential features at a low-detail level and then applying this acquired knowledge to the dataset at a higher resolution. The outcomes indicate comparable accuracy in training and fine-tuning on the identical dataset when compared to the knowledge transfer method. This suggests that the model can comprehend weather characteristics at various scales and adjust its actions accordingly.

Scaling tests on the former architecture have shown the scope for improvement in this field, as the bigger model has demonstrated its ability to surpass the smaller one in performance. However, challenges in training it on a substantial cluster of GPUs have hindered further experiments. With a more efficient approach to data encoding, and more experimentation time, there is no doubt future iteration of foundation models for climate will improve on the current version's design.

The iterative forecasting approach has also shown promise as the model can effectively learn variable representation and forecast their behaviour in sequence, which extends the possible lead time. It is yet to be determined if this technique is appropriate for predicting cyclones, as accumulated errors over time may lead to a weather forecast that is too uncertain to be of value.

This project's aforementioned findings will be published in the petascale data hackaton report, as well as shared with the research community for feedback and further improvement. Their contribution will be invaluable in the development of future foundation models for weather forecasting, as well as other fields of application.

In conclusion, the utilization of foundation models in predicting tropical cyclones indicates a noteworthy progression in our capacity to comprehend these complex weather phenomena. As advancements in technology persist, and with more data made available, we can anticipate even more enhancements in the accuracy and lead time of tropical cyclone forecasts.

Bibliography

[AHP$^+$22] Valentine Anantharaj, Samuel Hatfield, Inna Polichtchouk, Nils Wedi, Morgan E. O'Neill, Thomas Papatheodore, and Peter Dueben. An open science exploration of global 1-km simulations of the earth's atmosphere. In *2022 IEEE 18th International Conference on e-Science (e-Science)*, pages 427–428, 2022.

[ARZ$^+$22] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022.

[BFD$^+$22] Stella Bourdin, Sébastien Fromang, William Dulac, Julien Cattiaux, and Fabrice Chauvin. Intercomparison of four algorithms for detecting tropical cyclones using era5. *Geoscientific Model Development*, 15(17):6759–6786, 2022.

[BHA$^+$22] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022.

[Bow23] Samuel R. Bowman. Eight things to know about large language models, 2023.

[BSH21] Peter Bauer, Bjorn Stevens, and Wilco Hazeleger. A digital twin of earth for the green transition. *Nature Climate Change*, 11(2):80–83, 2021.

[BXZ$^+$22] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast, 2022.

[CCNC20] European Commission, Joint Research Centre, S Nativi, and M Craglia. *Destination Earth – Use cases analysis*. Publications Office, 2020.

[CHG$^+$23] Kang Chen, Tao Han, Junchao Gong, Lei Bai, Fenghua Ling, Jing-Jia Luo, Xi Chen, Leiming Ma, Tianning Zhang, Rui Su, et al. Fengwu: Pushing the skillful global medium-range weather forecast beyond 10 days lead. *arXiv preprint arXiv:2304.02948*, 2023.

[CPFL21] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12270–12280, 2021.

[CRD06] Fabrice Chauvin, Jean-François Royer, and Michel Déqué. Response of hurricane-type vortices to global warming as simulated by arpege-climat at high resolution. *Climate Dynamics*, 27(4):377–399, 2006.

[CZZ$^+$23] Lei Chen, Xiaohui Zhong, Feng Zhang, Yuan Cheng, Yinghui Xu, Yuan Qi, and Hao Li. Fuxi: A cascade machine learning forecasting system for 15-day global weather forecast. *arXiv preprint arXiv:2306.12873*, 2023.

[Dal21] Robert Dale. Gpt-3: What's it good for? *Natural Language Engineering*, 27(1):113–118, 2021.

[Das16] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016.

[DBK$^+$20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[DMG$^+$21] Peter Düben, Umberto Modigliani, Alan Geer, Stephan Siemen, Florian Pappenberger, Peter Bauer, Andy Brown, Martin Palkovic, Baudouin Raoult, Nils Wedi, and Vasileios Baousis. Machine learning at ecmwf: A roadmap for the next 10 years, 01/2021 2021.

[EBM$^+$16] V. Eyring, S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer, and K. E. Taylor. Overview of the coupled model intercomparison project phase 6 (cmip6) experimental design and organization. *Geoscientific Model Development*, 9(5):1937–1958, 2016.

[ES18] Abdulmotaleb El Saddik. Digital twins: The convergence of multimedia technologies. *IEEE MultiMedia*, 25(2):87–92, 2018.

[FH22] Noelia Ferruz and Birte Höcker. Controllable protein design with language models. *Nature Machine Intelligence*, 4(6):521–532, 2022.

[FT19] William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019.

[GML$^+$21] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

[Gra98] W.M. Gray. The formation of tropical cyclones. *Meteorl. Atmos. Phys. 67, 37–69*, 1998.

[HBS$^+$23] Jörn Hoffmann, Peter Bauer, Irina Sandu, Nils Wedi, Thomas Geenen, and Daniel Thiemert. Destination earth – a digital twin in support of climate services. *Climate Services*, 30:100394, 2023.

[HG23] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[HH17] Stephan Hoyer and Joe Hamman. xarray: Nd labeled arrays and datasets in python. *Journal of Open Research Software*, 5(1), 2017.

[HMVDW$^+$20] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[Hod94] Kevin I Hodges. A general method for tracking analysis and its application to meteorological data. *Monthly Weather Review*, 122(11):2573–2586, 1994.

[Int] InterTwin. https://www.intertwin.eu/.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kei22] Ryan Keisler. Forecasting global weather with graph neural networks, 2022.

[KKBL$^+$22] Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text generation from knowledge graphs with graph transformers, 2022.

[KN10] M. C. Kruk D. H. Levinson H. J. Diamond Knapp, K. R. and C. J. Neumann. The international best track archive for climate stewardship (ibtracs): Unifying tropical cyclone best track data. *Bulletin of the American Meteorological Society*, 2010.

[LGAM21] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 1, 2021.

[LSGW$^+$23] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. Graphcast: Learning skillful medium-range global weather forecasting, 2023.

[LWS20] Zhumin Lu, Guihua Wang, and Xiaodong Shang. Strength and spatial structure of the perturbation induced by a tropical cyclone to the underlying eddies. *Journal of Geophysical Research: Oceans*, 125(5):e2020JC016097, 2020.

[LYL$^+$21] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*, 2021.

[Mas12] Thomas Mason. Science and technology at oak ridge national laboratory. *OSTI*, 11 2012.

[Mas21] Fronza Massimiliano. Tropical cyclones detection with graph neural networks [final dissertation]. *Università di Trento*, 2021.

[MBA$^+$23] Michael Moor, Oishi Banerjee, Zahra Shakeri Hossein Abad, Harlan M Krumholz, Jure Leskovec, Eric J Topol, and Pranav Rajpurkar. Foundation models for generalist medical artificial intelligence. *Nature*, 616(7956):259–265, 2023.

[MCS$^+$23] S Karthik Mukkavilli, Daniel Salles Civitarese, Johannes Schmude, Johannes Jakubik, Anne Jones, Nam Nguyen, Christopher Phillips, Sujit Roy, Shraddha Singh, Campbell Watson, et al. Ai foundation models for weather and climate: Applications, design, and implementation. *arXiv preprint arXiv:2309.10808*, 2023.

[MKR16] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. Siamese network features for image matching. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 378–383, 2016.

[MSDAP+21] Joaquín Muñoz-Sabater, Emanuel Dutra, Anna Agustí-Panareda, Clément Albergel, Gabriele Arduini, Gianpaolo Balsamo, Souhail Boussetta, Margarita Choulga, Shaun Harrigan, Hans Hersbach, et al. Era5-land: A state-of-the-art global reanalysis dataset for land applications. *Earth system science data*, 13(9):4349–4383, 2021.

[NBK+23] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. Climax: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343*, 2023.

[NMC21] Stefano Nativi, Paolo Mazzetti, and Max Craglia. Digital ecosystems for developing digital twins of the earth: The destination earth case. *Remote Sensing*, 13(11), 2021.

[pdt20] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[PSH+22] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[RDS+20] Stephan Rasp, Peter D. Dueben, Sebastian Scher, Jonathan A. Weyn, Soukayna Mouatadid, and Nils Thuerey. Weatherbench: A benchmark data set for data-driven weather forecasting. *Journal of Advances in Modeling Earth Systems*, 12(11):e2020MS002203, 2020. e2020MS002203 10.1029/2020MS002203.

[Sal23] Erick Burgueño Salas. Number of named tropical cyclones worldwide from 1980 to 2022. *statista.com*, 2023.

[SB18] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[SEH+20] Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. Metnet: A neural weather model for precipitation forecasting. *arXiv preprint arXiv:2003.12140*, 2020.

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[SKQ19] Uwe Schulzweida, Luis Kornblueh, and Ralf Quast. Cdo user guide, 2019.

[SPG22] Weinstein-Raun B. Stein-Perlman, Z. and K. Grace. Expert survey on progress in ai. *AI impact*, 2022.

[TDD+13] Kevin J Tory, RA Dare, NE Davidson, JL McBride, and SS Chand. The importance of low-deformation vorticity in tropical cyclone formation. *Atmospheric Chemistry and Physics*, 13(4):2115–2132, 2013.

[Ulm21] Simone Ulmer. Scientists are building a digital twin of the earth to fight climate change. *World Economic Forum*, 2021.

[UZM+21] Paul A Ullrich, Colin M Zarzycki, Elizabeth E McClenny, Marielle C Pinheiro, Alyssa M Stansfield, and Kevin A Reed. Tempestextremes v2. 1: A community framework for feature detection, tracking, and analysis in large datasets. *Geoscientific Model Development*, 14(8):5023–5048, 2021.

[VSB$^+$22] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. Machine learning model sizes and the parameter gap, 2022.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[Wig19] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[WMK$^+$16] Kevin J.E. Walsh, John L. McBride, Philip J. Klotzbach, Sethurathinam Balachandran, Suzana J. Camargo, Greg Holland, Thomas R. Knutson, James P. Kossin, Tszcheung Lee, Adam Sobel, and Masato Sugi. Tropical cyclones and climate change. *WIREs Climate Change*, 7(1):65–89, 2016.

[WZZ$^+$22] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

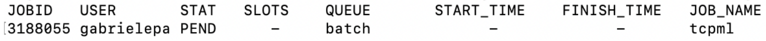
[ZHZZ23] Weikang Zhan, Qingyou He, Ying Zhang, and Haigang Zhan. Anticyclone eddies favor the genesis of off-season tropical cyclone in the western north pacific. *Journal of Geophysical Research: Atmospheres*, 128(1):e2022JD036945, 2023.

[ZLL$^+$23] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, Philip S. Yu, and Lichao Sun. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt, 2023.

Appendix A Difficulties in scaling the train phase

A.1 Jobs remaining pending

One of the main issues encountered during the development of this project is the fact that jobs on Summit often remain pending for a long time, as shown in Figure A.1. This issue is not that noticeable when requesting a low amount of nodes, for example 8 or 16, but becomes more and more evident as the number of nodes increases. Already when requesting 64 nodes, the queue time can be as long as 24 hours, and this is not a problem that can be solved by simply increasing the number of nodes, as the queue time for 128 nodes is even longer. While when requesting a low amount of nodes what is being performed is called "backfilling", which means using resources that are available at the current moment and are not enough to be locked for a larger job, when requesting a large amount of nodes the job is put in a queue, and is only executed when the requested resources are all available. This means that, when requesting 128 nodes, the job will only start when 128 nodes are free, and in case only a subset of these is available, the scheduler may decide to start a program which has been in the queue for a smaller amount if time, but only require a smaller amount of nodes.



```
JOBID   USER       STAT  SLOTS    QUEUE       START_TIME    FINISH_TIME   JOB_NAME
3188055 gabrielepa PEND    -      batch            -             -        tcpml
```

Figure A.1: Jobs remaining pending for several hours

This issue is not present on other clusters, such as Andes, where the queue times are generally shorter, as well as the magnitude of the programs being run. What also differs is the type of tasks being executed, where Andes is a commodity cluster generally used for data pre-processing and analysis, while Summit is a leadership class cluster, used for large scale simulations and machine learning, jobs which are notoriously computationally expensive and require long training times.

A.2 FSDP not available

As already mentioned in Section 2.5.1, the use of FSDP is not available on Summit, and this has caused several issues during the development of this project. In an ideal situation, comparison between the three available parallelization techniques would have been made, and the most suitable one would have been chosen. Since the aforementioned one is not usable, the only model parallelization approach that has been tested is DeepSpeed, and it is clear from Section 5.6 that it is not suitable for small models such as the one used in this project.

The issues present in using FSDP stand from a versioning issue present on Summit, which is already known and it would require a large amount of changes in different packages to be solved. Since Summit uses a specific collection of packages called open-ce, which are tested and verified to work together, it is not possible to simply update the version of a single package, as this would cause several other modules to break.

A possible solution to this issue would be to split the model using the Horovod [SB18] package, but is was decided to not pursue this approach, as it would require a large amount of work, and it would not be possible to compare the different parallelization techniques.

A.3 Local minima slowing down training

A problem we came across during the project's development is that, in certain instances, the loss fails to decrease for multiple epochs, causing the model to get stuck in a local minimum.

This problem is not universal and appears to be more frequent when training using the full resolution version of ERA5 and a bigger model. An instance of this pattern is illustrated in Figure A.2, wherein the loss is stable for multiple epochs before abruptly decreasing.

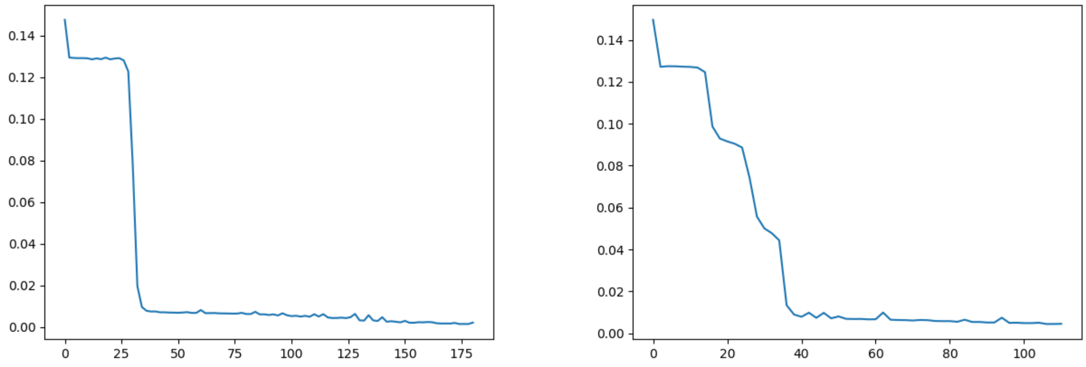


Figure A.2: Examples of losses getting stuck for several epochs and then decreasing suddenly. On the right it is shown how the same phenomena can happen several times in the same training process.

The primary hypothesis for this behaviour is the use of a specific training technique like DeepSpeed, which is utilised to parallelize the model across several GPUs. It is plausible that this approach, which is still in its initial developmental stage, is incapable of handling all possible situations and can lead to the model being stuck in a particular minimum, for example due to erroneous gradient averaging from different workers.

A resolution for this problem has not been discovered in the course of this project, resulting in some cases where the implementation of Early Stopping has become impractical. This is due to the model halting the training process too prematurely, consequently falling short of the desired level of accuracy. This issue, joined with the long queue times on Summit, has made the development of this project particularly long and tedious, as it was not possible to run several experiments in parallel, and the time required to run a single experiment was often too long to be able to test all desired variations of the model.

Appendix B Difficulties with accuracy metric

B.1 Balancing cyclone patches

While not being a real issue, the inability to run a satisfying number of tests rendered the use of the accuracy metric for cyclone prediction extremely difficult. This is due to the fact that the ratio of cyclone patches used per non-cyclone patch, one to 30, may not be the correct one for a real world scenario. In fact, cyclones are extremely rare events, and it is possible that the ratio used in most experiment is still too high, and that a ratio of one to 100 or even one to 1000 would be more appropriate. To compound this issue, the more the number of cyclone patches lowers, the higher the accuracy supposedly becomes, as the model has an easier and easier time predicting a flat probability map, and still achieving a low error score.

To solve this issue, a more in depth analysis on the frequency of occurrence of tropical cyclones should be performed, and the ratio of cyclone patches to non-cyclone patches should be adjusted accordingly.

To quickly discuss the matter, considering a single year of ERA5 data, the number of total patches a model is trained on is calculated as follows:

$$\begin{aligned} \text{Number of patches per image} &= \frac{1440}{32} \times \frac{721}{32} = 990 \\ \text{Patches per day} &= 990 \times 24 = 23.760 \\ \text{Patches per year} &= 23.760 \times 365 = 8.670.240 \end{aligned} \tag{B.1}$$

Considering the ratio of cyclones used in most experiments, one to 29, the number of cyclone patches per year is calculated as follows:

$$\text{Cyclone patches per year} = \frac{8.670.240}{30} = 289.008 \tag{B.2}$$

This means that, in a single year of training, the model is trained on 289.008 cyclone patches, which is an extremely high number considering that the number of cyclones in a year is much lower than that.

Considering that, according to [Sal23] the number of tropical cyclones per year hovers around 90, and the number of hurricane days around 170, the ratio of cyclone patches to non-cyclone patches should be adjusted accordingly, to be able to train the model on a more realistic dataset.

To calculate a correct ratio of cyclone and non-cyclone patches, the number of hurricane days can be used. Assuming that only one hurricane can happen on a single day, only one storm image can be present for each day, and the number of cyclone patches per year can be calculated as follows:

$$\text{Correct patch number} = \frac{170}{365 \times \text{Patches per day}} = 0.00002 \tag{B.3}$$

If the dataset were to remain the same size, so 300.000 samples, and the ratio were to be adjusted accordingly, the number of cyclone patches can be calculated as follows:

$$\text{Cyclone patches in dataset} = \text{Correct patch number} \times 300.000 = 6 \tag{B.4}$$

Passing six cyclone patches to a model would inevitably lead it always predicting a flat probability map, not considering the outlying samples. This means that, in order to train the model on a more realistic dataset, the number of samples would have to be increased, to allow for a higher number of cyclone patches to be present.

Since satisfying this requirement would have delayed the execution times of the program even more, it was opted to keep the initial ratio of samples, and leave this kind of optimization as future development.

B.2 Variability of p-value in different test sets

Another issue encountered during the development of this project is the variability of the p-value in different test sets. This is the case since the testing has been conducted on a subset of the original dataset, and the number of cyclone patches in each test set is not constant.

This is a problem as the p-value is used to determine the threshold which separates Gaussian and non-Gaussian probability densities in the two-dimensional map. Since this number is strictly dependant on the amount of randomly shuffled cyclone patches, it is possible that the threshold may not be correct, and that a different value would be more appropriate.

In a real world scenario, this issue would not be present, as the model would be tested on the entire dataset, and the number of cyclone patches would be constant. Since several testing phases were run, it was opted to adjust the p-value for each test set, to ensure that the threshold was always correct, as well as ensure the highest number of evaluation runs.

Another hyper-parameter to be adjusted is the noise threshold alpha, which is used to remove uncertainty from the prediction. It was noticed that a lower value for this parameter is beneficial, probably as it helps in keeping the Gaussian shape intact. For most experiments this value has been set to 0.002, but it is possible that a lower value would be more appropriate, and is something to be tested in future work.

Appendix C Comparison with other forecasting architectures

Similar approaches to the current work are already present in the literature, spanning from Numerical forecasting techniques to Machine Learning ones. In this section, a brief overview of the most important ones is presented, and a comparison to the current work is made.

It has to be noted, that all of these approaches are presented by research teams with a large amount of resources at their disposal, backed up by large corporations such as Microsoft, are often run on similar supercomputers to the one used in this project, and went though much longer development cycles.

Table C.1 summarizes the most important characteristics of the different approaches, and compares them to the current work. It is clear that, especially in the current landscape, the use of foundation models is a novel approach to the problem, and while it has been explored, no work has made a consistent effort in scaling one of these architectures to the size of the current work.

Another important point, is the fact that, before the ClimaX work, no effort was made in the direction of creating a general purpose foundation model for climate forecasting, and all the approaches in the literature are tailored to a specific task, such as precipitation forecasting or cyclone forecasting. Since a more general approach could play into the strengths of Vision Transformers, possible improvements on this subject are to be expected in the future.

As for the resolution of the forecast, as well as the lead time, the current work is able to compete with some of these approaches, ultimately falling short in the lead time category.

	Params.	Dataset	Lead-time	Forecast	Purpose	Org.
FM4C[1]	100M 600M	ERA5	From 1 hr To 3 days	Regional	Cyclone prediction	/
ClimaX	100M	ERA5 CMIP6	From 6 hrs To 30 days	Global	Weather forecast	Microsoft
Pangu Weather	256M	ERA5 IBTrACS	From 6 hrs To 6 days	Global	Cyclone tracking	Huawei
Fourcast	67M	ERA5	From 18 hrs To 7 days	Global	Wind forecast	Nvidia
IFS	≤1M	ERA5	From 6 hrs To 10 days	Global	General forecast	ECMWF
GraphCast	37M	ERA5	From 1 day To 10 days	Global	General forecast	Google

Table C.1: Comparison between this, and the more recent weather forecasting architectures

When, on the other hand, we take into account the structural differences between architectures, the more similar approach to the current work is without question ClimaX, where the same variable encoding and aggregation mechanism is used. There are however several differences, in the type of data used, where the current implementation uses three dimensional variables similarly to PanguWeather, and the lead time is set at training time and no parameter oriented training is used. The latter approach is to be considered more desirable, as it allows the model to better learn the behaviour of variables over time, and it guarantees any type of prediction with a single pre-training process.

An important emphasis has to be put on the organizations behind these works, as the computational resources necessary for training these kind of architectures is extremely high, access to a large amount of GPUs is necessary. Only a handful of organizations in the world have access to the required resources, and this is a problem that is not going to be solved in the near future. While some of the preminent national laboratories allow for access to their supercomputers, through hackatons similar to this event, the sheer size of the models and the amount of data necessary to train them is often too large to be able to run several experiments.