ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Diffusion generative models for weather forecasting

Relatore:
Chiar.mo Prof.
ANDREA ASPERTI

Correlatore:
Chiar.mo Dott.
FABIO MERIZZI

Presentata da:
ALBERTO PAPARELLA

Sessione II
Anno Accademico 2022-23

Sommario

Negli ultimi anni, i tradizionali approcci numerici utilizzati per la previsione accurata del tempo meteorologico hanno dovuto affrontare una crescente sfida da parte dei metodi basati sull'apprendimento profondo. Gli insiemi di dati storici ampiamente impiegati nelle previsioni meteorologiche a breve e medio termine seguono spesso una struttura regolare a griglia spaziale. Questa disposizione somiglia notevolmente alle immagini: ciascuna variabile meteorologica può essere infatti rappresentata come una mappa o, considerando l'aspetto temporale, come un video.

Diverse categorie di modelli generativi, tra cui le Reti Generative Avversarie (Generative Adversarial Networks), gli Autoencoder Variazionali (Variational Autoencoders) e i più recenti Modelli di Diffusione per la Rimozione del Rumore (Denoising Diffusion Models), hanno dimostrato ampiamente la loro efficacia nell'affrontare il problema della previsione del prossimo *frame*. Di conseguenza, risulta naturale valutare le loro prestazioni sui benchmark dedicati alla previsione meteorologica. Tra tutti, i modelli di diffusione si rivelano particolarmente affascinanti in questo contesto, considerando la natura intrinsecamente probabilistica della previsione del tempo. Essi mirano a modellare la distribuzione di probabilità degli indicatori meteorologici, fornendo una stima del valore atteso, ossia la previsione più probabile.

L'obiettivo principale di questa tesi è l'approfondimento dell'utilizzo di tali modelli nell'ambito delle previsioni meteorologiche, con particolare attenzione alle previsioni delle precipitazioni a breve termine (nowcasting). Per questo studio, si è fatto ricorso a un sottoinsieme specifico del dataset ERA5,

contenente dati orari riferiti all'Europa occidentale nel periodo compreso tra il 2016 e il 2021. Nel contesto di questa indagine, è stata analizzata l'efficacia dei modelli di diffusione nella gestione delle previsioni delle precipitazioni a breve termine. La valutazione delle prestazioni dei modelli è stata effettuata attraverso il confronto con i ben consolidati modelli U-Net, ampiamente documentati nella letteratura scientifica.

L'approccio proposto, denominato Generative Ensemble Diffusion (GED), sfrutta un modello di diffusione per generare un insieme di possibili scenari meteorologici, i quali vengono successivamente combinati in una previsione probabile mediante l'utilizzo di una rete di post-elaborazione. Questo approccio si è dimostrato significativamente più efficace rispetto ai recenti modelli di apprendimento profondo, ottenendo prestazioni globali migliori.

Il Capitolo 1 svolge un ruolo introduttivo, presentando il problema in esame: le previsioni meteorologiche. In questo contesto, vengono dettagliatamente discussi i metodi numerici e statistici utilizzati per affrontare questo compito, con un'attenzione particolare rivolta all'applicazione delle tecniche nel campo delle previsioni di precipitazioni a breve termine. Successivamente, si delinea il panorama dei modelli generativi, una classe di tecnologie alla base dei modelli centrali analizzati in questa tesi, i cosiddetti modelli a diffusione, i quali saranno approfonditamente esaminati nel Capitolo 2. Inoltre, si conduce un'esplorazione dello stato dell'arte in merito alla loro applicazione nelle previsioni meteorologiche e nella previsione delle precipitazioni.

Nel Capitolo 3, viene delineato un approccio innovativo nell'impiego dei modelli a diffusione per la previsione delle precipitazioni a breve termine. Questo capitolo include un'esaustiva descrizione dell'architettura del modello a diffusione integrata e un confronto dettagliato con altre metodologie simili presenti nella letteratura scientifica. Successivamente, nel Capitolo 4, l'efficacia di questa architettura è valutata attraverso un confronto con lo stato dell'arte, e i risultati ottenuti dimostrano chiaramente la sua validità, portando alla presentazione di un articolo [1] proposto per pubblicazione su *Neural Computing and Applications (NCAA).*

Introduction

In recent years, traditional numerical methods used for accurate weather prediction have faced increasing challenges from deep learning techniques. The historical datasets commonly employed for short and medium-range weather forecasts are typically structured in a regular spatial grid format. This arrangement closely resembles images, with each weather variable akin to a map or, when considering the temporal axis, as a video.

Several classes of generative models, including Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and the recent Denoising Diffusion Models (DDMs), have demonstrated their effectiveness in tackling the next-frame prediction problem. Consequently, it is only natural to assess their performance in the context of weather prediction benchmarks. DDMs, in particular, hold strong appeal in this domain due to the inherently probabilistic nature of weather forecasting. This methodology aims to model the probability distribution of weather indicators, with the expected value representing the most likely prediction.

This thesis is dedicated to investigating the application of diffusion models in the realm of weather forecasting, with a specific focus on precipitation nowcasting. To achieve this, a specific subset of the ERA5 dataset has been leveraged, encompassing hourly data for Western Europe spanning the years 2016 to 2021. Within this context, the effectiveness of diffusion models has been rigorously assessed in the challenging domain of precipitation nowcasting. The research is conducted in direct comparison to the well-established U-Net models, as extensively documented in existing literature.

The proposed approach, referred to as Generative Ensemble Diffusion (GED), harnesses a diffusion model to generate a diverse set of potential weather scenarios. These scenarios are subsequently amalgamated into a probable prediction through the application of a sophisticated post-processing network. In direct contrast to recent deep learning models, the GED approach consistently demonstrated superior performance across multiple performance metrics, underscoring its significant advancement in the field of weather forecasting.

Chapter 1 introduces the issue under examination, weather forecasting, describing the relative state-of-the-art numerical and statistical methods, with emphasis on precipitation nowcasting, as well as the predominant data sources for weather forecasting. The chapter also introduces generative models, the class of technologies comprising the ones under scrutiny, i.e., diffusion models, described in chapter 2. Furthermore, the state-of-the-art regarding the application of generative models to weather forecasting and precipitation nowcasting is also explored.

The proposed approach for the application of diffusion models to precipitation nowcasting is described in chapter 3, presenting the architecture of the integrated diffusion model, as well as a comparison with similar recent approaches found in the literature. This architecture is successively compared with the state-of-the-art in chapter 4, providing successful results, bringing to the writing of a paper [1] which has been submitted to *Neural Computing and Applications (NCAA).*

Contents

List of Figures

List of Tables

Listings

Chapter 1

Preliminaries

This chapter aims to offer an overview of the fundamental components of this thesis. It involves a comprehensive exploration of the primary subject under scrutiny, namely weather forecasting, with a particular emphasis on the specialized field of precipitation nowcasting. Furthermore, it provides insights into the current landscape of numerical and statistical models. Finally, it offers an overview of generative models, the larger class of techniques comprising the models under examination, namely diffusion generative models.

1.1 Weather forecasting

Weather forecasting is the intricate process of predicting the atmospheric conditions at a specific location and time in the future. This vital science profoundly impacts our daily lives, guiding choices ranging from scheduling outdoor activities, assessing travel safety, and managing agricultural and energy resources. One of its critical roles is in safeguarding public safety, as it provides crucial alerts and warnings about severe weather events such as hurricanes, tornadoes, and floods, enabling individuals to take timely precautions [12]. Additionally, businesses rely on weather forecasts to optimize their operations, affecting decisions concerning staffing and inventory management, ultimately enhancing both efficiency and profitability.

Weather forecasts can be broadly classified into **short-term** (0-3 days), **medium-term** (3-7 days), and **long-term** (beyond 7 days) predictions. Short-term forecasts exhibit greater accuracy, as the atmospheric dynamics are more predictable over shorter spans. Conversely, long-term projections carry inherent uncertainties due to the complexity of long-range forecasting.

Initiating the weather forecasting process entails amassing extensive data from diverse sources. This data encompasses crucial variables such as temperature, humidity, wind speed and direction, air pressure, cloud cover, and more. Observations are sourced from an extensive network comprising ground-based weather stations, weather balloons, satellites, radar systems, and most recently drones [13]. These observations form a snapshot of the current atmospheric state, forming the foundation for constructing computer models [14] that simulate future weather conditions.

The bedrock of contemporary weather forecasting is the discipline known as **Numerical Weather Prediction (NWP)** [15, 16]. NWP relies on sophisticated mathematical models to simulate the evolving behaviour of the atmosphere over time. These models partition the atmosphere into a three-dimensional grid and employ fundamental equations of fluid dynamics, thermodynamics, and heat transfer to prognosticate its evolution.

The models commence with the initial observational data and then advance the atmospheric state in discrete time intervals [17]. To heighten the precision of NWP models, a technique called **data assimilation** is deployed. This process fuses real-time observational data with model output to refine the model's initial conditions. By incorporating current observations, meteorologists rectify disparities between the model's predictions and the actual atmospheric state.

NWP models generate an extensive array of meteorological variables, including temperature, precipitation, wind speed, and more, at various future time intervals. Meteorologists scrutinize these model outputs to formulate weather forecasts, discerning patterns and trends in the data to forecast temperature, rainfall, storms, and other weather phenomena. After the forecast period, meteorologists gauge the accuracy of their predictions by comparing them with the actual atmospheric conditions to enhance future forecasts. Weather forecasts are continuously updated as fresh data becomes available, with short-term forecasts often receiving multiple updates daily, while long-term forecasts are typically refreshed less frequently. Some well-known NWP models include the **Global Forecast System (GFS)** [12], the **European Centre for Medium-Range Weather Forecasts (ECMWF)** model [3], and the **Weather Research and Forecasting (WRF)** model [18].

It is imperative to acknowledge the inherent limits posed by weather forecasting due to the innate complexity and chaotic nature of the atmosphere, as its accuracy hinges on several factors, including data quality, the intricacy of the atmospheric system in question, and the forecast duration. Specifically, forecasts for significant events such as hurricanes, tornadoes, floods, and extreme weather occurrences are much more complex, requiring tailored models and monitoring systems designed to predict and track these specific types of weather phenomena. Furthermore, running NWP models requires significant computational power, often involving supercomputers, due to the complexity of the equations and the need for high spatial and temporal resolutions.

Nevertheless, the continual refinement of data collection, modelling techniques, and computing capabilities has significantly elevated our capacity to deliver precise and timely weather predictions, ultimately saving lives and aiding in strategic planning.

1.1.1 Precipitation nowcasting

Precipitation nowcasting is a specialized branch of weather forecasting that focuses on predicting short-term precipitation patterns over a relatively short time horizon. Specifically, it deals with predicting the location, intensity, and movement of precipitation (rain, snow, sleet, or hail) for the next few hours, typically up to six hours in advance. It is essential for various applications, including flood prediction, transportation management, and outdoor event planning.

It relies on various sources of data to make accurate predictions, including weather radar systems, providing information on the location and intensity of precipitation in real time, satellite imagery, offering insights into cloud cover, which is essential for predicting precipitation, and surface observations, with ground-based weather stations providing valuable information on temperature, humidity, and wind speed, which can influence precipitation.

A critical tool for precipitation nowcasting is the **doppler radar** [19, 20]. It not only detects the presence of precipitation but also measures the velocity of raindrops and their direction of movement. Doppler radar data help meteorologists track the movement and intensity of precipitation systems, allowing for more accurate short-term predictions.

Numerical and statistical weather prediction models play a role in precipitation nowcasting by providing guidance on how precipitation patterns may evolve. These models often operate at higher spatial and temporal resolutions than traditional weather models to capture short-term changes in precipitation. Sophisticated algorithms are employed to process the available data and create precipitation nowcasts. These algorithms consider factors such as radar reflectivity, precipitation type, and atmospheric conditions.

While numerical weather prediction models may accurately forecast precipitation likelihood and overall intensity across extensive geographic regions and medium-term timeframes, they face a more formidable task when dealing with short spatial and temporal scales, as highlighted by [21]. Over intervals of less than 4 hours, they often fall short of the performance achieved by persistence-based forecasts, as noted by [22]. This discrepancy can be primarily attributed to the inherent stochastic nature of precipitation, compounded by the prolonged computational time typically demanded by numerical methods. These methods require substantial processing time to assimilate extensive data and incorporate them into the initial conditions of the models. The challenge is particularly pronounced in the case of **convective precipitation**, characterized by intense rainfall originating from cells spanning just a few tens of kilometers, as expounded upon in [23].

Machine learning techniques, including neural networks and decision trees, are increasingly used to improve nowcasting accuracy. Precipitation nowcasting models typically operate at fine spatial and temporal resolutions. This means they can provide predictions for small geographic areas and short time intervals, which is crucial for local and immediate decision-making. These techniques will be better discussed in subsection 1.3.1.

Precipitation nowcasting has numerous practical applications, including flood prediction and management, anticipating heavy rainfall or snowmelt helping authorities take proactive measures to prevent flooding, transportation, and aviation, as knowing where and when precipitation will occur is vital for planning road maintenance, and air traffic control, and airport operations, as well as outdoor events and activities, with event organizers and outdoor enthusiasts relying on nowcasts to plan activities and ensure safety during inclement weather.

1.1.2 An example of a NWP model for precipitation nowcasting: STEPS

A **Short-Term Ensemble Prediction System (STEPS)** [24, 25] is a specialized tool used in meteorology to provide probabilistic forecasts for relatively short timeframes, typically ranging from a few hours to a few days into the future. This system is designed to address the inherent uncertainty in weather forecasting by generating multiple forecasts or scenarios, known as **ensemble members**, rather than a single deterministic forecast.

Specifically, STEPS employs an ensemble approach (Figures 1.1 and 1.2), which involves running multiple simulations or model forecasts with slight variations in initial conditions and model parameters [26]. These ensemble members represent a range of possible weather outcomes. Each ensemble member has a likelihood associated with it, allowing meteorologists and decision-makers to assess the range of potential weather scenarios and their associated probabilities.

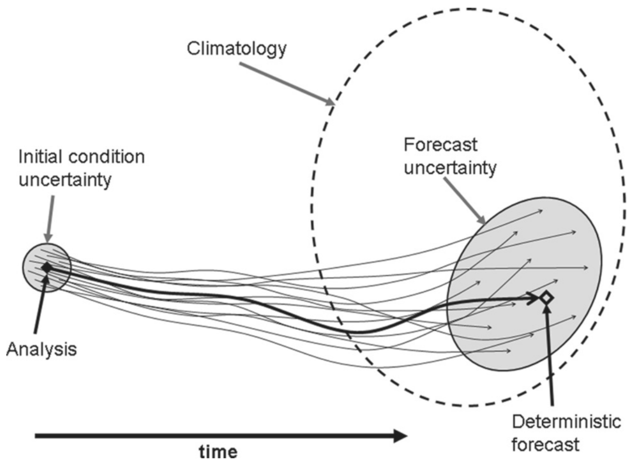


Figure 1.1: Forecasting high-impact weather using ensemble prediction systems. Image taken from [2].

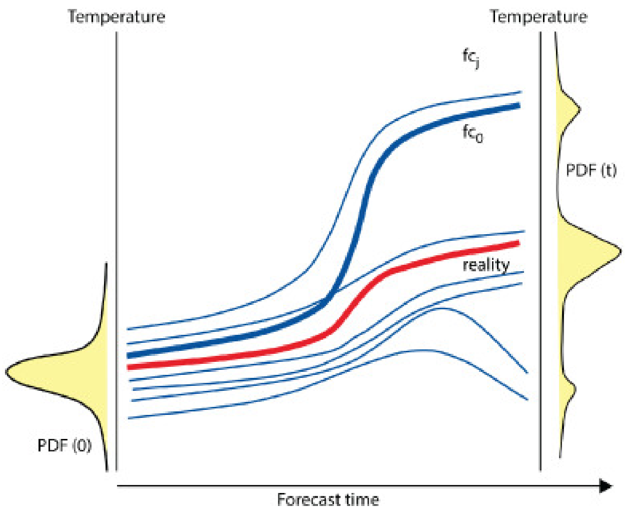


Figure 1.2: European Centre for Medium-Range Weather Forecasts (ECMWF) [3] Ensemble Prediction System. Image taken from [4].

The ensemble members typically start with slightly perturbed initial conditions. These perturbations are small adjustments to the initial observations used to initialize the models. By introducing variability in the initial conditions, STEPS captures the inherent uncertainty in weather observations. In addition to variations in initial conditions, STEPS may also employ different model configurations or parameterizations for its ensemble members. These variations reflect the uncertainty in the modeling itself. As new observational data becomes available, STEPS can assimilate this data into its ensemble members, updating the forecasts in real-time. **Data assimilation** is crucial for improving forecast accuracy.

STEPS is used for a wide range of applications, including severe weather prediction (e.g., thunderstorms, heavy rainfall), short-term climate monitoring, and decision-making in various sectors, such as agriculture, transportation, and emergency management. One of the advantages of STEPS is its ability to communicate forecast uncertainty. Users can assess the likelihood of various weather outcomes, aiding in risk assessment and decision-making.

To assess the performance of the ensemble forecasts, meteorologists use verification and skill scores, such as the **Brier Score** [27] and the **ROC curve** [28], to determine how well the ensemble system matches observed weather conditions. Researchers and meteorological agencies continuously work to improve STEPS by refining ensemble techniques, enhancing data assimilation methods, and incorporating advancements in NWP models.

PySTEPS [29] is an open-source Python library for short-term precipitation nowcasting. It is designed to provide tools and models for forecasting precipitation at high spatiotemporal resolution, typically within the range of 0 to 6 hours into the future. It's a valuable resource for researchers, meteorologists, and data scientists working on precipitation forecasting and related applications. It is an open-source project, which means it benefits from contributions and collaboration from the meteorological and data science communities. This open approach encourages the development and refinement of precipitation nowcasting techniques.

PySTEPS is capable of working with various sources of precipitation data, including radar, satellite, and rain gauge observations. It provides tools for data acquisition, quality control, and preprocessing, as well as several nowcasting models that can be used for short-term precipitation prediction. Some of these models are based on traditional statistical and radar-based techniques, while others leverage machine learning approaches. The library also supports ensemble methods for combining multiple models to improve forecasting accuracy.

PySTEPS operates at high spatial and temporal resolutions, allowing for precise and detailed precipitation nowcasting. This is crucial for applications like flood forecasting, emergency response, and transportation management. The library offers tools for verifying and evaluating the performance of nowcasting models, helping users assess the accuracy and reliability of their precipitation forecasts.

PySTEPS can be integrated with other Python libraries and tools commonly used in the data science and meteorology fields, including NumPy, Pandas, Matplotlib, and Jupyter notebooks, making it accessible and convenient for users familiar with these tools. Users can customize and adapt the PySTEPS library to their specific needs. This includes the ability to implement and experiment with new nowcasting models or techniques. Furthermore, it provides comprehensive documentation and tutorials to assist users in understanding and applying the library's functionality effectively.

1.2 Data sources for weather forecasting

This section provides an overview of the most peculiar data sources for weather forecasting found in the literature. Specifically, it introduces the dataset used for the experiments in this work (chapter 4), i.e., ERA5, also providing an in-depth description of the attributes of interest, and two other data sources taken into consideration that have not been used but can be interesting for future work.

1.2.1 ERA5

ERA5 [30] marks the fifth advancement in ECMWF's reanalysis series, encompassing global climate and weather data over an impressive 80-year span, commencing from 1940 onwards. **Reanalysis** is a complex process that blends model-generated data with observations sourced from across the globe, yielding a comprehensive and internally coherent dataset founded on the principles of physics. This technique, recognized as **data assimilation**, closely parallels the methodology employed by NWP centers. At specific intervals (e.g., every 12 hours at ECMWF), a previous forecast is melded with the most up-to-date observations in an optimized manner to generate a refreshed and improved assessment of the atmospheric condition, referred to as **analysis**, that significantly contributes to refining subsequent forecasts.

Reanalysis operates along similar lines but at a reduced resolution, enabling the development of a dataset spanning numerous decades. Unlike real-time forecasts, reanalysis isn't constrained by the necessity for immediate predictions. This flexibility allows for ample time to amass observations, and when delving further into the past, to incorporate enhanced versions of the original data. Collectively, these factors culminate in a substantial enhancement of the reanalysis product's quality.

ERA5 provides hourly estimates for a diverse range of atmospheric, ocean-wave, and land-surface parameters. These estimations come with an accompanying assessment of uncertainty, sampled by a 10-member ensemble every three hours. To enhance user convenience, both the ensemble's mean and its range have been pre-computed. These uncertainty estimates are closely tied to the evolving observation system, which has undergone substantial advancements over time, and they also pinpoint regions particularly responsive to atmospheric changes. To cater to a wide spectrum of climate-related applications, monthly-average values have also been pre-calculated. ERA5 undergoes daily updates with a brief delay of approximately 5 days. In the event of significant anomalies detected in this initial release, known as **ERA5T**, the data may deviate from the final release by 2 to 3 months.

To facilitate usage, the data has been subjected to regridding, aligning it with a regular latitude-longitude grid. The grid spacing is set at 0.25 degrees for the reanalysis data and 0.5 degrees for the uncertainty estimates (0.5 and 1 degree, respectively, for ocean waves). There are four main subsets available, including hourly and monthly products, encompassing both pressure levels (about upper air fields) and single levels (encompassing atmospheric, ocean-wave, and land surface parameters). Specifically, data for the experiments in chapter 4 has been extracted from the *ERA5 hourly data on single levels from 1940 to present.* This subset is conveniently accessible online via spinning disk storage, ensuring rapid and effortless retrieval. Table 1.1 deeply describes the features considered in this work.

Table 1.1: Description of *ERA5 hourly data on single levels from 1940 to present variables* tackled in this work.

Name	**Units**	**Description**
100m u-component of wind	ms^{-1}	This parameter signifies the eastward component of the 100-meter wind, representing the horizontal air velocity moving in an eastward direction at an altitude of 100 meters above the Earth's surface, measured in meters per second. It is possible to combine this parameter with its northward counterpart to obtain the speed and direction of the horizontal 100-meter wind.
100m v-component of wind	ms^{-1}	This parameter signifies the northward component of the 100-meter wind, representing the horizontal air velocity moving in an northward direction at an altitude of 100 meters above the Earth's surface, measured in meters per second. It is possible to combine this parameter with its eastward counterpart to obtain the speed and direction of the horizontal 100-meter wind.

Continued on next page

Table 1.1 – continued from previous page

Name	**Units**	**Description**
Geopotential	m^2s^{-2}	This parameter denotes the gravitational potential energy of a unit mass at a particular Earth's surface location relative to mean sea level. Additionally, it signifies the amount of work needed to raise a unit mass from mean sea level to that specific point, counteracting the force of gravity. Surface geopotential height, often referred to as orography, can be calculated by dividing the surface geopotential by the Earth's gravitational acceleration, symbolized as g and equivalent to 9.80665 meters per second squared (m/s^2). It's noteworthy that this parameter remains constant over time.
Land-sea mask	dimensionless	This parameter measures the land fraction within a grid box, distinguishing it from ocean or inland water bodies such as lakes, reservoirs, rivers, and coastal regions. It is a dimensionless value that falls within the range of zero to one. Starting from ECMWF Integrated Forecasting System (IFS) cycles beginning with CY41R1, introduced in May 2015, grid boxes with values above 0.5 can contain a mix of land and inland water, excluding ocean. Grid boxes with values of 0.5 or lower exclusively represent water surfaces. In these cases, the extent of lake cover is taken into account to determine whether it's ocean or inland water. In IFS cycles before CY41R1, grid boxes with values above 0.5 consist entirely of land, while those with values of 0.5 or less are solely oceanic. In these earlier model cycles, there is no differentiation between ocean and inland water. Notably, this parameter remains constant over time.

Continued on next page

Table 1.1 – continued from previous page

Name	Units	Description
Total precipitation	m	This parameter represents the combined amount of liquid and frozen water, encompassing both rain and snow, that descends to the Earth's surface. It is the summation of two types of precipitation: large-scale precipitation and convective precipitation. Large-scale precipitation is the result of the cloud scheme within the ECMWF Integrated Forecasting System (IFS). This scheme models the formation and dissipation of clouds and large-scale precipitation based on predicted changes in atmospheric variables (such as pressure, temperature, and moisture) directly at the grid box or larger spatial scales. Convective precipitation arises from the IFS's convection scheme, which simulates convection at spatial scales smaller than the grid box. Note that this parameter does not account for fog, dew, or precipitation that evaporates in the atmosphere before reaching the Earth's surface. The accumulation period for this parameter varies depending on the extracted data. In the case of reanalysis, it covers the one-hour period ending at the validity date and time. For ensemble members, it spans the three hours leading up to the validity date and time. The units for this parameter are expressed as depth in meters of water equivalent, representing the depth the water would reach if evenly distributed across the grid box.

Continued on next page

Table 1.1 – continued from previous page

Name	Units	Description
Surface pressure	Pa	This parameter represents atmospheric pressure, which measures the force per unit area exerted by the atmosphere on the Earth's surface, encompassing land, sea, and inland water. It essentially quantifies the total weight of the air in a vertical column above a specific location on the Earth's surface. Surface pressure is frequently used in conjunction with temperature to calculate air density. Given the substantial variation in pressure with altitude, particularly in mountainous regions, identifying low and high-pressure weather systems can be challenging. As a result, mean sea level pressure is typically preferred for this purpose over surface pressure. The units for this parameter are denominated in Pascals (Pa). Surface pressure is commonly measured in hectoPascals (hPa), and occasionally it's presented in the older millibar units (mb), where 1 hPa equals 1 mb or 100 Pa.

1.2.2 NWCSAF

The **NWCSAF** [5], which stands for the Satellite Application Facility on Support to Nowcasting and Very Short Range Forecasting, operates as a consortium within the broader network of Satellite Application Facilities (**SAF**) under the European Organisation for the Exploitation of Meteorological Satellites (**EUMETSAT**). Its primary mission is to develop and provide two independent software packages, known as **NWC/GEO** and **NWC/PPS**, capable of swiftly generating meteorological data from both geostationary and polar satellite information in near real-time. These software packages are intended for local installation at the user's location, aiming to enhance nowcasting and very short-range forecasting, typically covering weather predictions for up to 12 hours ahead.

NWC/GEO offers a comprehensive range of seven meteorological products that encompass the entire Earth's observable regions via geostationary satellites. These products include:

- Four cloud-related products:
  - Cloud Mask (CMA)
  - Cloud Type (CT)
  - Cloud Top Temperature, Height, and Pressure (CTTH)
  - Cloud Microphysics (CMIC), which furnishes information on cloud phase, cloud effective radius, cloud optical thickness, and liquid/ice water content.
- Two sets of precipitation products:
  - Precipitating Clouds (PC and PC-Ph), which assess the likelihood of precipitation for cloudy pixels.
  - Convective Rainfall Rate (CRR and CRR-Ph), providing instantaneous and hourly precipitation values, with a focus on convective clouds. The second product in each pair utilizes information from the Cloud Microphysics (CMIC) product.
- Two convection-related products:
  - Convection Initiation (CI), estimating the probability of a cloudy pixel developing into a thunderstorm.
  - Rapidly Developing Thunderstorms (RDT), tracking and analyzing convective systems, highlighting various characteristics.
- One clear air product, Imaging Satellite Humidity and Instability (iSHAI), offering vertical profiles of humidity, temperature, and ozone for clear air pixels. It also provides data on precipitable water in both the total column and three vertical layers, along with several instability indices.

- Three conceptual model products:
  - Automatic Satellite Image Interpretation (ASII)
  - Tropopause Folding (ASII-TF)
  - Gravity waves (ASII-GW)
  - These products interpret satellite images using conceptual models, with the latter two focusing on turbulence-related phenomena.
- One extrapolation product, Extrapolated Imagery (EXIM), predicts future satellite imagery or other NWC/GEO products by considering kinematic extrapolation.
- One Atmospheric Motion Vector (AMV) product, High-Resolution Winds (HRW), calculating AMVs and trajectories by tracking cloud and humidity features across successive satellite images.

An **atmospheric motion vector (AMV)** represents the horizontal shift between two Earth locations observed in two satellite images, and it is determined by tracking a square segment of $n \times n$ pixels referred to as a *tracer*. A *trajectory* is the route traced by the same tracer across multiple successive satellite images. Tracers are defined based on particular features in visible, infrared, or water vapor images. These features could be related to cloudiness (in the case of cloudy tracers) or specific humidity characteristics in regions devoid of clouds, particularly in water vapor images (for clear air tracers). These tracers have a fixed size, measured in pixels. Initially, tracers are identified in an initial image and then monitored as they move in subsequent images, with their displacements between these images providing the necessary data to compute the atmospheric motion vectors (AMVs). The HRW product provides valuable data, including pressure level data, which helps position the calculated atmospheric motion vectors (AMVs) and trajectories in the vertical dimension. AMVs and trajectories are continuously computed around the clock, 24 hours a day.

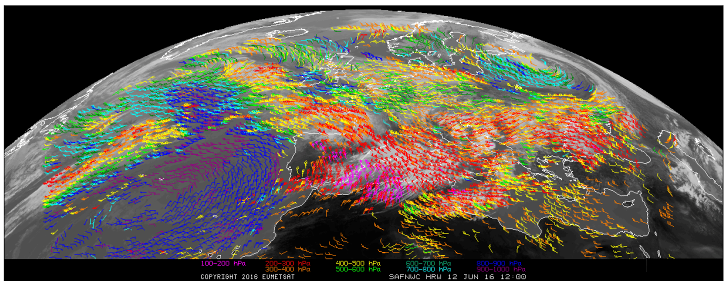


Figure 1.3: NWC/GEO-HRW output example of atmospheric motion vectors (AMVs) in the European and North Atlantic region. Image taken from [5].

When it comes to weather forecasting, traditional observations are limited in coverage, while satellite-based observations offer extensive global data at regular intervals. Therefore, the generation of atmospheric motion vectors (AMVs) from satellite images becomes a crucial source of wind information on a global scale, particularly in regions over the oceans and remote continental areas. In this context, the NWCSAF High-Resolution Winds (NWC/GEO-HRW) product proves to be highly valuable in both short-term weather prediction (nowcasting) and longer-term forecasting applications. It complements other available data sources and aids forecasters in several ways.

1.2.3 WeatherBench

WeatherBench [11] represents a very promising benchmark dataset for data-driven medium-range weather forecasting. It comprises data derived from the ERA5 archive that has been processed to facilitate the use of machine-learning models. It also proposes simple and clear evaluation metrics that enable a direct comparison between different methods, as well as baseline scores from simple linear regression techniques, deep learning models, as well as purely physical forecasting models.

The selection of available variables was guided by meteorological considerations. Geopotential, temperature, humidity, and wind represent prognostic state variables found in the majority of physical Numerical Weather Prediction (NWP) and climate models. Complementing the three-dimensional fields, the dataset encompasses various two-dimensional fields. For instance, the 2-meter temperature is a significant variable due to its direct relevance to human activities and susceptibility to the diurnal solar cycle. The 10-meter wind plays a vital role, especially in applications related to wind energy forecasts. Additionally, total cloud cover assumes a pivotal role in solar energy prediction. Lastly, top-of-atmosphere incoming solar radiation as an input variable can effectively encode diurnal variations. Furthermore, the dataset includes several potentially significant time-invariant fields, housed within the constants file. The initial trio of variables pertains to surface characteristics: the land-sea mask, a binary field distinguishing land and sea points; soil type, categorized into seven distinct classes; and orography, representing surface elevation. Additionally, two-dimensional fields containing latitude and longitude values at each point are provided, as they can prove valuable for the neural network to capture latitude-specific insights, including grid structure and the Coriolis effect. All the variables contained in the benchmark dataset are described in Table 1.2.

Evaluation is done for the years 2017 and 2018. **500 hPa geopotential** and **850 hPa temperature** have been chosen as primary verification fields. Geopotential at 500 hPa pressure, often abbreviated as **Z500**, is a commonly used variable that encodes the synoptic-scale pressure distribution. It is the standard verification variable for most medium-range NWP models. 850 hPa temperature has been chosen as a secondary verification field because temperature is a more impact-related variable. 850 hPa is usually above the planetary boundary layer and therefore not affected by diurnal variations but provides information about broader temperature trends, including cold spells and heat waves.

The **root mean squared error (RMSE)** has been chosen as the primary metric because it is easy to compute and mirrors the loss used for most ML applications. RMSE is defined as:

$$RMSE = \frac{1}{N_{forecasts}} \sum_{i}^{N_{forecasts}} \sqrt{\frac{1}{N_{lat}N_{lon}} \sum_{j}^{N_{lat}} \sum_{k}^{N_{lon}} L(j)(f_{i,j,k} - t_{i,j,k})^2}$$

where f is the model forecast and t is the ERA5 truth. $L(j)$ is the latitude weighting factor for the latitude at the jth latitude index:

$$L(j) = \frac{cos(lat(j))}{\frac{1}{N_{lat}} \sum_{j}^{N_{lat}} cos(lat(j))}.$$

Table 1.2: List of variables contained in the WeatherBench dataset. Table taken from [11].

Long name	**Short name**	**Description**	**Unit**	**Levels**
geopotential	z	Proportional to the height of a pressure level	m^2s^{-2}	13
temperature	t	Temperature	K	13
specific_humidity	q	Mixing ratio of water vapor	$kgkg^{-1}$	13
relative_humidity	r	Humidity relative to saturation	%	13
u_component_of_wind	u	Wind in x/longitude-direction	ms^{-1}	13
v_component_of_wind	v	Wind in y/latitude direction	ms^{-1}	13
vorticity	vo	Relative horizontal vorticity	$1s^{-1}$	13
potential_vorticity	pv	Potential vorticity	$Km^2kg^{-1}s^{-1}$	13
2m_temperature	t2m	Temperature at 2 m height above surface	K	1
10m_u_component_of_wind	u10	Wind in x/longitude-direction at 10 m height	$ms-1$	1
10m_v_component_of_wind	v10	Wind in y/latitude-direction at 10 m height	ms^{-1}	1
total_cloud_cover	tcc	Fractional cloud cover	(0–1)	1
total_precipitation	tp	Hourly precipitation	m	1
toa_incident_solar_radiation	tisr	Accumulated hourly incident solar radiation	Jm^{-2}	1
constants		*File containing time-invariant fields*		
land_binary_mask	lsm	Land-sea binary mask	(0/1)	1
soil_type	slt	Soil-type categories see	$text$	1
orography	orography	Height of surface	m	1
latitude	lat2d	2D field with latitude at every grid point		1
longitude	lon2d	2D field with longitude at every grid point		1

1.3 Generative models

Generative models are a class of machine learning models designed to generate new data samples that resemble a given dataset. These models have gained significant attention and popularity in various fields, including computer vision, natural language processing, and art generation. They're used for tasks like image generation, text generation, speech synthesis, and more.

There are several types of generative models, but two of the most well-known and widely used ones are **Variational Autoencoders (VAEs)** [31, 32, 33] and **Generative Adversarial Networks (GANs)** [34, 35]. VAEs are a type of generative model that combine ideas from autoencoders [36] and probabilistic graphical models. They consist of two main parts: an encoder and a decoder. The encoder maps the input data into a lower-dimensional latent space, while the decoder maps points in this latent space back to the data space. VAEs are trained to learn the probability distribution of data in the latent space, making it possible to generate new samples by sampling from this distribution. They are particularly useful for applications like image generation and data denoising.

GANs are another popular class of generative models that consist of two neural networks: a generator and a discriminator. The generator tries to generate fake data samples from random noise according to the desired distribution, while the discriminator aims to distinguish between real and generated (*fake*) data. During training, these two networks are in constant competition: the generator tries to generate data that is indistinguishable from real data, while the discriminator tries to get better at telling real from fake. The generator and discriminator are trained alternately, with each adversarial component being frozen during the training of its counterpart. The ultimate goal is for the generator to excel, producing samples that the discriminator cannot differentiate from genuine data. GANs are known for their ability to create highly realistic data samples, and they have been used for generating images, videos, and even text.

Other notable generative models include **Autoregressive Models**, generating data one element at a time, with each element being conditioned on the previous ones (e.g., PixelRNN and PixelCNN [37] models for image generation, and the GPT - Generative Pre-trained Transformer - series [38] for text generation), and **Flow-Based Models** [39, 40], focusing on learning the data distribution directly and used for density estimation and data generation. **Normalizing flows** [39] are a popular approach in this category.

Generative models have a wide range of applications beyond simply generating data. They can be used for data augmentation, style transfer, anomaly detection, and more. The choice of which generative model to use depends on the specific task and the nature of the data. Additionally, the field of generative modeling is continuously evolving, with new architectures and techniques emerging regularly to improve the quality and diversity of generated data.

1.3.1 Generative models for weather forecasting

A wealth of historical datasets is readily available for short and medium-range weather forecasting, and these datasets often adhere to a structured spatial grid format that closely resembles visual images. Each weather variable within these datasets can be effectively depicted as a map, and when taking the temporal dimension into account, they can be envisioned as dynamic sequences akin to videos.

The challenge of predicting the next frame in a video sequence is a well-established problem in image processing, and a range of generative models have demonstrated their effectiveness in this regard, as evidenced by studies such as [41, 42, 43, 44]. Consequently, it is not surprising that recent research efforts have gravitated toward the application of **deep neural network (DNN)** architectures to the domain of weather nowcasting, as exemplified by studies like [45, 46, 47, 48, 49, 50, 51, 52]. Notably, these models do not rely on explicit physical laws that describe atmospheric dynamics. Instead, they employ backpropagation-based learning methods to directly forecast weather patterns using observed data.

The majority of the aforementioned models are trained with the aim of minimizing the **log-likelihood** of their predictions, often quantified using metrics like **mean squared error (MSE)**. However, it is widely recognized in other domains of image processing that log-likelihood-based optimization, particularly when dealing with multimodal output, tends to produce predictions that are overly smoothed, introducing a degree of blurriness. This effect becomes more pronounced as the lead time for predictions increases

Rather than solely predicting the anticipated quantity of precipitation, an alternative approach involves characterizing its **probability distribution**, a core objective of generative modeling as highlighted in works like [53, 54]. Typically, this probability distribution is acquired through implicit learning, where a *generator* is trained to generate data that conforms to the desired distribution.

For several years, the dominant class of generative models has been Generative Adversarial Networks (GANs). In comparison to likelihood-based models like Variational Autoencoders (VAEs), GANs typically offer superior generative quality, albeit potentially at the expense of reduced sampling diversity [55]. GANs are known to be susceptible to a phenomenon known as **mode collapse** [56], in which the generator tends to output only a limited set of examples, sometimes even ignoring its noise input and generating identical outputs for certain inputs. This is important, as it prevents GANs from being suitable for ensemble techniques. In the realm of weather forecasting, Generative Adversarial Networks (GANs) have found applications in various domains, including downscaling [57, 58, 59], precipitation estimation from remote satellite sensors [60, 61], and disaggregation [62]. Notably, the **Deep Generative Models of Rainfall (DGMR)** [6], described in subsection 1.3.2, stands as a distinguished example of a generative nowcasting model based on GANs.

The conventional dominance of GANs in the realm of generative modeling has recently faced a formidable challenge in the form of **Denoising Diffusion Models (DDM)** [63], which will be deeply discussed in chapter 2.

1.3.2 An example of a generative model for precipitation nowcasting: DGMR

The **Deep Generative Models of Rainfall (DGMR)** [6] is a notable and advanced generative model designed specifically for rainfall nowcasting. It is highly regarded in the field of weather forecasting and has demonstrated impressive capabilities in generating accurate rainfall predictions.

DGMR is based on the GAN framework, a deep learning architecture consisting of two neural networks – a generator and a discriminator – that engage in an adversarial training process. The generator aims to produce realistic rainfall predictions, while the discriminator tries to distinguish between real and generated rainfall data. Furthermore, DGMR is often implemented as a **conditional GAN**, meaning that it takes additional information as input to condition its predictions. In the context of rainfall nowcasting, this additional information could include past rainfall observations, atmospheric data, or other relevant meteorological factors. By conditioning on this information, DGMR can produce contextually relevant rainfall forecasts.

To enhance the quality of its predictions, DGMR typically incorporates **regularization** techniques. These regularization terms are added to the GAN's loss function to encourage the model to generate rainfall forecasts that are consistent with the underlying physical processes and observed data. DGMR is often designed to produce high-resolution rainfall predictions, allowing it to capture fine-scale details in precipitation patterns. This is particularly valuable for applications such as flood prediction and monitoring.

DGMR is the result of ongoing research and development in the field of weather forecasting and machine learning. It represents a significant advancement in the use of generative models for weather-related tasks. DGMR and similar generative models are employed in various weather-related applications, including rainfall nowcasting, flood forecasting, and climate modeling.

Their ability to generate high-quality, spatiotemporally detailed rainfall predictions makes them valuable tools for improving our understanding of and preparedness for extreme weather events.

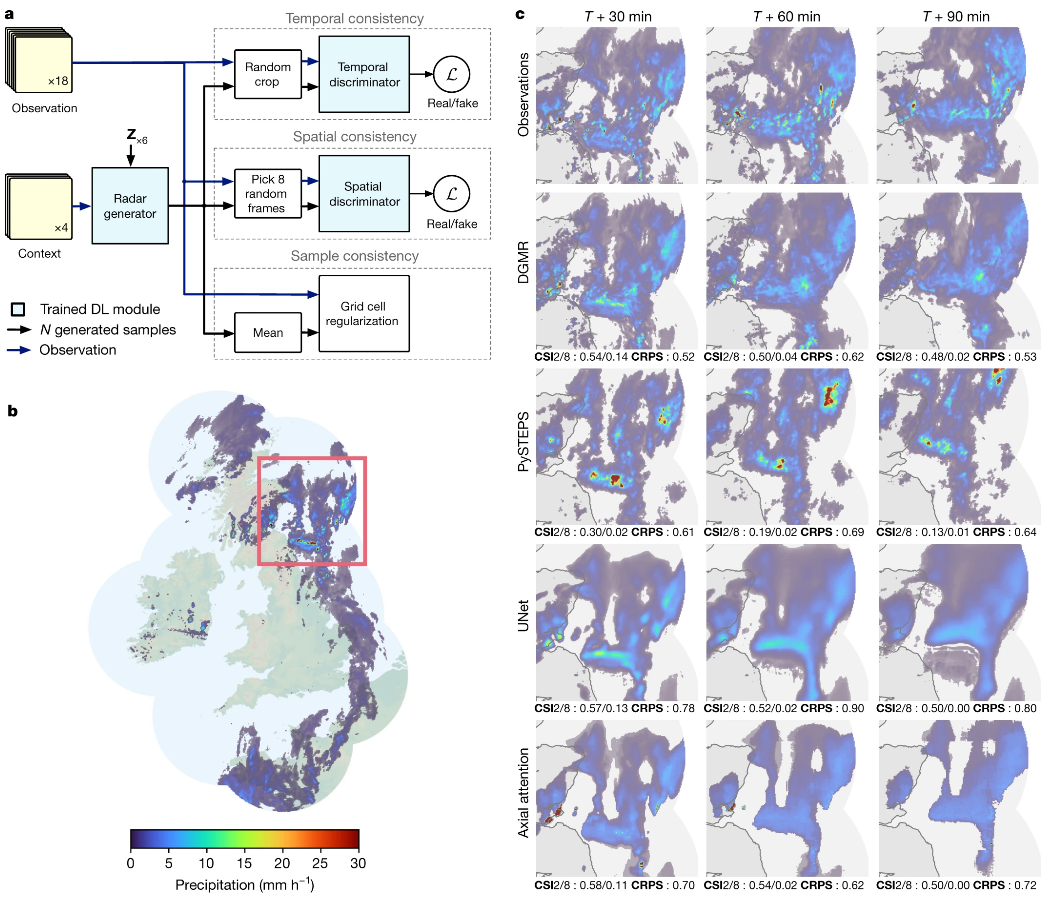


Figure 1.4: DGMR model overview and case study of performance on a challenging precipitation event starting on 24 June 2019 at 16:15 UK, showing convective cells over eastern Scotland. Image taken from [6].

Chapter 2

Generative diffusion models

This chapter aims to deeply describe Denoising Diffusion Models (DDM) and its most noticeable variants, namely Denoising Diffusion Probabilistic Models (DDPM) [63], Denoising Diffusion Implicit Models (DDIM) [64], and Latent Diffusion Models (LDM) [7]. DDM has emerged as a generative technique with distinctive characteristics that have played a pivotal role in numerous contemporary and widely-recognized applications [65, 66], spanning across domains such as video generation [67, 68]. These distinguishing attributes encompass exceptional generation quality, heightened sensitivity and adaptability to conditioning, a wide-ranging sampling capability, robust training stability, and commendable scalability [69, 70].

Essentially, a diffusion model leverages a single network to effectively eliminate noise from images, with the flexibility to parametrically adjust the level of noise to be removed. This network is subsequently employed

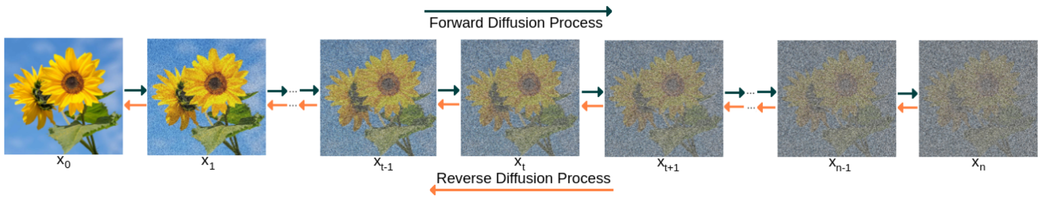


Figure 2.1: Forward (from left to right) and reverse (from right to left) diffusion process.

to produce novel samples by iteratively diminishing noise in a designated *noisy* image. This iterative process commences from an entirely random noise configuration and is conventionally known as **reverse diffusion**. Its objective is to effectively *invert* the direct diffusion process, where noise is incrementally added to the source image (refer to Figure 2.1).

Diffusion models represent a class of powerful probabilistic generative models renowned for their proficiency in capturing intricate, high-dimensional data distributions. These models have found extensive utility across diverse domains, including computer vision, natural language processing, and generative art. Fundamentally, diffusion models are rooted in the mathematical concept of a diffusion process, which characterizes the stochastic, continuous random movement of particles through time. This process simulates the dispersion or diffusion of a given quantity across space or time, with particles naturally gravitating from regions of high concentration to those of lower concentration. This gradual blending or mixing of the quantity defines the core principle.

In the realm of machine learning, diffusion models harness the principles of diffusion processes to elucidate data generation. Rather than directly sampling data points from a fixed distribution, these models ingeniously iterate to transform an initially simple distribution—often a well-known one like a Gaussian or uniform distribution—into the desired complex data distribution. The central concept revolves around a sequence of **diffusion steps**, wherein each step updates the probability distribution of the data. This is achieved by incrementally introducing Gaussian noise to the existing data samples and continuously refining them.

From a mathematical standpoint, when we start with a distribution represented as $q(x_0)$ responsible for generating the data, the objective of generative models is to identify a parameter vector θ. This parameter vector is utilized to shape the distribution $p_\theta(x_0)$, which is characterized by a neural network. The goal here is for this parameterized distribution, $p_\theta(x_0)$, to closely approximate and model the original data distribution $q(x_0)$.

2.1 Denoising Diffusion Probabilistic Models (DDPM)

Denoising Diffusion Probabilistic Models (DDPM) [63] posit that the generative distribution $p_\theta(x_0)$ follows a specific form defined as:

$$p_\theta(x_0) = \int p_\theta(x_{0:T}) dx_{1:T} \tag{2.1}$$

Here, the time horizon extends to $T > 0$, and $p_\theta(x_{0:T})$ can be further expressed as:

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{2.2}$$

In this formulation, DDPM characterizes the generative process by modeling the joint distribution over a sequence of time steps, where $p_\theta(x_T)$ represents the initial distribution, and the subsequent terms capture the conditional transitions from x_t to x_{t-1} as the process evolves.

Training in diffusion models traditionally relies on a **variational lower bound of the negative log-likelihood**:

$$\begin{aligned}
&- \log p_\theta(x_0) \\
&\leq - \log p_\theta(x_0) + D_{\mathrm{KL}}(q(x_{1:T}|x_0) \| p_\theta(x_{1:T}|x_0)) \\
&= - \log p_\theta(x_0) + \mathbb{E}_q\Big[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})/p_\theta(x_0)} \Big] \\
&= - \log p_\theta(x_0) + \mathbb{E}_q\Big[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} + \log p_\theta(x_0) \Big] \\
&= \mathbb{E}_q\Big[\log q(x_{1:T}|x_0) - p_\theta(x_{0:T}) \Big] = \mathcal{L}(\theta)
\end{aligned} \tag{2.3}$$

What sets diffusion models apart from typical latent variable models like Variational Autoencoders (VAEs) [31, 32, 33] is that they employ a fixed, non-trainable inference procedure denoted as $q(x_{1:T}|x_0)$. Moreover, the latent variables in diffusion models exhibit relatively high dimensionality, which usually matches the dimensions of the visible space.

2.2 Denoising Diffusion Implicit Models (DDIM)

In the specific case of **Denoising Diffusion Implicit Models (DDIM)** [64], the authors introduce a non-Markovian diffusion process defined as:

$$q_\sigma(x_{1:T}|x_0) = q_\sigma(x_T|x_0)\prod_{t=2}^{T} q_\sigma(x_{t-1}|x_t, x_0) \tag{2.4}$$

Here, the term $q_\sigma(x_T|x_0)$ is described by a Gaussian distribution:

$$q_\sigma(x_T|x_0) = \mathcal{N}(x_T|\sqrt{\alpha_T}x_0, (1-\alpha_T)\cdot I) \tag{2.5}$$

Additionally, the conditional distribution $q_\sigma(x_{t-1}|x_t, x_0)$ takes the form:

$$q_\sigma(x_{t-1}|x_t, x_0) = \mathcal{N}\Big(x_{t-1}\Big|\mu_{\sigma_t}(x_0, \alpha_{t-1}); \sigma_t^2 \cdot I\Big) \tag{2.6}$$

with

$$\mu_{\sigma_t}(x_0, \alpha_{t-1}) = \sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot \tfrac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}.$$

The choice of $q(x_{t-1}|x_t, x_0)$ is strategically made to fulfill two critical aspects of the DDPM diffusion process: the Gaussian nature of $q(x_{t-1}|x_t, x_0)$ when conditioned on x_0, and the ability to recover the same marginal distribution as in DDPM, where $q_\sigma(x_t|x_0) = \mathcal{N}(x_t|\sqrt{\alpha_t}x_0; (1-\alpha_t)\cdot I)$. This property allows us to represent x_t as a linear combination of x_0 and a noise variable $\epsilon_t \sim \mathcal{N}(\epsilon_t|0; I)$:

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t. \tag{2.7}$$

Next, our task is to define a trainable generative process denoted as $p_\theta(x_{0:T})$, where the conditional distribution $p_\theta(x_{t-1}|x_t)$ is crafted to incorporate the structure from $q_\sigma(x_{t-1}|x_t, x_0)$. The concept here is that when provided with a noisy observation x_t, the process begins by predicting x_0 and then employs this prediction to derive x_{t-1} according to equation 2.6.

In practical terms, a neural network denoted as $\epsilon_\theta^{(t)}(x_t, \alpha_t)$ is trained to effectively map a given pair of inputs, x_t and α_t (representing the noise rate), into an estimate of the noise component, ϵ_t. This estimated noise, when added to x_0, facilitates the construction of the next time step, x_t. Consequently, the **conditional distribution** $p_\theta(x_{t-1}|x_t)$ is approximated as a Dirac delta function $\delta_{f_\theta^{(t)}}$, where:

$$f_\theta^{(t)}(x_t, \alpha_t) = \frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, \alpha_t)}{\sqrt{\alpha_t}}. \tag{2.8}$$

Using $f_\theta^{(t)}(x_t, \alpha_t)$ as an approximation of x_0 at timestep t, x_{t-1} is subsequently calculated as follows:

$$x_{t-1} = \sqrt{\alpha_{t-1}} \cdot f_\theta^{(t)}(x_t, \alpha_t) + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta(x_t, \alpha_t) \tag{2.9}$$

.

Regarding the **loss** function, the term in equation 2.3 can be further decomposed into the sum of the following terms [71]:

$$L_\theta = L_T + L_{t-1} + \cdots + L_0 \tag{2.10}$$

where

$$\begin{aligned} L_T &= D_{\mathrm{KL}}(q(x_T|x_0) \parallel p_\theta(x_T)) \\ L_t &= D_{\mathrm{KL}}(q(x_t|x_{t+1}, x_0) \| p_\theta(x_t|x_{t+1})) \\ &\qquad\qquad\qquad \text{for } 1 \le t \le T-1 \\ L_0 &= -\log p_\theta(x_0|x_1) \end{aligned}$$

This breakdown of the loss function provides a more granular perspective on the optimization process.

All the aforementioned distributions take the form of Gaussians, facilitating the calculation of their KL divergences in a closed-form manner, following a Rao-Blackwellized approach. After a series of manipulations, we arrive at the following formulation:

$$L_t = \mathbb{E}t \sim [1,T], x_0, \epsilon_t \Big[\gamma_t |\epsilon_t - \epsilon\theta(x_t, t)|^2\Big] \tag{2.11}$$

This expression can be interpreted as the **weighted mean squared error** between the predicted noise and the actual noise at time t. In practice, the weighting parameters are often omitted, as experimental evidence suggests that the training process tends to perform better without them. The pseudocode for training and sampling is given in the following Algorithms.

Algorithm 1 Training

```
1: repeat
2:     $x_0 \sim q(x_0)$
3:     $t \sim$Uniform(1,..,T)
4:     $\epsilon \sim \mathcal{N}(0; I)$
5:     $x_t = \sqrt{\alpha_t} x_b + \sqrt{1-\alpha_t}\epsilon$
6:     Backpropagate on $||\epsilon - \epsilon_\theta(x_t, \alpha_t)||^2$
7: until converged
```

Sampling is an iterative process, starting from a purely noisy image $x_T \sim \mathcal{N}(0, I)$. The denoised version of the image at timestep t is obtained using equation 2.9.

Algorithm 2 Sampling

```
1: $x_T \sim \mathcal{N}(0, I)$
2: for $t = T, ..., 1$ do
3:     $\epsilon = \epsilon_\theta(x_a, x_t, \alpha_t)$
4:     $\tilde{x}_0 = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\epsilon)$
5:     $x_{t-1} = \sqrt{\alpha_{t-1}}\tilde{x}_0 + \sqrt{1-\alpha_{t-1}}\epsilon$
6: end for
```

2.3 Latent Diffusion Models (LDM)

The **Latent Diffusion Model (LDM)**, introduced by Rombach et al. in 2022 [7], offers an innovative approach to image processing. Unlike conventional models that operate in pixel space, LDM conducts the diffusion process in the latent space. This choice significantly reduces the training cost and enhances inference speed. The underlying motivation for this approach stems from the insight that the majority of an image's information contributes to perceptual details, while the fundamental semantic and conceptual composition remains intact even after aggressive compression.

LDM achieves its capabilities through a two-step process. Initially, it employs an autoencoder to eliminate pixel-level redundancy, effectively trimming away superfluous information. Subsequently, LDM utilizes a diffusion process on the learned latent space to manipulate and generate semantic concepts. This dual approach effectively decomposes the tasks of perceptual compression and semantic compression, harnessing the power of generative modeling to enhance the quality and efficiency of image processing.

The perceptual compression process in the model relies on an autoencoder architecture. The encoder ε takes the input image $x \in \mathbb{R}^{H\times W\times 3}$ and compresses it into a smaller 2D latent vector $z = \varepsilon(x) \in \mathbb{R}^{h\times w\times c}$ with down-

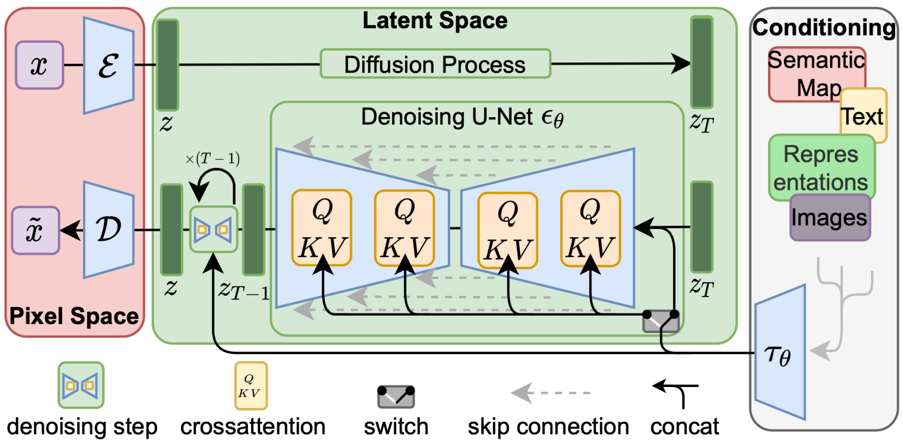


Figure 2.2: The architecture of latent diffusion model. Image taken from [7].

sampling rate $f = H/h = W/w = 2^m$, where m is a natural number. Here, h and w are the dimensions of the latent space, and c represents the number of channels in the latent representation. Subsequently, a decoder $\mathcal{D}$ is employed to reconstruct the images from the latent vector, yielding $\tilde{x} = \mathcal{D}(z)$.

To enhance the training of the autoencoder and avoid arbitrarily high variance in the latent spaces, the authors explore two types of regularization techniques: **KL-reg**, applying a small KL penalty to encourage the learned latent space to approximate a standard normal distribution, similar to the concept in Variational Autoencoders (VAE) [72] and **VQ-reg**, in which a vector quantization layer is integrated within the decoder, reminiscent of the VQVAE architecture [73]; however, in LDM, the quantization layer is absorbed by the decoder. Both these regularization techniques contribute to the effective training and control of the autoencoder's latent space.

The diffusion and denoising processes take place on the latent vector z. The denoising model employed is a time-conditioned U-Net architecture, which is augmented with a **cross-attention** mechanism to accommodate flexible conditioning information for image generation. This conditioning information can include class labels, semantic maps, or blurred variants of an image. The design of the model is such that it integrates different modalities of information using the cross-attention mechanism. Each type of conditioning information is paired with a domain-specific encoder τ_θ, which projects the conditioning input y into an intermediate representation ($\tau_\theta(y) \in \mathbb{R}^{M \times d_\tau}$) that can be effectively utilized by the cross-attention component:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}}) \cdot V$$

where

$$Q = W_Q^{(i)} \cdot \varphi_i(z_i), K = W_K^{(i)} \cdot \tau_\theta(y), V = W_V^{(i)} \cdot \tau_\theta(y)$$

and

$$W_Q^{(i)} \in \mathbb{R}^{d \times d_\epsilon^i}, W_K^{(i)}, W_V^{(i)} \in \ \times{}_{\tau}, \varphi_i(z_i) \in \mathbb{R}^{N \times d_\epsilon^i}, \tau_\theta(y) \in \mathbb{R}^{M \times d_\tau}.$$

Recently, the principles of LDMs have successfully been extended to the temporal domain [74], allowing for the generation of coherent video sequences.

2.4 Conditioning

The process of generating data often necessitates a means of exerting control over the sample creation process to influence the final output. This procedure is commonly referred to as **conditioned** or **guided** diffusion. Diverse strategies have been developed to seamlessly incorporate image and/or text embeddings into the diffusion process, thereby facilitating guided generation. In mathematical terms, **guidance** entails the conditioning of a prior data distribution, typically denoted as $p(x)$, with specific constraints, such as class labels or image/text embeddings. This conditioning yields the emergence of a conditional distribution, represented as $p(x|y)$.

To transform a diffusion model, denoted as p_θ, into a conditional diffusion model, we introduce conditioning information, denoted as y, at each step of the diffusion process, yielding the following formulation:

$$p_\theta(x_{0:T}|y) = p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t, y) \tag{2.12}$$

The learning of this distribution typically follows one of two approaches. The first approach relies on an auxiliary classifier, akin in spirit to AC-GANs [75]. The second approach, in contrast, operates without a classifier, offering an alternative methodology for modeling conditional diffusion.

The concept underpinning classifier guidance is as follows: the objective is to learn the gradient of the logarithm of the conditional density $p_\theta(x_t|y)$. By applying Bayes' rule, this can be expressed as:

$$\nabla_{x_t} \log p_\theta(x_t|y) = \nabla_{x_t} \log\left(\frac{p_\theta(y|x_t)\cdot p_\theta(x_t)}{p_\theta(y)}\right) \tag{2.13}$$

Given that the gradient operator solely pertains to x_t, the term $p_\theta(y)$ can be eliminated; after simplification, we arrive at:

$$\begin{aligned}\nabla_{x_t} \log p_\theta(x_t|y) =& \nabla_{x_t} \log p_\theta(x_t) \\ &+ s \cdot \nabla_{x_t} \log p_\theta(y|x_t)\end{aligned} \tag{2.14}$$

Here, the scalar term s assumes the role of modulating the strength of the guidance component. As elucidated in [69], one approach to guide the diffusion process during generation entails the utilization of a classifier, denoted as $f_\phi(y|x_t,t))$. This method involves the training of a classifier $f_\phi(y|x_t,t)$ using a noisy image x_t to predict its corresponding class label y. Subsequently, the gradient $\nabla_x \log f_\phi(y|x_t)$ can be harnessed to steer the diffusion sampling process towards the targeted conditioning information y through adjustments to the noise prediction. It's worth noting that this technique is particularly well-suited for scenarios involving discrete labels, and as such, further elaboration is omitted.

The concept of conditioned diffusion, devoid of reliance on an external classifier, has been extensively explored in [76]. This approach involves the training of both a conditional diffusion model, denoted as $\epsilon_\theta(x_t,t,y)$, and an unconditional model, denoted as $\epsilon_\theta(x_t,t,0)$. In many cases, the same neural network architecture can serve both models: during training, the class label y is randomly set to 0, thereby exposing the model to both conditional and unconditional scenarios. The estimated noise, represented as $\hat{\epsilon}_\theta(x_t|t,y)$ at time step t, subsequently emerges as a judiciously weighted combination of the conditional and unconditional predictions:

$$\hat{\epsilon}_\theta(x_t,t,y) = \epsilon_\theta(x_t,t,y) + s \cdot \epsilon_\theta(x_t,t) \tag{2.15}$$

Chapter 3

Generative diffusion models for precipitation nowcasting

The application of diffusion models to weather forecasting presents a natural and compelling avenue of exploration, with multiple teams independently addressing this challenge across various datasets. In a notable instance, as described in [77], diffusion models have been effectively employed for a downscaling task. Moreover, in [10], a diffusion model was meticulously trained on a dataset originating from the MeteoSwiss operational radar network [78, 79] and subsequently evaluated using data derived from the radar composite of the German Weather Service (DWD) spanning April to September 2022 [80]. These findings were subjected to comparison against a GAN-based Deep Generative Models of Rainfall (DGMR) (described in subsection 1.3.2) and a statistical model, PySTEPS (described in subsection 1.1.2), revealing discernible enhancements in both accuracy and diversification.

The purpose of this chapter is to present a novel model, the **Generative Ensemble Diffusion (GED)**. This model is rooted in the Denoising Diffusion Implicit Model (DDIM) described in section 2.2 and is trained on a sequence of rainfall data, enriched with additional meteorological features.

GED operates by generating multiple outputs, which are subsequently amalgamated into a final prediction using a **U-Net** architecture. This amalgamation process facilitates the creation of a comprehensive precipitation forecast, capitalizing on the DDIM's aptitude for modeling the probability distribution of weather data and the U-Net's prowess in extracting salient features.

Our exploration begins with an introduction to the distinct components that constitute the GED model, elucidating their roles within the broader framework. This discussion culminates in an examination of the overarching structure of the model. For a comprehensive analysis contrasting the architecture proposed in [10] with the GED model, refer to section 3.3.

3.1 The DDIM architecture

As we have seen in chapter 2, **diffusion models** essentially function as iterative denoising algorithms, with their central trainable component being the denoising network, represented as $\epsilon_\theta(x_t, \alpha_t)$. This network takes as input the noisy images, denoted as x_t, along with a corresponding noise variance, α_t, and aims to accurately estimate the level of noise affecting the image. The training of this underlying denoising network follows a conventional procedure. Initially, a sample, x_0, is drawn from the dataset and subjected to a predetermined amount of random noise. The network is then tasked with the responsibility of estimating the noise content within these perturbed images.

3.1.1 Denoising

The denoising network consists of a **U-Net** architecture, which stands as one of the most prevalent choices for denoising tasks [81, 82, 83, 84]. Notably, the U-Net architecture is frequently employed in diffusion models [85]. Originally devised for semantic segmentation purposes [8], the U-Net has garnered widespread acclaim and found utility across a wide spectrum of image manipulation tasks.

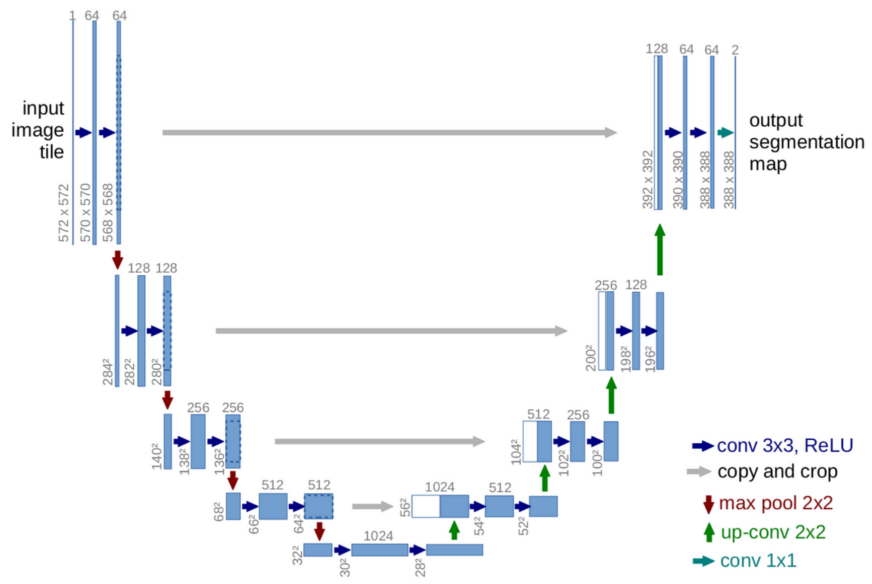


Figure 3.1: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. Image taken from [8].

The network structure, shown in Figure 3.1, consists of a downsampling sequence of layers, succeeded by an upsampling sequence, all while incorporating skip connections linking layers of equivalent dimensions. Typically, the configuration of the U-Net is defined by specifying the number of downsampling blocks and the channel count for each block. The upsampling component mirrors a symmetrical pattern, and the spatial dimensions align with the image resolution, which, in our case, is set at 96x96.

The entire structure of a U-Net can be succinctly encoded in a single list, as exemplified by [32, 64, 96, 128]. This list serves a dual purpose, denoting both the number of downsampling blocks (in this instance, 4) and the associated channel counts, which typically increase as the spatial dimensions decrease. For the experiments in chapter 4, a U-Net configuration of [64, 128, 256, 384] has been used, which empirically proved to be the most effective.

To enhance the U-Net's sensitivity to the noise variance, α_t is introduced as an input. Subsequently, an ad-hoc sinusoidal transformation to embed this noise variance is employed. This transformation involves dissecting the value into a set of frequencies, a technique reminiscent of positional encodings in Transformers [86]. The embedded noise variance is then vectorized and concatenated with the noisy images along the channel axes before being fed into the U-Net. This strategic addition bolsters the network's ability to discern noise levels, a critical factor for achieving superior performance. Sinusoidal embeddings are implemented using a Lambda layer.

3.1.2 Conditioning

Conditioning the model is imperative to steer the diffusion process toward a forecast determined by the known prior weather conditions. In the proposed approach, conditioning is achieved in a classifier-free manner [76] by directly appending the conditioning frames to the noisy images along the channel axis.

In practical terms, the model $\epsilon_\theta(x_t, t, y)$ receives as input the noisy images, denoted as $x_t = \{r_1, r_2, r_3\}$, where each r_h represents the future prediction of rain precipitation for h hours ahead. The conditioning information, $y = \{r_{-8..0}, u_{-1,0}, v_{-1,0}, lsm, geopot\}$, encompasses the previous 8 hours of precipitation data $r_{-11..0}$, the prior two hours' data for both wind components $u_{-1,0}$ and $v_{-1,0}$, along with two static maps characterizing the land-sea mask (lsm) and geopotential ($geopot$).

Figure 3.2 illustrates how conditioning is implemented by stacking this additional information alongside the channel axis in the denoising network. This implementation seamlessly integrates the conditioning information into the U-Net architecture. Specifically, each temporal slice in the input data is treated analogously to a color channel in an RGB image. By applying 2D convolutions across these temporal slices independently, the model can extract sufficient frame-level temporal features to generate a sequence that aligns coherently with the past frames used for conditioning.

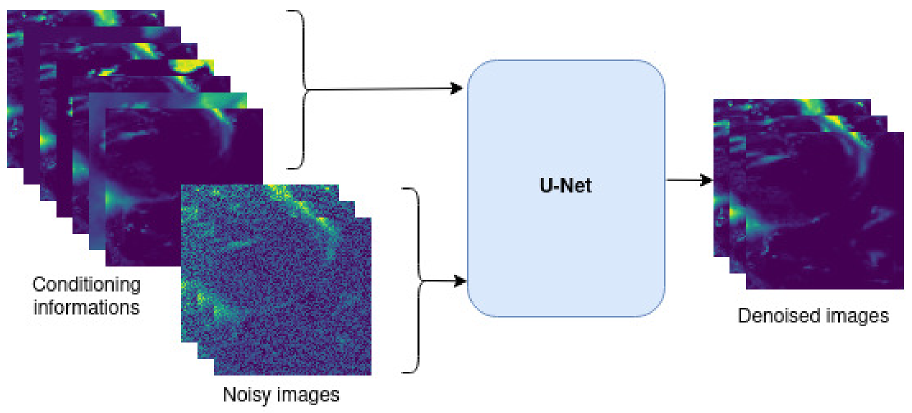


Figure 3.2: Conditioning is implemented by stacking additional information alongside the channel axis in the denoising network.

For instance, when training with a batch size of 16, the input data provided to the denoising network would have the shape [16, 96, 96, 17], with the last dimension encompassing both the conditioning information and the noisy images. In contrast, the output would consist solely of the denoised 3 frames, resulting in a shape of [16, 96, 96, 3].

Notably, this training process has demonstrated consistent success in achieving temporal conditioning solely with conditioned instances. This stands in contrast to some examples in the literature, which required alternating between conditioned and non-conditioned training instances, followed by a weight mixing stage, as described in [87].

3.2 A novel approach: Generative Ensemble Diffusion (GED)

The proposed **Generative Ensemble Diffusion (GED)** approach leverages the power of diffusion to integrate the inherent probability distribution of meteorological patterns, which is then used to synthesize a probable precipitation prediction. Image samples generated through the diffusion process, in line with the generative traits of the model, yield a highly diverse set of outputs despite sharing identical conditioning information. Based on the assumption that the diffusion model captures the stochastic essence of weather dynamics, a prediction can be derived from an ensemble of possible outcomes.

The proposed methodology shares a fundamental alignment with the ensemble post-processing techniques commonly employed in Numerical Weather Prediction models. As covered in subsection 1.1.2, ensemble post-processing involves the statistical refinement of a set of multiple weather forecasts generated from slightly different initial conditions, offering a spectrum of potential weather outcomes and serving as an estimation of forecast uncertainty.

Ensemble post-processing methods encompass a wide range of techniques, including statistical approaches like linear regression and distributional regression [88, 89], as well as more advanced machine learning algorithms such as QRF [90] and EMOS-GB [91]. In recent years, there has been a significant research focus on neural methods for ensemble post-processing, with several notable successful examples [92, 93]. These methods aim to enhance the accuracy and reliability of weather forecasts by combining information from multiple model runs, effectively mitigating uncertainties in predictions.

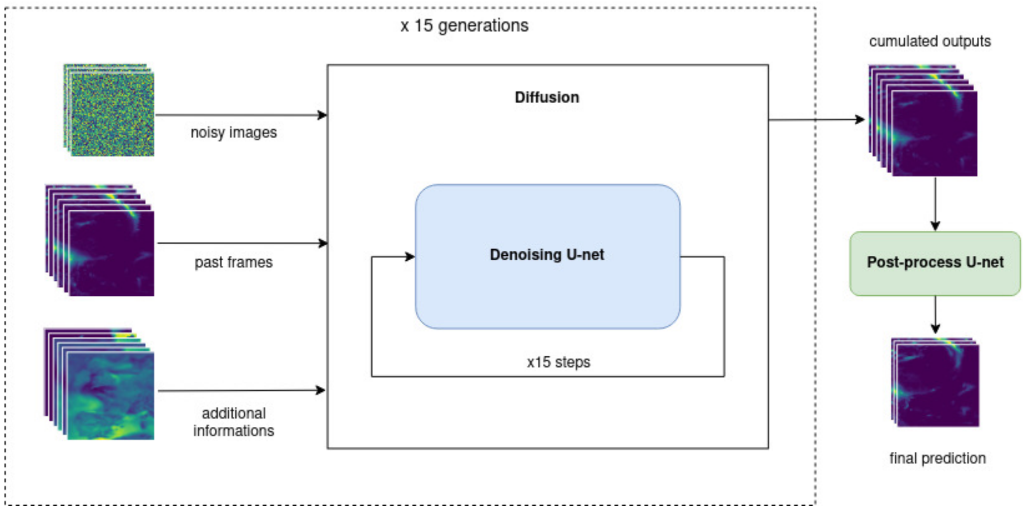


Figure 3.3: Generative Ensemble Diffusion (GED) prediction structure, showing the multiple denoising cycles and the final post-processing step.

Two distinct approaches have been proposed for implementing ensemble predictions: one based on a straightforward statistical method and the other employing a neural model. The first approach entails running multiple diffusion generations iteratively (a process that can be parallelized across the batch dimension for significant speedup, as noted in [94]). Subsequently, the mean of these generated images is calculated. This technique effectively consolidates the probability distribution of images into an averaged outcome, resulting in a more precise forecast. Experimental findings (section 4.4) demonstrate that computing the mean of multiple generations consistently yields superior results compared to using a single diffusion generation. Thus, this approach harnesses the diversity of potential outcomes to generate a more accurate and robust prediction. You can reference Figure 3.3 for clarity.

Instead of merely computing a mean, the second approach employs a more sophisticated module for prediction synthesis, a technique commonly employed in the literature [24, 25]. Specifically, a U-Net architecture is utilized to amalgamate the generated outcomes into a more plausible prediction. This approach not only facilitates more informed decision-making in the integration of outputs but also offers the opportunity to add a post-processing layer specifically trained on the target image. This stands in contrast to the diffusion model, which is trained on the noise differences of individual diffusion steps. Experimental results (section 4.4) have consistently validated the effectiveness of this methodology, demonstrating its superiority over both simple diffusion and diffusion ensemble with the mean method.

3.3 Concurrent work

At the time of writing, another notable work addressing precipitation nowcasting using diffusion models is presented in [10]. However, comparing the two architectures poses a significant challenge due to the substantial differences in spatiotemporal scales within the data they handle.

In [10], the authors work with time steps of 5 minutes, employing 4 time steps (equivalent to 20 minutes) of precipitation data as input while predicting precipitation up to 20 time steps (or 100 minutes) into the future. Furthermore, the geographical scale differs considerably, with radar signals collected at a 1 km resolution covering a rectangular area spanning 710 km in the east–west direction and 640 km in the north–south direction, encompassing all of Switzerland and some adjacent regions.

In the mentioned work, the network is trained on 256x256 pixel images, corresponding to a geographical area of 256 km^2. To manage computational costs effectively, they adopt the concept of the **Latent Diffusion Model (LDM)** described in section 2.3, popularized by Stable Diffusion [95], where the diffusion process operates within a latent variable space mapped to the physical pixel space through an autoencoder.

The diffusion model employed in their work exhibits several distinctions from GED, starting with the number of denoising iterations, which is set at 50 as opposed to our 15 iterations. Moreover, their Denoiser incorporates a forecaster stack based on **Adaptive Fourier Neural Operators (AFNOs)** [96][97] for conditioning the model. This conditioning involves temporal cross-attention to map between input and output time coordinates and a distinct denoiser stack also founded on AFNOs, facilitating cross-attention simulations. However, the exact significance of these modules remains undocumented, as no ablation study was conducted. For an overview of the whole structure of the model, reference to Figure 3.4.

Other related works in precipitation nowcasting, particularly those based on **Convolutional Neural Networks (CNNs)**, often incorporate **3D convolutions** or utilize alternative conditioning methods like **Recurrent Neural Networks (RNNs)** or **Long Short-Term Memory networks (LSTMs)** to explicitly model temporal connections [98, 99, 100, 101]. In recent years, multiple publications have showcased promising results by treating timesteps as multiple channels within the network architecture, effectively achieving temporal prediction with **2D convolutions** [102, 103].

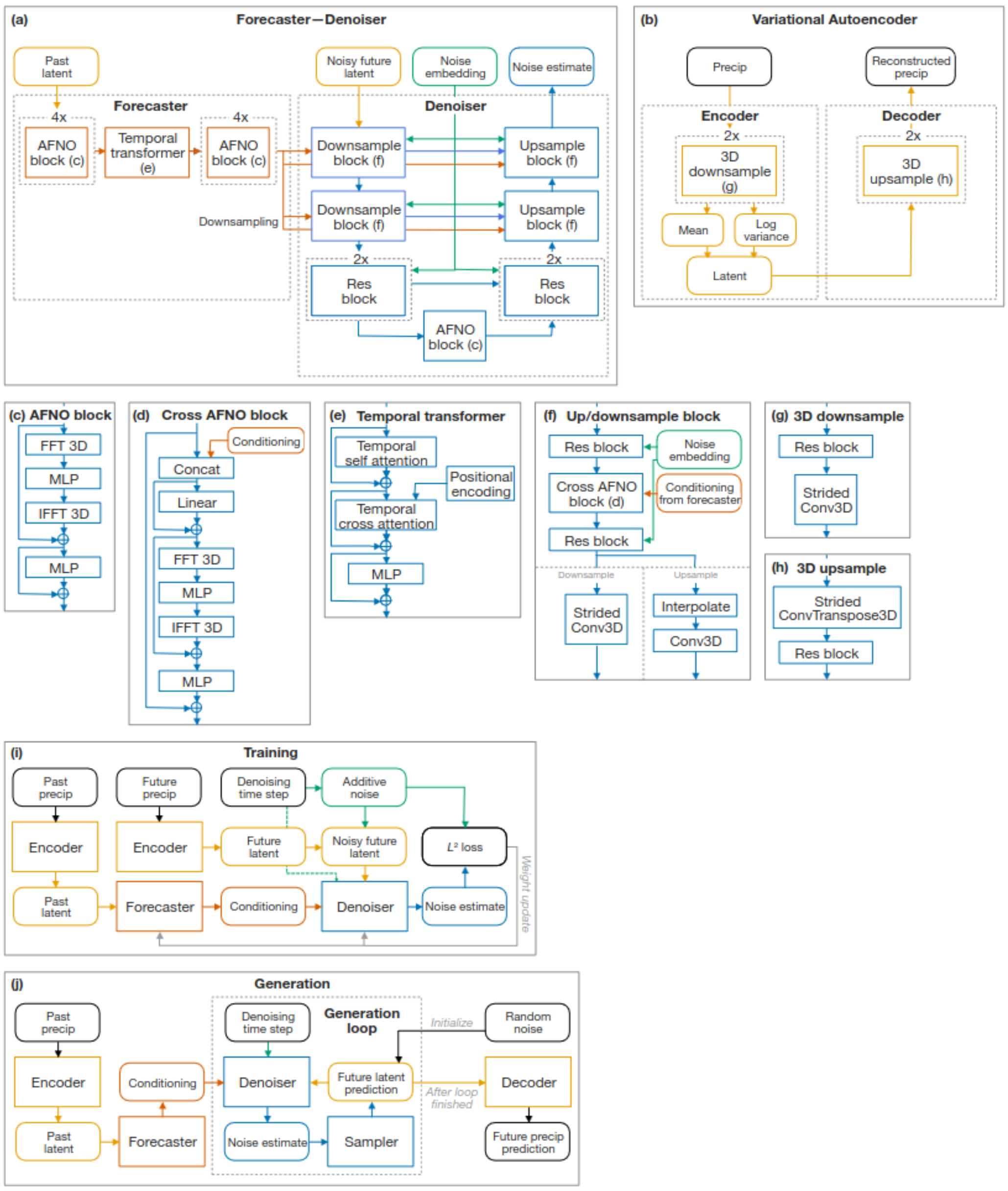


Figure 3.4: An overview of the LDCast neural networks. (a) The forecaster and denoiser stacks. (b) The VAE used to transform precipitation sequences to the latent space. (c)–(h) The layer blocks used in the network diagrams. (i) The training procedure. (j) The forecast generation procedure. *Conv* denotes convolution. *MLP* (multilayer perceptron) is a block consisting of a linear layer, activation function and another linear layer. *Res block* denotes a ResNet-type residual block [9]; the noise embedding is added to the input of the block. Image taken from [10].

In the proposed diffusion model, temporal data is handled using 2D convolutions like processing different color channels in image processing. Specifically, each temporal slice in the input data is treated analogously to a color channel in an RGB image. By independently applying 2D convolutions across these temporal slices, akin to the processing of various color channels, the model successfully matches the performance of competing 3D CNN models.

Chapter 4

Experiments

This chapter aims to compare the proposed **Generative Ensemble Diffusion (GED)** model introduced in chapter 3 with the **Weather Fusion UNet (WF-UNet)** model introduced in [98]. The comparison entails incorporating precipitation and wind speed variables as inputs in the learning process. The chosen dataset spans six years of precipitation and wind radar images, ranging from January 2016 to December 2021, covering 14 European countries. This dataset features a temporal resolution of 1 hour and a spatial resolution of 31 square km, relying on the traditional **ERA5** dataset [30].

When assessed using the conventional **Mean Squared Error** metric, the diffusion model consistently demonstrates superior predictive accuracy compared to the WF-UNet model.

4.1 Dataset description and preprocessing

To evaluate the performance of the model, experiments have been conducted using the same dataset and setup as outlined in previous studies such as [98, 100, 99], with a specific focus on precipitation nowcasting. The chosen dataset is a subset of **ERA5** [30], a state-of-the-art global atmospheric reanalysis dataset developed by the **European Centre for Medium-Range Weather Forecasts (ECMWF)**.

As covered in subsection 1.2.1, ERA5 provides a comprehensive numerical representation of Earth's recent climate history, spanning several decades and featuring global coverage at a high spatial resolution of approximately 31 kilometers. It offers hourly estimates of various atmospheric, land, and oceanic climate variables, including but not limited to temperature, precipitation, humidity, wind speed and direction, and sea surface temperature.

This dataset is the result of a sophisticated and consistent data assimilation system that combines millions of diverse observations with intricate Earth system modeling. The integration of these components produces a coherent and reliable dataset that is widely respected and extensively used across a range of scientific disciplines. Its applications extend to weather forecasting, climate research, hydrological studies, energy production prediction, and numerous policy-related endeavors, highlighting its importance in various fields.

The primary focus in this study is precipitation nowcasting, with the key target variable being **Total Precipitation**. Total Precipitation is defined as the accumulated amount of liquid and frozen water that reaches the Earth's surface, encompassing both rain and snowfall. For this investigation, a specific region of interest has been delineated within a geographical rectangle. This region spans from latitude -12° to latitude 12° in the north-south direction and from longitude 36° to longitude 60° in the east-west direction. The images obtained from this region cover a substantial portion of Western Europe, partially encompassing 14 different countries.

The dataset comprises observations collected over six years, spanning from 2016 to 2021, with measurements taken at hourly intervals. Each data point in the dataset has a dimension of 96×96, with each value representing the depth of fallen water. This depth measurement corresponds to the amount of water equivalent to the depth in meters that would result if the collected precipitation were uniformly spread over a grid box measuring 31 square kilometers in size.

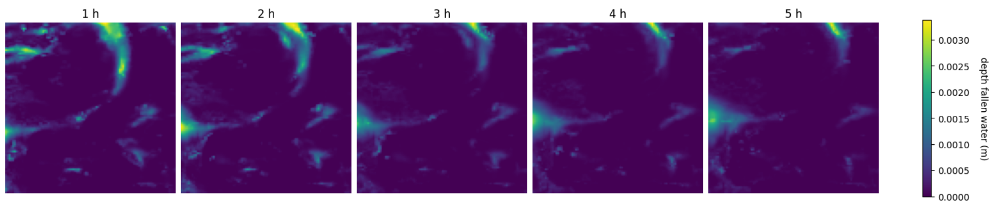


Figure 4.1: Example of precipitation data from the ERA5 dataset.

To prepare the dataset for the experiments, normalization has been performed by scaling the values of both the training and testing sets. This scaling was achieved by dividing all data points by the highest value observed in the training set. Subsequently, the dataset has been divided into two distinct subsets: a training set encompassing the years 2016 to 2020 and a testing set for the year 2021.

It's worth noting that precipitation, the target parameter, tends to exhibit sparsity, often being absent in the analyzed region. This characteristic results in a dataset containing a substantial amount of non-informative data, which can introduce a bias toward predicting zero values [99]. To address this issue, an additional parameter has been introduced in the data generator. This parameter allows to filter and return only sequences where a certain percentage of rain is present in the pixels, effectively simulating the conditions outlined in the **EU-50** and **EU-20** datasets as specified in [98]. In these datasets, images contain at least 50% and 20% of rain in the pixels, respectively. This filtering process involves computing the number of non-zero pixels within a larger region measuring 105x173, after which the image is cropped to its final dimension of 96x96.

4.2 Additional features

Precipitation nowcasting often require a holistic consideration of various meteorological variables that extend beyond the mere presence or absence of rain. Elements such as temperature, atmospheric pressure, humidity, wind speed, and wind direction play pivotal roles in shaping precipitation patterns.

The interactions among these variables and other meteorological factors can be intricate, contributing to the overall complexity of atmospheric dynamics. Furthermore, the model's awareness of the underlying physical structure is essential. Incorporating elements like time embeddings, land/sea masks, and elevation information can enhance the model's capacity to make well-informed predictions.

An ablation study (described in 4.4 and shown in Table 4.2) was conducted, incorporating several supplementary meteorological features sourced from the ERA5 dataset. These additional features, shown in Figure 4.2 and described in Table 4.1, encompassed **wind speed**, derived from both northerly and easterly wind components, the **land-sea mask**, a **geopotential map**, and a **sinusoidal time embedding**. Before training, all these features underwent normalization, scaling them to a range between 0 and 1.

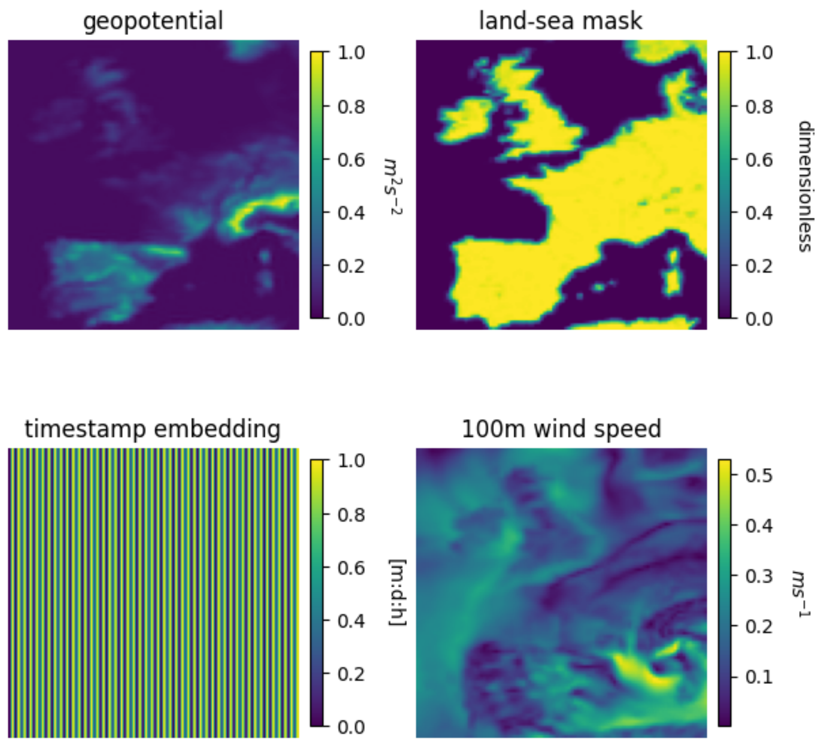


Figure 4.2: Visual example of the additional features.

Name	**Units**	**Description**
100m wind speed	ms^{-1}	Wind speed of air at a height of 100 meters above the surface of the Earth, given easterly and northerly components u and v the speed is obtained by $\sqrt{(u^2 + v^2)}$.
Timestamp	[m,d,h]	Timestamp including month, day, and hour of the start of the given sequence, tile encoded into a 96x96 array.
Land-sea mask	dimensionless	Proportion of land, as opposed to ocean or inland waters in a grid box.
Geopotential	$m^2 s^{-2}$	Gravitational potential energy of a unit mass, at a particular location at the surface of the Earth, relative to mean sea level.

Table 4.1: Additional features units and details.

4.3 Training and Evaluation

The diffusion model was trained with a batch size of 2 throughout 40 epochs. For optimization, the **AdamW** algorithm was employed, utilizing a learning rate of 1e-04 and a weight decay of 1e-05. Interestingly, it was observed increasing the batch size had a detrimental effect on the training process. Furthermore, a fine-tuning phase was conducted, encompassing 10 epochs with a reduced learning rate of 1e-05 and a weight decay of 1e-06, resulting in modest improvements in overall results. To facilitate training, a generator was utilized to produce random batches of sequences covering the training years from 2016 to 2020. As with the reference model, sequences composed of more than 50% non-rain values were excluded from the training process. Following standard diffusion model practices, **Mean Absolute Error (MAE)** has been used as a loss function and applied to the noise difference. Training experiments based on image loss, rather than noise, yielded inferior results.

The evaluation of the diffusion model was carried out using data from the test year of 2021, maintaining a fixed number of 15 diffusion steps and assessing performance using the **Mean Squared Error (MSE)** metric, defined as in Equation 4.1. In this equation, n represents the total number of samples, y_i denotes the ground truth value, and $\hat{y}_i$ signifies the predicted value. It's important to note that all metrics were computed on denormalized data.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.1}$$

Generative Ensemble Diffusion (GED) features a **post-processing U-Net** that shares the spatial dimensions of the denoising U-net, excluding the embedded variance. This network takes fifteen distinct generative outputs from the diffusion model as input, each containing predictions for the subsequent three hours. Subsequently, the U-net produces an output comprising three images, each predicting the rainfall for one of the upcoming three hours.

The training of this model follows a process similar to that used for the diffusion model. It utilizes **AdamW** as the optimizer with a learning rate set at 1e-4 and a weight decay of 1e-5. The loss function employed is the **Mean Squared Error (MSE)**, calculated as the discrepancy between the predicted images and their corresponding ground truth. During training, data is dynamically generated by the diffusion model using random sequences from the training years. Evaluation, on the other hand, is carried out using sequences from the test year of 2021.

4.4 Results

In this section, we outline the experimental setup and describe the experiments conducted. All experiments were conducted using models implemented in the TensorFlow/Keras framework. The training dataset comprised precipitation data and additional features for the designated region, spanning from 2016 to 2021. Conversely, the test set exclusively utilized data collected in 2021. To compute the results, we conducted a comprehensive analysis of all sequences within the specified year for both the EU-20 and EU-50 datasets.

All models simultaneously generate three different predictions. While it's possible that using a distinct model instance for each prediction could potentially yield a slight improvement in overall performance, this would come at the cost of increased training and inference times. This consideration is particularly significant for the practical application of precipitation nowcasting, where efficiency in real-time predictions is crucial.

The primary objective is to minimize the **Mean Squared Error (MSE)** for the initial three hours of the prediction model. However, determining the optimal input data and model configuration remains an ongoing debate. To tackle this challenge, a series of initial tests have been conducted, aiming at identifying the most advantageous set of input features. Table 4.2 provides a comparison of scores obtained using different sets of additional features with the Single Diffusion model.

Single diffusion with different inputs on EU-50			
Inputs	**MSE 1h**	**MSE 2h**	**MSE 3h**
8 rain	2.62e-04	4.60e-04	6.21e-04
8 rain + lsm + geopot	2.60e-04	4.61e-04	6.23e-04
8 rain + lsm + geopot + time	2.60e-04	4.56e-04	6.16e-04
8 rain + lsm + geopot + time + 2 wind speed	<u>2.59e-04</u>	<u>4.51e-04</u>	<u>6.03e-04</u>

Table 4.2: Results comparison on the EU-50 Dataset using different sets of additional features with the Single Diffusion model. All sets include 8 frames representing total precipitation (*rain*). *lsm* and *geopot* stand for land-sea mask and geopotential map, respectively. *time* represents the timestamp embedding.

Regarding model comparison, Table 4.3 and 4.4 offer a comprehensive analysis comparing several models. These include the standard **Core U-Net model** [8], **BroadU-Net** [100], **WF-UNet** (which incorporates additional features as proposed by [98]), the proposed diffusion model with a singular generative output (**Single Diffusion**), and two distinct implementations of the **Generative Ensemble Diffusion (GED)**. The GED models generate a final prediction by integrating 15 different generations of the three predicted frames. In the first implementation, the prediction is calculated by averaging all the values (**mean**), while the second version employs a **post-processing U-Net** for this task (**post-process**). The primary metric utilized for these results is **Mean Squared Error (MSE)**, supplemented with additional metrics such as **Accuracy**, **Precision**, and **Recall**.

The results suggest that while a single diffusion prediction is outperformed by the U-Net models, both implementations of GED significantly surpass the performance of any U-Net-based approach. Notably, the GED version with post-processing demonstrates the most superior overall performance.

MSE values and additional metrics for EU-20 dataset

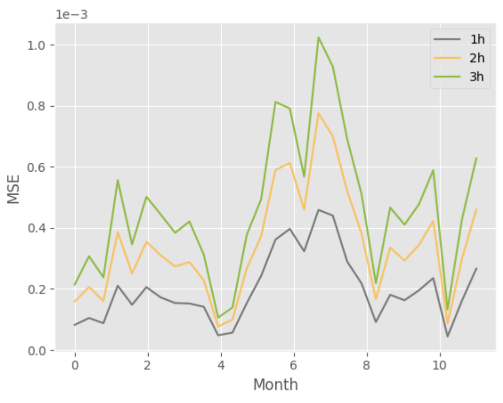
Model	MSE	Accuracy	Precision	Recall
		1 hour ahead		
Core U-net	2.97e-04	0.863	0.698	0.837
Broad U-net	3.05e-04	0.861	0.706	0.803
WF-UNet	2.67e-04	0.933	0.790	0.847
Single Diffusion	2.86e-04	0.911	0.754	0.888
GED (mean)	2.25e-04	0.930	0.786	0.901
GED (postprocess)	2.03e-04	0.923	0.798	0.909
		2 hour ahead		
Core U-net	5.02e-04	0.813	0.609	0.796
Broad U-net	5.05e-04	0.819	0.638	0.712
WF-UNet	4.87e-04	0.895	0.664	0.807
Single Diffusion	4.69e-04	0.886	0.705	0.831
GED (mean)	3.93e-04	0.900	0.731	0.848
GED (postprocess)	3.53e-04	0.898	0.742	0.849
		3 hour ahead		
Core U-net	6.71e-04	0.800	0.612	0.657
Broad U-net	6.55e-04	0.806	0.637	0.609
WF-UNet	6.34e-04	0.877	0.626	0.736
Single Diffusion	6.10e-04	0.853	0.638	0.758
GED (mean)	5.20e-04	0.880	0.689	0.801
GED (postprocess)	4.70e-04	0.891	0.701	0.796

Table 4.3: Results comparison on the EU-20 Dataset.

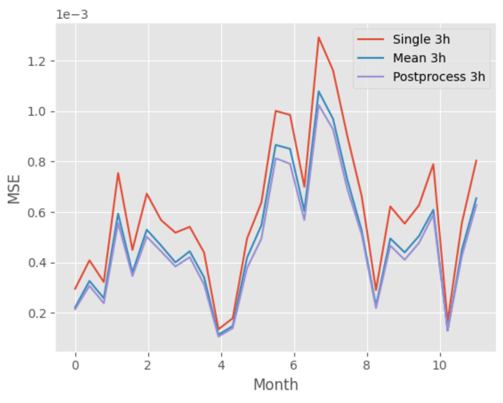
MSE values and additional metrics for EU-50 dataset

Model	MSE	Accuracy	Precision	Recall
1 hour ahead				
Core U-net	3.18e-04	0.862	0.698	0.833
Broad U-net	3.24e-04	0.860	0.705	0.795
WF-UNet	2.50e-04	0.921	0.803	0.849
Single Diffusion	2.59e-04	0.915	0.767	0.882
GED (mean)	2.02e-04	0.924	0.782	0.885
GED (postprocess)	1.99e-04	0.913	0.803	0.907
2 hour ahead				
Core U-net	5.02e-04	0.813	0.609	0.796
Broad U-net	5.05e-04	0.819	0.638	0.712
WF-UNet	4.62e-04	0.877	0.684	0.813
Single Diffusion	4.51e-04	0.875	0.699	0.844
GED (mean)	3.59e-04	0.882	0.711	0.862
GED (postprocess)	3.40e-04	0.878	0.724	0.860
3 hour ahead				
Core U-net	6.71e-04	0.800	0.612	0.657
Broad U-net	6.55e-04	0.806	0.637	0.609
WF-UNet	6.31e-04	0.855	0.647	0.743
Single Diffusion	6.03e-04	0.848	0.672	0.801
GED (mean)	4.92e-04	0.856	0.701	0.828
GED (postprocess)	4.65e-04	0.861	0.706	0.821

Table 4.4: Results comparison on the EU-50 Dataset.



(a)



(b)

Figure 4.3: Single Diffusion results for the year 2021 on EU50, depicting significant score variations depending on the month of the year. Subfigure (a) illustrates the month-wise dissimilarity in scores for each of the three predicted hours. In subfigure (b), we observe that the dissimilarity remains consistent across predictions computed with Single Diffusion, GED (mean), and GED (post-process).

The diffusion model's performance on the 2021 test set exhibits significant dissimilarities across the year, as vividly depicted in Figure 4.3a for all three forecasted timeframes. This dissimilarity persists regardless of whether the prediction is generated using Single Diffusion, GED (mean), or GED (post-process), as demonstrated in Figure 4.3b. Such consistency may be attributed to the inherent complexities of precipitation forecasting, particularly during specific seasonal periods.

Precipitation patterns are notably susceptible to pronounced seasonal fluctuations [104]. For instance, the transition between seasons can trigger abrupt atmospheric changes, introducing challenges in accurately predicting precipitation types and quantities [105]. Convective precipitation, prevalent during warm and humid conditions in the warmer months, is closely associated with rapidly evolving weather systems like thunderstorms. Even in traditional operational meteorology [106, 107], accurately forecasting these systems remains a formidable challenge due to their rapid manifestation and susceptibility to a multitude of intricate and dynamic atmospheric processes.

This has prompted the research community to propose dedicated solutions for this specific challenge in precipitation forecasting[108, 109], particularly in regions prone to such phenomena[110, 111, 112, 113].

Analyzing the obtained results, it becomes evident that this specific challenge significantly impacts the proposed model, particularly during the period from June to September, as indicated by the notable peak in MSE scores between the fifth and eighth months. This timeframe corresponds to the summer months when weather phenomena like convective precipitation and the transit of weather fronts, including cold and warm fronts, are more prevalent.

Meanwhile, transitional seasons, such as spring and fall, witness distinct air masses interacting within these fronts, leading to rapid changes in precipitation distribution. This intricate interplay among air masses introduces further complexity, rendering the precise timing and location of precipitation a formidable task. However, the inclusion of wind speed as an additional feature in the model appears promising, as suggested by the improved performance during the third (April) to fifth (June) months.

Conclusions

In this study, we undertook the challenging task of precipitation nowcasting using diffusion models. Our experimental endeavors, focused on established benchmarks detailed in existing literature, revealed that the proposed Generative Ensemble Diffusion (GED) approach outperformed competing U-Net-based models in terms of overall performance.

The primary goal was to test the hypothesis that a diffusion model could successfully represent the complex and chaotic nature of weather patterns by modeling their probability distribution. This research has been conducted using a subset of the ERA5 dataset, which contains hourly data from a Western European region. The chosen training data covered the years 2016 to 2020, while the testing was performed on data from the year 2021. Furthermore, pre-processing techniques have been adopted, consistent with those used in existing literature, particularly referencing the work of [98].

Experimental results show that including supplementary meteorological features leads to improved prediction quality. Specifically, integrating wind speed, the land-sea mask, timestamp information, and geopotential data contributed significantly to enhancing the overall accuracy of the predictions.

In the initial stages, single generative predictions from the diffusion model exhibited lower performance compared to U-net-based predictions with similar settings. However, a significant breakthrough occurred when conducting multiple generative predictions in parallel. By amalgamating these diverse outcomes through a post-processing step, a substantial improvement has been achieved in prediction quality, outperforming well-established U-net models.

The probabilistic nature of the model, coupled with its ensemble forecasting approach, renders it particularly well-suited for predicting rare events. These events, while occurring with low probability, can have a substantial impact on both the population and the economy, as highlighted in previous research [114].

In summary, this research contributes to the ongoing development of precipitation nowcasting techniques and presents a promising avenue for harnessing diffusion models to gain deeper insights into weather patterns. The integration of GED with post-processing showcases the potential to improve precipitation nowcasting, thereby offering valuable insights for a range of applications and decision-making processes.

One noteworthy aspect of this research is that it was conducted with limited computing resources. The majority of computations were carried out on a single workstation, which was equipped with a Quadro RTX A4000 GPU featuring 16GB of VRAM and 32GB of RAM. Despite these resource constraints, this work produced valuable insights and promising results.

Nevertheless, this research is part of an ongoing collaboration with the *High-Performance Computing Department of Cineca*. As a next step, the model will be evaluated using the state-of-the-art Leonardo system, which offers significantly greater computational power. Specifically, diffusion models will be tested on more intricate weather benchmarks (such as WeatherBench, described in subsection 1.2.3). These benchmarks encompass predictions at varying spatial and temporal scales, with a particular focus on medium and long-term temporal ranges.

Furthermore, this methodology will be applied to the downscaling of meteorological indicators within the framework of the European Cordis Project *Optimal High-Resolution Earth System Models for Exploring Future Climate Changes*. This ambitious initiative aligns with a commitment to advancing the field of climate modeling and enhancing our understanding of future climate changes at a high resolution.

Appendix A

Code fragments

This appendix provides the salient parts of the code used for the experiments. The whole code is archived in the following GitHub repository. The ERA-5 dataset can be openly accessed at the Copernicus website.

In support of the training process, a generator has been implemented to produce randomized batches of sequences spanning the training years (Listing A.1). Additionally, a modified version of this generator that sequentially samples batches has also been implemented. This adjustment has been made to address the challenge posed by evaluating the model on a single year, which inherently contains a significant amount of variability, as illustrated in Figure 4.3. Notice how the additional information is stacked on the fourth axis together with the total precipitation previous frames, as shown in Figure 3.2, in the following order: the first 2 elements of each sequence contain the geopotential map and the land-sea mask, the third element contains the sinusoidal embedding of the timestamp, the fourth and fifth element contain wind speed information of the previous 2 hours, the remaining 11 elements represent total precipitation frames, with 8 frames representing the previous 8 hours and 3 frames representing the ground truth for the following 3 hours. Furthermore, the generator takes an additional parameter `min_rainfall` used to discard sequences not containing a minimum percentage of rainfall in its pixels, to replicate EU-20 and EU-50 datasets from [98].

```python
class DataGenerator(keras.utils.Sequence):
    def __init__(self, data, batch_size=24, min_rainfall = 0.0, time =None, wind = None) :
        self.data = data
        self.time = time
        self.wind = wind
        self.sequence = 11
        self.batch_size = batch_size
        self.num_samples = data.shape[0]
        self.num_batches = int(np.ceil(self.num_samples / self.batch_size))
        self.min_rainfall = min_rainfall # Percent of minimum rainfall per image

    def __len__(self):
        return self.num_batches

    def __getitem__(self, idx):
        result = np.zeros((self.batch_size,96,96,self.sequence + 5))
        result[:,:,:,:2] = addon
        for i in range(self.batch_size):
            while True:
                random = np.random.randint(0,(self.num_samples-self.sequence))
                items = self.data[random:random+self.sequence]
                items = np.expand_dims(items, axis=-1)
                items = np.swapaxes(items, 0, 3)
                if ((np.sum(items[:,:,:,-3] != 0) / (173*105)) < self.min_rainfall):
                    pass
                else:
                    result[i,:,:,2] = (utils.date_to_sinusoidal_embedding(self.time[random]) + 1) / 2
                    result[i,:,:,3:5] = np.transpose(self.wind[random+6:random+8],(1, 2, 0))
                    result[i,:,:,5:] = items[:,:96,:96,:]
                    break
        return result
```

Listing A.1: DataGenerator class implementation from generators.py file.

Listings A.2 and A.3 show the implementation of the U-Net architecture (3.1). The use of `LayerNormalization` instead of `BatchNormalization` (line 23 of Listing A.2), typical of RNNs and Transformers architectures, proved to slightly improve performance. As described in subsections 3.1.2 and 3.3, only 2D convolutions are employed. For upsampling, bilinear interpolation is employed. Functions `get_network` and `get_post_network` are used to create a new U-Net for the Diffusion Model and post processing, respectively. Parameters such as the number and width of blocks can ben set in the `setup.py` file.

Listings A.4, A.5, A.6, A.7 show the implementation of the Diffusion Model. The function `generate2` in Listing A.5 can be used after training to generate new samples (i.e., predictions) using the trained diffusion model, given an adequate input. The function `training_step` in Listing A.6 specified noise loss for training, i.e., evaluating the MAE value between the predicted noise and the exact noise added to the ground truth during the diffusion step instead of the MAE value between the two images. Image loss is still evaluated as a metric. The implementation also provides a `plotter` utility function (Listing A.7) to plot a specified sequence and the relative generated output, as well as MSE values, to keep track of model performance during training, plotting at the end of each epoch.

Training is specified in the `training.py` file. Two keras callback functions have been specified: the `saver` callback function is used to store the weights of the model every specified number of epochs, while a `ReduceLROnPlateau` keras callback function is employed to half the learning rate each time the image loss did not improve for two consecutive epochs.

Evaluation is performed using the `evaluation.py` file. Two experiment functions have been specified, evaluating MSE values on single and ensemble diffusion respectively. Listing A.8 shows the implementation of the former. Other metrics can be evaluated using the `compute_metrics` or `metrics_aggregator` functions found in the `utils.py` file.

```python
def sinusoidal_embedding(x):
    embedding_min_frequency = 1.0
    frequencies = tf.exp(
        tf.linspace(
            tf.math.log(embedding_min_frequency),
            tf.math.log(embedding_max_frequency),
            embedding_dims // 2,
        )
    )
    angular_speeds = 2.0 * math.pi * frequencies
    embeddings = tf.concat(
        [tf.sin(angular_speeds * x), tf.cos(angular_speeds * x)], axis=3
    )
    return embeddings

def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = layers.Conv2D(width, kernel_size=1)(x)
        x = layers.LayerNormalization(axis=-1,center=True, scale=True)(x)
        x = layers.Conv2D(
            width, kernel_size=3, padding="same", activation=keras.
    activations.swish
        )(x)
        x = layers.Conv2D(width, kernel_size=3, padding="same")(x)
        x = layers.Add()([x, residual])
        return x
    return apply

def DownBlock(width, block_depth):
    def apply(x):
        x, skips = x
        for _ in range(block_depth):
            x = ResidualBlock(width)(x)
            skips.append(x)
        x = layers.AveragePooling2D(pool_size=2)(x)
        return x
    return apply

def UpBlock(width, block_depth):
    def apply(x):
        x, skips = x
        x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)
        for _ in range(block_depth):
            x = layers.Concatenate()([x, skips.pop()])
            x = ResidualBlock(width)(x)
        return x

    return apply
```

Listing A.2: U-Net blocks implementation from models.py file.

```python
def get_network(image_size, input_frames, output_frames, widths, block_depth):
    noisy_images = keras.Input(shape=(image_size, image_size, input_frames+output_frames))
    noise_variances = keras.Input(shape=(1, 1, 1))

    e = layers.Lambda(sinusoidal_embedding)(noise_variances)
    e = layers.UpSampling2D(size=image_size, interpolation="nearest")(e)

    x = layers.Conv2D(widths[0], kernel_size=1)(noisy_images)
    x = layers.Concatenate()([x, e])

    skips = []
    for width in widths[:-1]:
        x = DownBlock(width, block_depth)([x, skips])

    for _ in range(block_depth):
        x = ResidualBlock(widths[-1])(x)

    for width in reversed(widths[:-1]):
        x = UpBlock(width, block_depth)([x, skips])

    x = layers.Conv2D(output_frames, kernel_size=1, kernel_initializer="zeros")(x)

    return keras.Model([noisy_images, noise_variances], x, name="residual_unet")

def get_post_network(image_size, input_frames, output_frames, widths, block_depth):
    noisy_images = keras.Input(shape=(image_size, image_size, input_frames))

    x = layers.Conv2D(widths[0], kernel_size=1)(noisy_images)

    skips = []
    for width in widths[:-1]:
        x = DownBlock(width, block_depth)([x, skips])

    for _ in range(block_depth):
        x = ResidualBlock(widths[-1])(x)

    for width in reversed(widths[:-1]):
        x = UpBlock(width, block_depth)([x, skips])

    x = layers.Conv2D(output_frames, kernel_size=1, kernel_initializer="zeros")(x)

    return keras.Model([noisy_images], x, name="residual_unet")
```

Listing A.3: U-Net implementation from models.py file.

```python
class DiffusionModel(keras.Model):
    def __init__(self, image_size, input_frames, output_frames, widths,
    block_depth):
        super().__init__()
        self.normalizer = layers.Normalization()
        self.network = get_network(image_size, input_frames, output_frames,
    widths, block_depth)
        self.ema_network = keras.models.clone_model(self.network)
        self.input_frames = input_frames
        self.output_frames = output_frames

    def compile(self, **kwargs):
        super().compile(**kwargs)
        self.noise_loss_tracker = keras.metrics.Mean(name="n_loss")
        self.image_loss_tracker = keras.metrics.Mean(name="i_loss")

    @property
    def metrics(self):
        return [self.image_loss_tracker, self.noise_loss_tracker]

    def denormalize(self, images):
        # convert the pixel values back to 0-1 range
        images = self.normalizer.mean + images * self.normalizer.variance
    **0.5
        return tf.clip_by_value(images, 0.0, 1.0)

    def diffusion_schedule(self, diffusion_times):
        # diffusion times -> angles
        start_angle = tf.acos(max_signal_rate)
        end_angle = tf.acos(min_signal_rate)
        diffusion_angles = start_angle + diffusion_times * (end_angle -
    start_angle)
        # angles -> signal and noise rates
        signal_rates = tf.cos(diffusion_angles)
        noise_rates = tf.sin(diffusion_angles)
        # note that their squared sum is always: sin^2(x) + cos^2(x) = 1
        return noise_rates, signal_rates

    def denoise(self, noisy_images, noise_rates, signal_rates, training):
        # the exponential moving average weights are used at evaluation
        if training:
            network = self.network
        else:
            network = self.ema_network
        # predict noise component and calculate the image component using it
        pred_noises = network([noisy_images, noise_rates**2], training=
    training)
        pred_images = (noisy_images[:,:,:,-self.output_frames:] -
    noise_rates * pred_noises) / signal_rates
        return pred_noises, pred_images
```

Listing A.4: DiffusionModel class implementation from models.py file (1): initialization, compilation, metrics, denormalization, diffusion schedule and denoising functions.

```python
    def reverse_diffusion(self, initial_noise, diffusion_steps):
        # reverse diffusion = sampling
        num_images = initial_noise.shape[0]
        step_size = 1.0 / diffusion_steps
        past = initial_noise[:,:,:,:-self.output_frames]
        #future = initial_noise[-1]
        # important line:
        # at the first sampling step, the "noisy image" is pure noise
        # but its signal rate is assumed to be nonzero (min_signal_rate)
        next_noisy_images = initial_noise
        for step in range(diffusion_steps):
            noisy_images = next_noisy_images
            # separate the current noisy image to its components
            diffusion_times = tf.ones((num_images, 1, 1, 1)) - step * step_size
            noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=False
            )
            # network used in eval mode
            # remix the predicted components using the next signal and noise rates
            next_diffusion_times = diffusion_times - step_size
            next_noise_rates, next_signal_rates = self.diffusion_schedule(
                next_diffusion_times
            )
            next_noisy_frames = (
                next_signal_rates * pred_images + next_noise_rates * pred_noises
            )
            #concatenate predicted single frame with past known frames
            next_noisy_images = tf.concat([past, next_noisy_frames], axis = -1)
        return pred_images

    def generate2(self, images, diffusion_steps):
        # noise -> images -> denormalized images
        initial_noise = tf.random.normal(shape=(images.shape[0], images.shape[1], images.shape[2], self.output_frames))
        images[:,:,:,-self.output_frames:] = initial_noise
        generated_images = self.reverse_diffusion(images, diffusion_steps)
        generated_images = self.denormalize(tf.concat([images[:,:,:,:-self.output_frames],generated_images],axis=-1))
        return generated_images
```

Listing A.5: DiffusionModel class implementation from models.py file (2): reverse diffusion and generate2 functions.

```python
    def train_step(self, images):
        # normalize images to have standard deviation of 1, like the noises
        #normalize only real images
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, image_size, image_size, self.output_frames))
        # sample uniform random diffusion times
        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        # mix the images with noises accordingly
        target = images[:,:,:,-self.output_frames:]
        noisy_images = signal_rates * target + noise_rates * noises
        #concat the images with added noises with the originals
        noise_two =  tf.concat([images[:,:,:,:-self.output_frames], noisy_images],axis=-1)
        #print(noise_two.shape)
        with tf.GradientTape() as tape:
            # train the network to separate noisy images to their components
            pred_noises, pred_images = self.denoise(
                noise_two, noise_rates, signal_rates, training=True
            )
            noise_loss = self.loss(noises, pred_noises)  # used for training
            image_loss = self.loss(target, pred_images)  # only used as metric
        # Training on noise_loss (default)
        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
        self.noise_loss_tracker.update_state(noise_loss)
        self.image_loss_tracker.update_state(image_loss)
        # track the exponential moving averages of weights
        for weight, ema_weight in zip(self.network.weights, self.ema_network.weights):
            ema_weight.assign(ema * ema_weight + (1 - ema) * weight)
        return {m.name: m.result() for m in self.metrics}
```

Listing A.6: DiffusionModel class implementation from models.py file (3): train step function.

```
    def test_step(self, images):
        # normalize images to have standard deviation of 1, like the noises
        images = self.normalizer(images, training=False)
        noises = tf.random.normal(shape=(batch_size, image_size, image_size,
     self.output_frames))
        # sample uniform random diffusion times
        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        # mix the images with noises accordingly
        target = images[:,:,:,-self.output_frames:]
        noisy_images = signal_rates * target + noise_rates * noises
        noise_two =  tf.concat([images[:,:,:,:-self.output_frames],
    noisy_images],axis=-1)
        # use the network to separate noisy images to their components
        pred_noises, pred_images = self.denoise(
            noise_two, noise_rates, signal_rates, training=False
        )
        noise_loss = self.loss(noises, pred_noises)
        image_loss = self.loss(target, pred_images)
        self.image_loss_tracker.update_state(image_loss)
        self.noise_loss_tracker.update_state(noise_loss)
        return {m.name: m.result() for m in self.metrics}

    def plotter(self,epoch, logs):
        sample = test_generator50.__getitem__(1)
        hist = np.copy(sample)
        sample = model.normalizer(sample)
        tmp = model.generate2(np.copy(sample),15)
        hist = hist * maxRtesr
        tmp = tmp * maxRtesr
        mse = np.mean(np.sum((hist[:,:,:,:]-tmp[:,:,:,:])**2,axis=(1,2)),
    axis=0)
        mse = np.round(mse, 8)
        print("\n mse values :")
        print("\n" + str(mse) + "\n")
        plt.figure(figsize=(6,6))
        for i in range(tmp.shape[-1]):
                plt.subplot(1, tmp.shape[-1], i + 1)
                plt.imshow(tmp[0,:,:,i])
                plt.axis("off")
        plt.tight_layout()
        plt.show()
        plt.close()
```

Listing A.7: DiffusionModel class implementation from models.py file (4): test step and plotter functions.

```python
def experiment(generator = test_generator50 , n_iter=29):
    history = np.zeros((n_iter ,3))
    raw_data = np.zeros((n_iter ,batch_size ,96 ,96 ,6))

    #define final mse array
    mses = np.zeros((3))
    for i in range(n_iter):

        #select a random batch in the test set
        sample = generator.__getitem__(i)
        # save a copy as ground truth
        hist = np.copy(sample)
        #normalize sample before generation
        sample = model.normalizer(sample)
        # compute generation with 15 diffusion steps
        tmp = model.generate2(np.copy(sample) ,15)

        # Denormalize prediction and g.t.
        hist = hist * maxRtrain
        tmp = tmp * maxRtrain

        raw_data[i,:,:,:,:3] = hist[:,:,:,-3:]
        raw_data[i,:,:,:,3:] = tmp[:,:,:,-3:]

        # compute the metric, sum on last two axis, mean on first.
        mse = np.mean(np.sum((hist[:,:,:,-3:]-tmp[:,:,:,-3:])**2,axis=(1,2))
    ,axis=0)
        history[i] = mse
        print(mse)

        # add 3 relevant meteric values to array
        mses += mse[-3:]
    # return average of all mses
    return mses / n_iter, history, raw_data
```

Listing A.8: Evaluation function for Single Diffusion generation.

Appendix B

Ablation study

This appendix provides an ablation study performed during the development of the model to find the best configuration. They can be useful as basic guideline for future experiments and model improvements. Experiments include removal or addition of additional information, with *ADDONS* referring to geopotential map and land-sea mask, change in the U-Net architecture (e.g., removing the final layer), or changing the number of input frames used for conditioning. All diffusion model experiments have been evaluated on single diffusion, except when specified otherwise. *eval* specified the portion of test set used for evaluation. All scores refer to MSE values.

Table B.1: Results on EU-20 for various configurations.

Model	Score 1h	Score 2h	Score 3h	Details
WF-Unet	2.67e-04	4.87e-04	6.34e-04	Trained by authors on EU20.
Diffusion model, input: 12 rain, output frames: 3, batch: 2, eval: 100%	3.59e-04	5.98e-04	7.77e-04	14 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.

Continued on next page

Table B.1 – continued from previous page

Model	Score 1h	Score 2h	Score 3h	Details
Diffusion model, input: 12 rain, output frames: 3, batch 2, ENSEMBLE: 15, eval 12.5%	2.56e-04	4.17e-04	5.44e-04	14 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 12 rain + ADDONS, output frames: 3, batch: 32, eval: 100%	3.23e-04	5.33e-04	6.84e-04	34 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 12 rain + ADDONS, output frames: 3, batch: 32, ENSEMBLE: 15, eval: 100%	2.59e-04	4.56e-04	5.86e-04	50 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 12 rain + ADDONS, output frames: 3, batch: 4, ENSEMBLE: 15, eval: 100%	2.28e-04	4.31e-04	5.71e-04	22 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 12 rain + ADDONS + 4 WIND, output frames: 3, batch: 4, eval: 100%	3.02e-04	5.12e-04	6.80e-04	34(16) epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.

Continued on next page

Table B.1 – continued from previous page

Model	Score 1h	Score 2h	Score 3h	Details
Diffusion model, input: 8 rain + ADDONS + 4 WIND, output frames: 3, batch: 4, eval: 100%	2.94e-04	5.01e-04	6.56e-04	34(16) epochs, lr: $1e^{-4}$ + $1e^{-5}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 8 rain + ADDONS + 4 WIND, output frames: 3, batch: 4, ENSEMBLE: 15, eval: 100%	2.28e-04	4.28e-04	5.80e-04	34(16) epochs, lr: $1e^{-4}$ + $1e^{-5}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 8 rain + ADDONS + 4 WIND, output frames: 3, batch: 4, UNET ENSEMBLE: 10, eval: 100%	2.51e-04	3.99e-04	5.17e-04	34(16) epochs, lr: $1e^{-4}$ + $1e^{-5}$. 5 epochs Unet 100/16. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 4 rain, output frames: 4, batch: 2, eval: 100%	3.40e-04	5.67e-04	7.24e-04	24 epochs, (12 lr: $1e^{-4}$ + 12 lr: $5e^{-5}$). AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.

Continued on next page

Table B.1 – continued from previous page

Model	Score 1h	Score 2h	Score 3h	Details
Diffusion model, input: 8 rain, output frames: 3, batch: 2, eval: 100%	3.04e-04	5.14e-04	6.74e-04	24 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 12 rain, output frames: 3, batch: 2, eval: 100%	3.28e-04	5.33e-04	6.88e-04	24 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 4 rain + 4 PRESSURE, output frames: 3, batch: 2, eval: 100%	3.10e-04	5.19e-04	6.71e-04	16 epochs, lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.
Diffusion model, input: 4 rain + 4 PRESSURE, output frames: 3, batch: 2, eval: 100%	3.32e-04	5.90e-04	7.26e-04	24 epochs (possible overfit!), lr: $1e^{-4}$. AdamW (no addons) with weight decay $0.1 \cdot lr$. MAE as loss. [64, 128, 256, 384] widths and 2 block depth.

Bibliography

[1] Andrea Asperti, Fabio Merizzi, Alberto Paparella, Giorgio Pedrazzi, Matteo Angelinelli, and Stefano Colamonaco. Precipitation nowcasting with generative diffusion models, 2023.

[2] Richard Swinbank, Petra Friederichs, and Sabrina Wahl. *Forecasting high-impact weather using ensemble prediction systems*, page 95–112. Special Publications of the International Union of Geodesy and Geophysics. Cambridge University Press, 2016.

[3] J.R. Holton and G.J. Hakim. *An Introduction to Dynamic Meteorology*. International Geophysics. Elsevier Science, 2013.

[4] Roberto Buizza. Introduction to the special issue on "25 years of ensemble forecasting". *Quarterly Journal of the Royal Meteorological Society*, 145(S1):1–11, 2019.

[5] Javier García-Pereda, José Miguel Fernández-Serdán, Óscar Alonso, Adrián Sanz, Rocío Guerra, Cristina Ariza, Inés Santos, and Laura Fernández. Nwcsaf high resolution winds (nwc/geo-hrw) stand-alone software for calculation of atmospheric motion vectors and trajectories. *Remote Sensing*, 11(17), 2019.

[6] Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, et al. Skilful precipitation nowcast-

ing using deep generative models of radar. *Nature*, 597(7878):672–677, 2021.

[7] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[10] Jussi Leinonen, Ulrich Hamann, Daniele Nerini, Urs Germann, and Gabriele Franch. Latent diffusion models for generative precipitation nowcasting with accurate uncertainty quantification. *CoRR*, abs/2304.12891, 2023.

[11] 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.

[12] Molly E. Brown. *Famine Early Warning Systems and Remote Sensing Data.* 2008.

[13] James O. Pinto, Debbie O'Sullivan, Stewart Taylor, Jack Elston, C. B. Baker, David Hotz, Curtis Marshall, Jamey Jacob, Konrad Barfuss, Bruno Piguet, Greg Roberts, Nadja Omanovic, Martin Fengler, Anders A. Jensen, Matthias Steiner, and Adam L. Houston. The Status

and Future of Small Uncrewed Aircraft Systems (UAS) in Operational Meteorology. *Bulletin of the American Meteorological Society*, 102(11):E2121–E2136, November 2021.

[14] Paul A. Dirmeyer, C. Adam Schlosser, and Kaye L. Brubaker. Precipitation, Recycling, and Land Memory: An Integrated Analysis. *Journal of Hydrometeorology*, 10(1):278, January 2009.

[15] Lewis Fry Richardson and Peter Lynch. *Weather Prediction by Numerical Process*. Cambridge Mathematical Library. Cambridge University Press, 2 edition, 2007.

[16] Julian Hunt. The emergence of numerical weather prediction: Richardson's dream by peter lynch (cup, november 2006), pp. xi + 280, hardback isbn 0521857295. *Quarterly Journal of The Royal Meteorological Society - QUART J ROY METEOROL SOC*, 133:2143–2144, 10 2007.

[17] R.A. Pielke. *Mesoscale Meteorological Modeling*. International Geophysics. Elsevier Science, 2002.

[18] Jordan G. Powers, Joseph B. Klemp, William C. Skamarock, Christopher A. Davis, Jimy Dudhia, David O. Gill, Janice L. Coen, David J. Gochis, Ravan Ahmadov, Steven E. Peckham, Georg A. Grell, John Michalakes, Samuel Trahan, Stanley G. Benjamin, Curtis R. Alexander, Geoffrey J. Dimego, Wei Wang, Craig S. Schwartz, Glen S. Romine, Zhiquan Liu, Chris Snyder, Fei Chen, Michael J. Barlage, Wei Yu, and Michael G. Duda. The weather research and forecasting model: Overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, 98(8):1717 – 1737, 2017.

[19] Liu Liang, Mihail Popescu, Marjorie Skubic, Marilyn Rantz, Tarik Yardibi, and Paul Cuddihy. Automatic fall detection based on doppler radar motion signature. IEEE, 4 2012.

[20] Rodger A. Brown and John M. Lewis. PATH TO NEXRAD: Doppler Radar Development at the National Severe Storms Laboratory. *Bulletin of the American Meteorological Society*, 86(10):1459–1470, October 2005.

[21] Madalina Surcel, Isztar Zawadzki, and M. K. Yau. A study on the scale dependence of the predictability of precipitation patterns. *Journal of the Atmospheric Sciences*, 72(1):216 – 235, 2015.

[22] Shejule Priya Ashok and Sreeja Pekkat. A systematic quantitative review on the performance of some of the recent short-term rainfall forecasting techniques. *Journal of Water and Climate Change*, 13(8):3004–3029, 2022.

[23] Juanzhen Sun, Ming Xue, James W. Wilson, Isztar Zawadzki, Sue P. Ballard, Jeanette Onvlee-Hooimeyer, Paul Joe, Dale M. Barker, Ping-Wah Li, Brian Golding, Mei Xu, and James Pinto. Use of nwp for nowcasting convective precipitation: Recent progress and challenges. *Bulletin of the American Meteorological Society*, 95(3):409 – 426, 2014.

[24] Neill Bowler, Clive Pierce, and Alan Seed. Steps: A probabilistic precipitation forecasting scheme which merges an extrapolation nowcast with downscaled nwp. *Quarterly Journal of the Royal Meteorological Society*, 132:2127 – 2155, 01 2007.

[25] Alan Seed, Clive Pierce, and Katie Norman. Formulation and evaluation of a scale decomposition-based stochastic precipitation nowcast scheme. *Water Resources Research*, 49, 10 2013.

[26] Stéphane Vannitsem, Daniel S Wilks, and Jakob Messner. Statistical postprocessing of ensemble forecasts. 2018.

[27] Glenn W. Brier. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1):1, January 1950.

[28] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[29] Seppo Pulkkinen, Daniele Nerini, Andrés Pérez Hortal, Carlos Velasco-Forero, Alan Seed, Urs Germann, and Loris Foresti. Pysteps: an open-source python library for probabilistic precipitation nowcasting (v1.0). *Geoscientific Model Development*, 12:4185–4219, 10 2019.

[30] Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, Adrian Simmons, Cornel Soci, Saleh Abdalla, Xavier Abellan, Gianpaolo Balsamo, Peter Bechtold, Gionata Biavati, Jean Bidlot, Massimo Bonavita, Giovanna De Chiara, Per Dahlgren, Dick Dee, Michail Diamantakis, Rossana Dragani, Johannes Flemming, Richard Forbes, Manuel Fuentes, Alan Geer, Leo Haimberger, Sean Healy, Robin J. Hogan, Elías Hólm, Marta Janisková, Sarah Keeley, Patrick Laloyaux, Philippe Lopez, Cristina Lupu, Gabor Radnoti, Patricia de Rosnay, Iryna Rozum, Freja Vamborg, Sebastien Villaume, and Jean-Noël Thépaut. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

[31] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1278–1286. JMLR.org, 2014.

[32] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Found. Trends Mach. Learn.*, 12(4):307–392, 2019.

[33] Andrea Asperti, Davide Evangelista, and Elena Loli Piccolomini. A survey on variational autoencoders from a green AI perspective. *SN Comput. Sci.*, 2(4):301, 2021.

[34] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[35] Abdul Jabbar, Xi Li, and Bourahla Omar. A survey on generative adversarial networks: Variants, applications, and training, 2020.

[36] David E Rumelhart, James L McClelland, and CORPORATE PDP Research Group. *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations.* MIT press, 1986.

[37] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[38] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

[39] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[40] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[41] Cheng Tan, Zhangyang Gao, and Stan Z Li. Simvp: Towards simple yet powerful spatiotemporal predictive learning. *arXiv preprint arXiv:2211.12509*, 2022.

[42] Cheng Tan, Zhangyang Gao, Siyuan Li, Yongjie Xu, and Stan Z Li. Temporal attention unit: Towards efficient spatiotemporal predictive learning. *arXiv preprint arXiv:2206.12126*, 2022.

[43] Yuankang Ye, Feng Gao, Wei Cheng, Chang Liu, and Shaoqing Zhang. Msstnet: A multi-scale spatiotemporal prediction neural network for precipitation nowcasting. *Remote Sensing*, 15(1):137, 2022.

[44] Yufan Zhou, Haiwei Dong, and Abdulmotaleb El Saddik. Deep learning in next-frame prediction: A benchmark review. *IEEE Access*, 8:69273–69283, 2020.

[45] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 802–810, 2015.

[46] G. Ayzel, T. Scheffer, and M. Heistermann. Rainnet v1.0: a convolutional neural network for radar-based precipitation nowcasting. *Geoscientific Model Development*, 13(6):2631–2644, 2020.

[47] Gabriele Franch, Daniele Nerini, Marta Pendesini, Luca Coviello, Giuseppe Jurman, and Cesare Furlanello. Precipitation nowcasting with orographic enhanced stacked generalization: Improving deep learning predictions on extreme events. *Atmosphere*, 11(3), 2020.

[48] 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.

[49] Rilwan A. Adewoyin, Peter Dueben, Peter Watson, Yulan He, and Ritabrata Dutta. TRU-NET: a deep learning approach to high resolution prediction of rainfall. *Mach. Learn.*, 110(8):2035–2062, 2021.

[50] Lasse Espeholt, Shreya Agrawal, Casper Sønderby, Manoj Kumar, Jonathan Heek, Carla Bromberg, Cenk Gazen, Rob Carver, Marcin Andrychowicz, Jason Hickey, et al. Deep learning for twelve hour precipitation forecasts. *Nature communications*, 13(1):5145, 2022.

[51] 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. *CoRR*, abs/2211.02556, 2022.

[52] Yusuke Hatanaka, Yannik Glaser, Geoff Galgon, Giuseppe Torri, and Peter Sadowski. Diffusion models for high-resolution solar forecasts. *ArXiv*, abs/2302.00170, 2023.

[53] Achraf Oussidi and Azeddine Elhassouny. Deep generative models: Survey. In *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–8, 2018.

[54] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, 44(2):e202100008, 2021.

[55] Andrea Asperti and Valerio Tonelli. Comparing the latent space of generative models. *Neural Computing & Applications*, To appear, 2022.

[56] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Bolei Zhou, Hendrik Strobelt, and Antonio Torralba. Seeing what a gan cannot generate. pages 4501–4510, 10 2019.

[57] Jussi Leinonen, Daniele Nerini, and Alexis Berne. Stochastic super-resolution for downscaling time-evolving atmospheric fields with a

generative adversarial network. *IEEE Trans. Geosci. Remote. Sens.*, 59(9):7211–7223, 2021.

[58] Ilan Price and Stephan Rasp. Increasing the accuracy and resolution of precipitation forecasts using deep generative models. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, pages 10555–10571, 2022.

[59] Lucy Harris, Andrew T. T. McRae, Matthew Chantry, Peter D. Dueben, and Tim N. Palmer. A generative deep learning approach to stochastic downscaling of precipitation forecasts. *Journal of Advances in Modeling Earth Systems*, 14(10):e2022MS003120, 2022.

[60] Negin Hayatbini, Bailey Kong, Kuo-lin Hsu, Phu Nguyen, Soroosh Sorooshian, Graeme Stephens, Charless Fowlkes, Ramakrishna Nemani, and Sangram Ganguly. Conditional generative adversarial networks (cgans) for near real-time precipitation estimation from multispectral goes-16 satellite imageries—persiann-cgan. *Remote Sensing*, 11(19), 2019.

[61] Cunguang Wang, Guoqiang Tang, and Pierre Gentine. Precipgan: Merging microwave and infrared data for satellite precipitation estimation using generative adversarial network. *Geophysical Research Letters*, 48(5):e2020GL092032, 2021.

[62] S. Scher and S. Peßenteiner. Technical note: Temporal disaggregation of spatial rainfall fields with generative adversarial networks. *Hydrology and Earth System Sciences*, 25(6):3207–3225, 2021.

[63] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[64] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. *arXiv e-prints*, page arXiv:2010.02502, October 2020.

[65] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[66] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *CoRR*, abs/2205.11487, 2022.

[67] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[68] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022.

[69] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 8780–8794, 2021.

[70] Andrea Asperti, Davide Evangelista, Samuele Marro, and Fabio Merizzi. Image embedding for denoising generative models. *Artificial Intelligence Review*, To appear, 2023.

[71] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. 37:2256–2265, 2015.

[72] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[73] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[74] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22563–22575, 2023.

[75] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651, 2017.

[76] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022.

[77] Lucy Harris, Andrew TT McRae, Matthew Chantry, Peter D Dueben, and Tim N Palmer. A generative deep learning approach to stochastic downscaling of precipitation forecasts. *Journal of Advances in Modeling Earth Systems*, 14(10):e2022MS003120, 2022.

[78] Urs Germann, Gianmario Galli, Marco Boscacci, and Martin Bolliger. Radar precipitation measurement in a mountainous region. *Quarterly Journal of the Royal Meteorological Society*, 132(618):1669–1692, 2006.

[79] S. Willemse and M. Furger. From weather observations to atmospheric and climate sciences in switzerland: Celebrating 100 years of the swiss society for meteorology. chapter 9. 2016.

[80] K. Stephan, S. Klink, and C. Schraff. Assimilation of radar-derived rain rates into the convective-scale model cosmo-de at dwd. *Quarterly Journal of the Royal Meteorological Society*, 134(634):1315–1326, 2008.

[81] Javier Gurrola-Ramos, Oscar Dalmau, and Teresa E. Alarcón. A residual dense u-net neural network for image denoising. *IEEE Access*, 9:31742–31754, 2021.

[82] Sehyung Lee, Makiko Negishi, Hidetoshi Urakubo, Haruo Kasai, and Shin Ishii. Mu-net: Multi-scale u-net for two-photon microscopy image denoising and restoration. *Neural Networks*, 125:92–103, 2020.

[83] Mattias P Heinrich, Maik Stille, and Thorsten M Buzug. Residual u-net convolutional neural network architecture for low-dose ct denoising. *Current Directions in Biomedical Engineering*, 4(1):297–300, 2018.

[84] Rina Komatsu and Tad Gonsalves. Comparing u-net based models for denoising color images. *AI*, 1(4):465–486, 2020.

[85] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[87] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[88] Tilmann Gneiting, Adrian E Raftery, Anton H Westveld, and Tom Goldman. Calibrated probabilistic forecasting using ensemble model output statistics and minimum crps estimation. *Monthly Weather Review*, 133(5):1098–1118, 2005.

[89] Alexander Henzi, Johanna F Ziegel, and Tilmann Gneiting. Isotonic distributional regression. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 83(5):963–993, 2021.

[90] Maxime Taillardat, Olivier Mestre, Michaël Zamo, and Philippe Naveau. Calibrated ensemble forecasts using quantile regression forests and ensemble model output statistics. *Monthly Weather Review*, 144(6):2375–2393, 2016.

[91] Jakob W Messner, Georg J Mayr, and Achim Zeileis. Nonhomogeneous boosting for predictor selection in ensemble postprocessing. *Monthly Weather Review*, 145(1):137–147, 2017.

[92] Saleh Ashkboos, Langwen Huang, Nikoli Dryden, Tal Ben-Nun, Peter Dueben, Lukas Gianinazzi, Luca Kummer, and Torsten Hoefler. ENS-10: A dataset for post-processing ensemble weather forecast. *CoRR*, abs/2206.14786, 2022.

[93] Benedikt Schulz and Sebastian Lerch. Machine learning methods for postprocessing ensemble forecasts of wind gusts: A systematic comparison. *Monthly Weather Review*, 150(1):235 – 257, 2022.

[94] Andrea Asperti, Davide Evangelista, and Moreno Marzolla. Dissecting flops along input dimensions for greenai cost estimations. In *7th International Conference on Machine Learning, Optimization & Data Science, Grasmere, Lake District, England – UK, October 5-8 2021.*, Springer International Publishing, pages 86–100, 2022.

[95] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent dif-

fusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10674–10685, 2022.

[96] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive Fourier Neural Operators: Efficient Token Mixers for Transformers. *arXiv e-prints*, page arXiv:2111.13587, November 2021.

[97] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators. *arXiv e-prints*, page arXiv:2202.11214, February 2022.

[98] Christos Kaparakis and Siamak Mehrkanoon. Wf-unet: Weather fusion unet for precipitation nowcasting. *CoRR*, abs/2302.04102, 2023.

[99] Kevin Trebing, Tomasz Stanczyk, and Siamak Mehrkanoon. Smaat-unet: Precipitation nowcasting using a small attention-unet architecture, 2021.

[100] Jesús García Fernández and Siamak Mehrkanoon. Broad-UNet: Multi-scale feature learning for nowcasting tasks. *Neural Networks*, 144:419–427, dec 2021.

[101] Jesús García Fernández, Ismail Alaoui Abdellaoui, and Siamak Mehrkanoon. Deep coastal sea elements forecasting using U-Net based models. *arXiv e-prints*, page arXiv:2011.03303, November 2020.

[102] G. Ayzel, M. Heistermann, A. Sorokin, O. Nikitin, and O. Lukyanova. All convolutional neural networks for radar-based precipitation

nowcasting. *Procedia Computer Science*, 150:186–192, 2019. Proceedings of the 13th International Symposium “Intelligent Systems 2018” (INTELS’18), 22-24 October, 2018, St. Petersburg, Russia.

[103] Carla L. Bromberg, Cenk Gazen, Jason J. Hickey, John Burge, Luke Barrington, and Shreya Agrawal. Machine learning for precipitation nowcasting from radar images. page 4, 2019.

[104] Alexandre Tuel and Olivia Martius. The influence of modes of climate variability on the sub-seasonal temporal clustering of extreme precipitation. *iScience*, 25(3):103855, 2022.

[105] Phong Le, James Randerson, Rebecca Willett, Stephen Wright, Padhraic Smyth, Clement Guilloteau, Antonios Mamalakis, and Efi Foufoula-Georgiou. Climate-driven changes in the predictability of seasonal precipitation. *Nature Communications*, 14, 06 2023.

[106] P.S. Ray. American Meteorological Society, 1986.

[107] David J. Stensrud, Ming Xue, Louis J. Wicker, Kevin E. Kelleher, Michael P. Foster, Joseph T. Schaefer, Russell S. Schneider, Stanley G. Benjamin, Stephen S. Weygandt, John T. Ferree, and Jason P. Tuell. Convective-scale warn-on-forecast system: A vision for 2020. *Bulletin of the American Meteorological Society*, 90(10):1487 – 1500, 2009.

[108] Lei Han, He Liang, Haonan Chen, Wei Zhang, and Yurong Ge. Convective precipitation nowcasting using u-net model. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–8, 2022.

[109] Vlado Spiridonov, Julian Baez, Bosko Telenta, and Boro Jakimovski. Prediction of extreme convective rainfall intensities using a free-running 3-d sub-km-scale cloud model initialized from wrf km-scale nwp forecasts. *Journal of Atmospheric and Solar-Terrestrial Physics*, 209:105401, 2020.

[110] Yanbo Nie, Jianqi Sun, and Jiehua Ma. Seasonal prediction of summer extreme precipitation frequencies over southwest china based on machine learning. *Atmospheric Research*, page 106947, 2023.

[111] Wenguang Wei, Zhongwei Yan, Xuan Tong, Zuoqiang Han, Miaomiao Ma, Shuang Yu, and Jiangjiang Xia. Seasonal prediction of summer extreme precipitation over the yangtze river based on random forest. *Weather and Climate Extremes*, 37:100477, 2022.

[112] André de Sousa Araújo, Adma Raia Silva, and Luis E. Zárate. Extreme precipitation prediction based on neural network model – a case study for southeastern brazil. *Journal of Hydrology*, 606:127454, 2022.

[113] L Bodri and V Čermák. Prediction of extreme precipitation using a neural network: application to summer flood occurrence in moravia. *Advances in Engineering Software*, 31(5):311–321, 2000.

[114] Tim N Palmer. The economic value of ensemble forecasts as a tool for risk assessment: From days to decades. *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography*, 128(581):747–774, 2002.

Ringraziamenti

Alla fine di questo elaborato, mi sembra doveroso dedicare uno spazio per esprimere la mia profonda gratitudine a tutte le persone che sono state al mio fianco durante il mio percorso universitario. Senza il loro sostegno, il mio percorso accademico non sarebbe stato lo stesso.
Innanzitutto, vorrei ringraziare il mio relatore, il Prof. Andrea Asperti, per la sua disponibilità, la sua pazienza e la sua fiducia in me. La sua dedizione alla ricerca è un punto di riferimento. Senza la sua guida esperta e il suo incoraggiamento costante, questa tesi non sarebbe stata possibile.
Desidero anche ringraziare il mio correlatore, il Dott. Fabio Merizzi, per la sua contagiosa positività, per aver condiviso con me la sua esperienza e per le preziose osservazioni che ha fornito durante la stesura di questa tesi.
Un ringraziamento speciale va ai miei genitori e alla mia famiglia per il loro amore, il loro sostegno incondizionato e per aver sempre creduto in me. Grazie di cuore per avermi sostenuto in ogni fase del mio percorso accademico.
Desidero anche ringraziare i miei amici, per essere stati sempre presenti, per le discussioni stimolanti, per le risate condivise e per il supporto morale nei momenti di difficoltà. Non avrei potuto chiedere amici migliori.
Grazie a tutti coloro che hanno contribuito in modo diretto o indiretto a questa tesi. La vostra gentilezza, il vostro supporto e la vostra presenza nella mia vita sono inestimabili.

Alberto