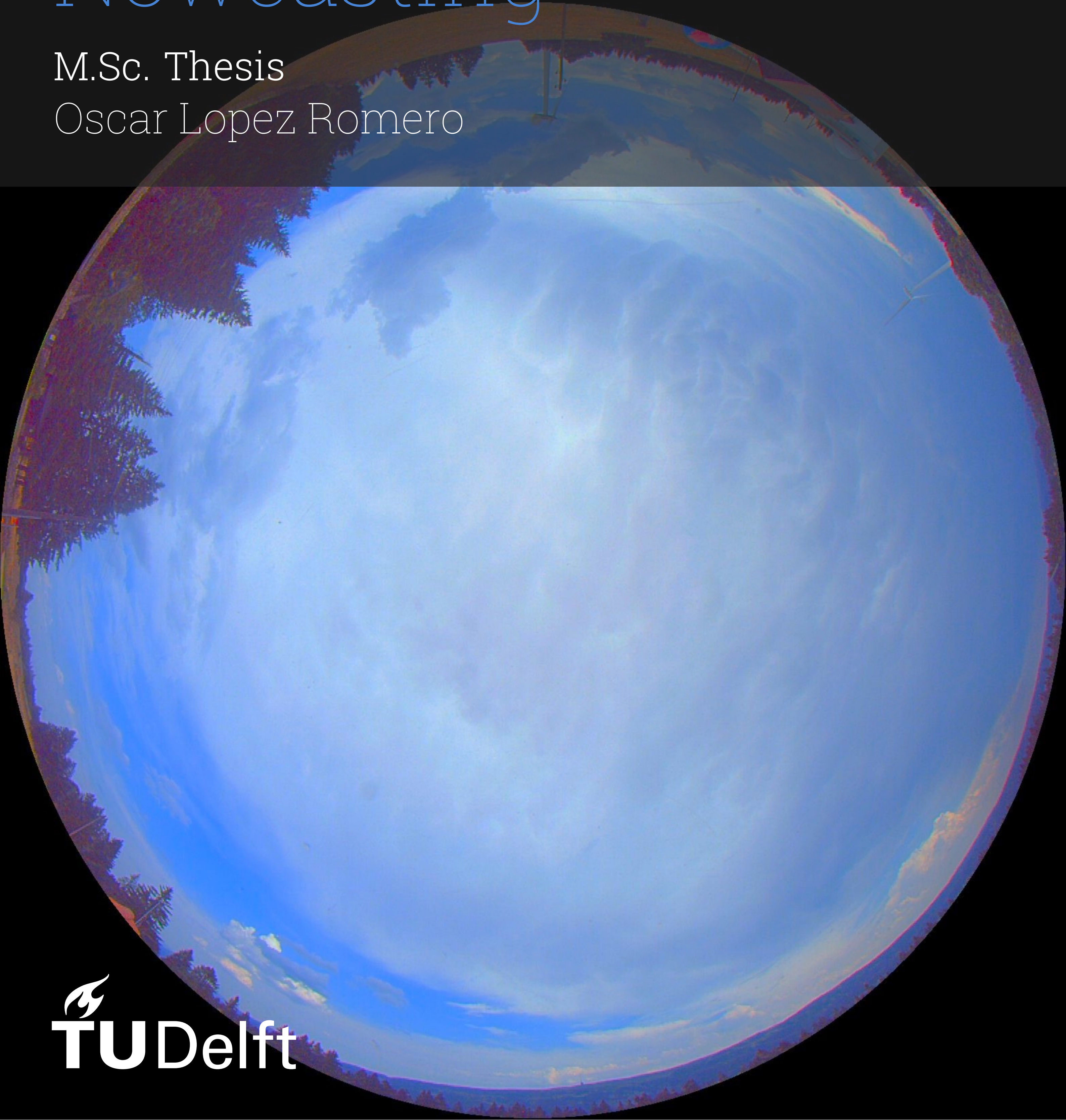
Latent Diffusion for Generative Solar Nowcasting

M.Sc. Thesis

Oscar Lopez Romero



Latent Diffusion for Generative Solar Nowcasting

by

Oscar Lopez Romero
Student Number: 5760313

In partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering

at the Delft University of Technology.
To be defended publicly on the 25th of October 2024

Carried out from February to October 2024 at the Central European Hitachi Energy Research Center
in Baden-Dättwil, Switzerland.

Advisor:	Dr. Jochen L. Cremer
Supervisor:	Dr. Jan Poland

TU Delft

Acknowledgements

I would like to express my gratitude to everyone at the Central European Hitachi Energy Research Center. The vibrant community of interns and scientists has created an incredibly supportive and stimulating environment. The beautiful setting, the collaborative atmosphere, and the access to cutting-edge resources make this an ideal place to do research.
I owe my deepest gratitude to my supervisor, Dr. Jan Poland. Not only for sharing his immense expertise during countless hours of technical advice, but especially for his guidance, positivity and continous encouragement. I am very grateful for the trust he put in me, and his constant support and dedication have been decisive in the successful completion of this project.
I would also like to extend my sincere gratitude to my advisor, Dr. Jochen Cremer. His insights and advice have provided me with much-needed clarity and direction navigating the complexity of this topic.
Finally, I would like to thank my family and friends for their support and understanding throughout this process. Their encouragement has been a great source of strength and motivation.

Oscar Lopez Romero
Baden, Switzerland, October 2024

Abstract

The rapid shift toward renewable energy has positioned solar power as a key player in reducing carbon emissions. Yet, the inherent variability of solar irradiance, in particular abrupt fluctuations caused by local cloud movements, poses significant challenges for grid stability and hinders the large-scale adoption of this technology. Accurate short-term forecasting of solar irradiance becomes crucial to mitigate these issues.

Traditional forecasting methods, such as Numerical Weather Prediction, lack the spatial and temporal resolution required to predict these sudden changes in real time. To address this gap, we propose a novel generative AI pipeline that employs diffusion models to predict future sky conditions using ground-based sky images. These predicted images are processed by a convolutional neural network to forecast the solar irradiance reaching the ground.

We benchmark our approach against existing machine learning and traditional forecasting techniques, showing promising improvements in predicting short-term irradiance, particularly during dynamic situations. Additionally, we explore the role of stochasticity in diffusion models, and develop a probabilistic framework that generates full probability distributions rather than single-point predictions. This allows our method not only to deliver robust predictive performance but also to quantify the uncertainty associated with each prediction.

Contents

List of Figures

List of Tables

Acronyms

AFNO	Adaptive Fourier Neural Operator
AI	Artificial Intelligence
AR	Autoregression
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
AWS	Amazon Web Services
CNN	Convolutional Neural Network
ConvGRU	Convolutional Gated Recurrent Unit
ConvLSTM	Convolutional Long Short-Term Memory
CSI	Clear-Sky Index
DGMR	Deep Generative Model of Rainfall
DHI	Diffuse Horizontal Irradiance
DM	Diffusion Model
DNI	Direct Normal Irradiance
ECMWF	European Centre for Medium-range Weather Forecasts
FC	Fully Connected (Neural Network Layer)
GAN	Generative Adversarial Network
GD	Gradient Descent
GenAI	Generative Artificial Intelligence
GHI	Global Horizontal Irradiance
GNN	Graph Neural Network
GPU	Graphics Processing Unit
HDR	High Dynamic Range
HREF	High Resolution Ensemble Forecast
HRRR	High-Resolution Rapid Refresh
IFS	Integrated Forecasting System
KLD	Kullback–Leibler Divergence
LDM	Latent Diffusion Model
LSTM	Long Short-Term Memory
MA	Moving Average
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error

NLL	Negative Log-Likelyhood
NLP	Natural Language Processing
NN	Neural Network
NWP	Numerical Weather Prediction
PDE	Partial Differential Equation
PV	Photovoltaic
RBR	Red Blue Ratio
ResNet	Residual Neural Network
RGB	Red Green Blue (color model)
RNN	Recurrent Neural Network
SSIM	Structural Similarity Index
VAE	Variational Autoencoder
ViT	Vision Transformer
VLB	Variational Lower Bound

1 Introduction

1.1. Background and motivation

The energy sector is undergoing an unprecedented transformation. Countries, companies, and individuals are rapidly trying to reduce their carbon emissions by improving energy efficiency, reducing carbon intensive activities, and sourcing energy from renewable sources. Solar power, abundant and with a relatively low environmental impact, is one of the most mature and cost-effective renewable energy technologies [1], and is set to be one of the cornerstones of this transition. However, the inherent intermittency of solar irradiance poses a challenge to the large-scale adoption of this technology. Cloud cover, air particles, or outside temperature are some of the factors that affect the output of a Photovoltaic (PV) power plant. Smooth changes do not pose a big problem; however, abrupt variations in the power output of a PV plant introduce electrical transients, harmonics, and voltage fluctuations that can compromise the reliability and stability of the entire electrical system [2]. Furthermore, these unforeseen deviations in energy production can introduce high volatility in energy markets, resulting in large price swings.

There are several ways to address the issue. One of the most promising ones is to install battery packs next to PV plants that can be charged and discharged when necessary to smooth out abrupt changes in power output, thereby certain predefined ramp rate limitations are respected. However, to enable this, accurate short-term forecasts of solar irradiance are required. Traditional weather forecasting methods, such as Numerical Weather Prediction (NWP), lack the spatial and temporal resolution needed for real-time applications and often struggle to predict short-term changes in weather. This is because satellite imagery and radar data, ubiquitous in weather forecasting, do not capture fine details necessary for precise irradiance predictions at specific locations. Furthermore, NWP requires long data assimilation steps that take several hours, limiting its usefulness in short-term applications.

This has led researchers to find alternative ways to predict short-term changes in weather. Vastly different methods have been tried, from traditional statistics to computer vision and Machine Learning (ML) [3]. However, the high-level framework is often similar. A typical workflow involves the collection of images, the identification of clouds, the derivation of cloud movements, and a forecasting step that produces an irradiance prediction [4].

Following this framework and inspired by recent advances in Generative Artificial Intelligence (GenAI), we present a diffusion-based pipeline that uses fisheye images of the sky above the location of interest to predict possible future sky conditions, which are processed by a Convolutional Neural Network (CNN) to forecast the irradiance reaching the ground.

1.2. Research Direction and Thesis Outline

As introduced above, the main contribution of this work is the development of a generative AI pipeline capable of producing accurate short-term irradiance forecasts. In particular, we focus on Diffusion Models (DMs), given their recent success in several vision tasks, such as text-to-image generation [5], image restoration and enhancement [6], or video prediction [7]. Furthermore, a comprehensive litera-

ture study showed that, to the best of our knowledge, despite the large number of ML-infused weather prediction models, a relatively small amount deals with irradiance nowcasting, and none of them employed a DM backbone.
This overarching objective opens up several possible research directions and sub-questions. For instance, one approach could involve identifying the optimal neural network architecture, dedicating our time and resources to training models with varying structures and hyperparameters. Another approach might focus on a single model, thoroughly studying it's performance by evaluating different prediction horizons, studying how the stochasticity of diffusion affects the results, examining the effectiveness of recursive inference (the model is fed its own predictions to generate predictions further ahead into the future)... Alternatively, we could also undertake a more theoretical study, dive deep into the literature, repurpose published models for the task at hand, look at their strong and weak points, etc. Given the bounded timeline and computational resources, we cannot pursue all of these paths in full depth. However, we reach a compromise that allows us to present a model that yields satisfactory accuracy, study its performance under different conditions, and compare it against other ML models as well as traditional methods.
In essence, our work will attempt to answer the following questions.

- What are current state-of-the-art techniques in weather forecasting using ML and how do they compare to traditional methods?
- Do latent diffusion models improve the accuracy of predicting short-term irradiance? How do they perform in stable and challenging conditions?
- How to design a diffusion model for the task of irradiance nowcasting? How to use past observations to condition the denoising process?
- Can we inference the model recursively to arbitrarily extend the prediction horizon? How does this affect its predictive performance?
- How does the inherent uncertainty of the denoising process affect the results? Can it be leveraged to quantify uncertainty in the model predictions?

The thesis is organized as follows. Chapter 2 consists of a technical background that explains the necessary concepts to follow along. In particular, we introduce some traditional methods for weather forecasting, several ML concepts, and some basic notions of photovoltaic power. Furthermore, this chapter also deals with related work, that is, publications about data-driven weather forecasting models relevant or similar to this project. Chapter 3 deals with the data used in this project, we explain how the data was acquired, what are its particularities, and what processing steps were applied. In chapter 4 we present our model, introduce the architecture, the design decisions and how it is implemented in practice. Chapter 5 showcases the results, we compare our model to several benchmarks and display its strengths and weaknesses. Finally, in Chapter 6, we conclude the project, revisit the research questions, highlight our contributions, and suggest directions for future research.

2 Technical Background

2.1. Traditional Methods for Weather Forecasting

The approach used for weather forecasting and solar irradiance prediction largely depends on the spatial and temporal resolutions required for the end application. This section introduces three of the most common methods, namely NWP, optical flow, and statistical methods.

2.1.1. Numerical Weather Prediction

The roots of numerical weather prediction go back to the 1920s when pioneering scientists such as Lewis Fry Richardson and Jule Gregory Charney laid the foundations for numerical simulations of atmospheric processes [8]. The advent of computers in the latter half of the twentieth century revolutionized NWP, enabling the development of complex numerical models capable of simulating atmospheric behavior with increasing fidelity.
At its core, NWP relies on solving a set of partial differential equations that model the behaviour of the ocean and the atmosphere [9]. These equations are discretized over a domain covering the whole planet, or sections thereof, and solved with time resolutions ranging from a few hours to several days and spatial resolutions as little as 10 km (Fig. 2.1).

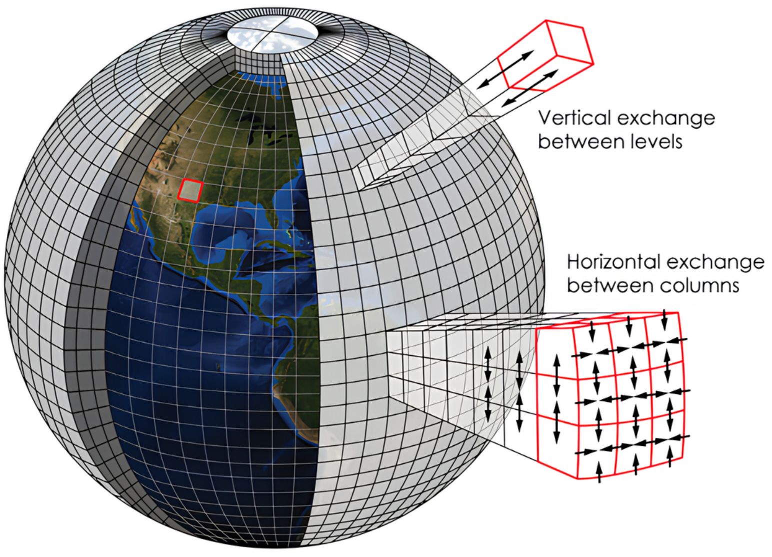

Vertical exchange between levels
Horizontal exchange between columns

Figure 2.1: Earth system discretization for NWP. Modified from [10]

A critical aspect of NWP is the initialization of the models. NWP is initialized using observational data

from a variety of sources, including satellites, weather stations, and radars. This data is preprocessed and merged with predictions from other models in a step called data assimilation, which seeks to find the best possible estimate of the true atmospheric state [11].
The computational requirements of NWP are substantial. Manipulating the vast datasets and solving the complex computations requires high-performance clusters only accessible to a handful of weather agencies in the world. Furthermore, the long running times and the time needed for data assimilation make NWP not suitable for very short-term predictions [12].
The main challenge faced by NWP models lies in the chaotic nature of the atmosphere. This fundamental characteristic of weather systems, famously described by Edward Norton Lorenz as the ”butterfly effect”, refers to extreme sensitivity to small variations in the initial conditions. In other words, small differences in the initial state can lead to big differences in the forecast outcomes over time [13]. This is aggravated by the fact that since the set of partial differential equations is solved numerically, small errors will always be present and compound over time.
To account for the uncertainty in the forecast outcomes of NWP, it is common to use ensembles. This approach merges the output of several different models, as well as equal models with slight differences in the initial conditions, and produces a confidence value in addition to the forecast [14]. However, this comes at the cost of even larger computational resources.

2.1.2. Optical Flow

A very common approach for irradiance forecasting involves the identification and monitoring of cloud movements using sky images. By tracking the movement of clouds, these vision-based algorithms can predict when clouds might obstruct the sun, affecting photovoltaic production.
In particular, optical flow is a method used to analyze the motion of objects within a sequence of images. First, clouds are detected using a variety of image processing techniques [15]. Then, a motion detection algorithm derives the motion vectors from consecutive frames, which can be used to map the position of clouds in the future.
One of the simplest and most common approaches to identify whether a pixel shows a clear sky or a cloud is the Red Blue Ratio (RBR) threshold. The RBR is defined as the intensity in the red channel divided by the intensity in the blue channel (Equation 2.1).

$$RBR = \frac{R}{B} \tag{2.1}$$

This method leverages the fact that clouds typically appear much brighter in the blue channel compared to the red channel, and simply consists in classifying a pixel as cloudy if the RBR exceeds a certain value (Fig. 2.2). This method is capable of reliably detecting white opaque clouds, however thick black or thin clouds are hard to distinguish [3].

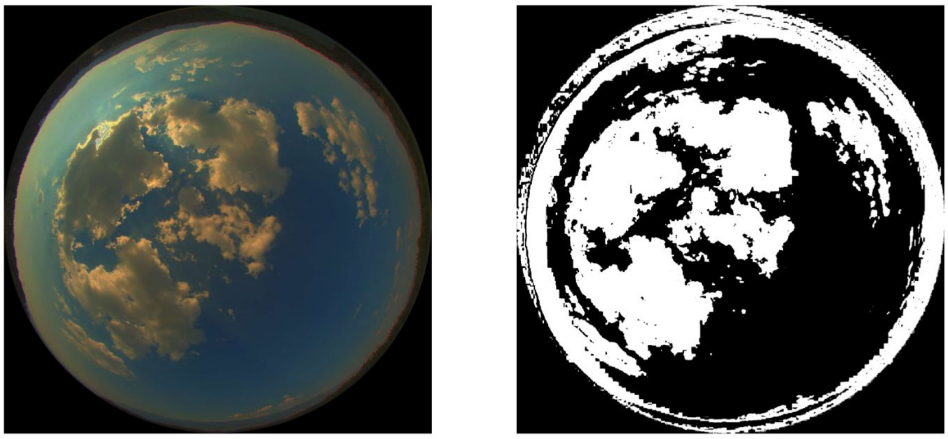
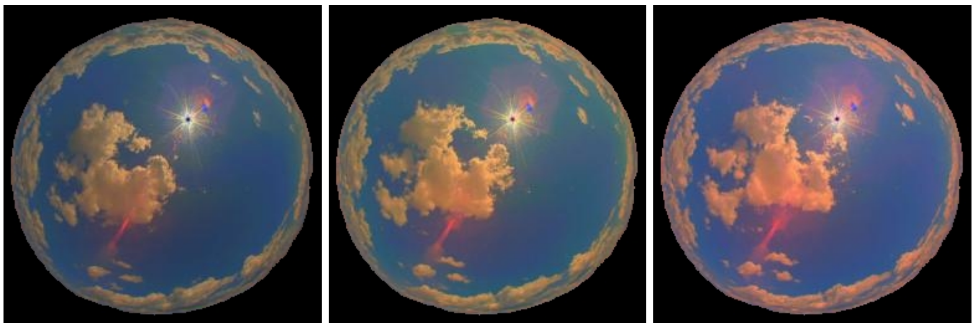
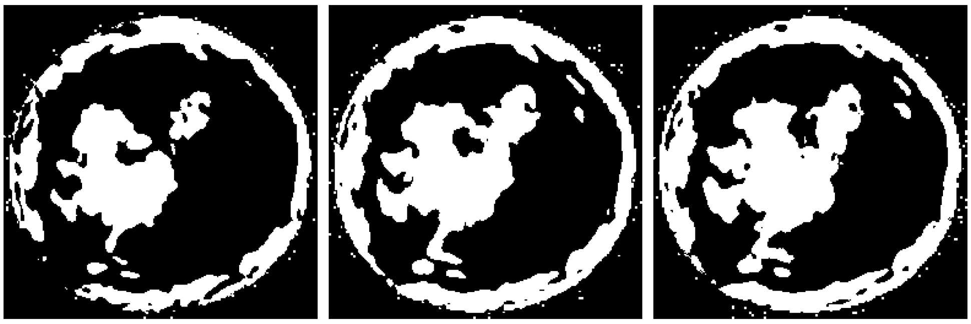


Figure 2.2: Example of cloud detection using RBR threshold.

Once the clouds have been segmented, a sequence of frames is used to compute their motion. The most widely used methods for optical flow estimation are the Farnerback method [16] and the Lucas-Kanade method [17]. Explaining how these methods work is beyond the scope of this project, hence the interested reader is referred to the original sources. Open-source implementations of both methods are available online. Figure 2.3 shows an example of the whole process where we take a sequence of frames, binarize them using the RBR threshold, and derive the motion vectors of the clouds using the Farnerback algorithm implementation of OpenCV [18].


(a) Original images.


(b) Images after applying RBR threshold.

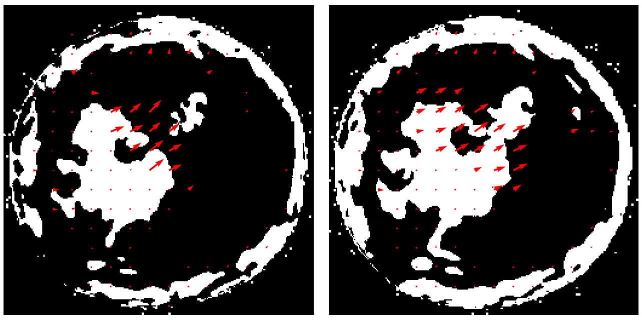

(c) Motion vectors derived using Farnerback optical flow algorithm.

Figure 2.3: Optical flow method

A natural next step would involve using the motion vectors to forecast the position of the clouds in a future instant, and checking whether the sun is occluded or visible. This would already be able to produce a binary prediction of high or low irradiance, which can be merged with past observations or additional information to produce an actual number. Alternatively, the images resulting from the optical flow algorithm can be fed to image processing methods in a hybrid setup that directly yields an irradiance prediction.

2.1.3. Statistical Methods

This class of methods aims to predict future weather using only past observations. They make the key assumption that there exists a causality relationship between time and a specific weather variable, hence they are also referred to as time series models [19].
The simplest of the time series models is called persistence. The idea behind it is that the changes

in irradiance are smooth; therefore, the most reliable prediction for the next time step is the irradiance value in the current instant. In short-term weather forecasting, persistence is often a surprisingly hard to beat baseline. Indeed, weather conditions tend to change gradually rather than abruptly, so most of the time predicting that the irradiance will remain the same yields very high accuracy.
An advanced version of this method consists in assuming that the meteorological factors that affect irradiance at time t will remain the same at time $t+1$. However, some known variations, such as the Sun's zenith and azimuth angles, are taken into account. This kind of model, called physics-informed persistence, also produces very accurate forecasts [20].
Autoregressive methods are also often used to forecast weather variables. One of the simplest, the Autoregressive Moving Average (ARMA), consists of two components (two polynomials), one for the Autoregression (AR) and one for the Moving Average (MA), with order p and q respectively. The AR part (Eq. 2.2) tries to predict future values based on previous observations, while the MA (Eq. 2.3) includes information on past forecasting errors. The drawback of ARMA is that it can only be used if the time series to be predicted is stationary, which is not the case in irradiance time-series, hence a detrending step would also be necessary [21].

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \tag{2.2}$$

$$X_t = \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{2.3}$$

Where X_t is the observation at time t, $\varphi_1 \ldots \varphi_p$ and $\theta_1 \ldots \theta_q$ are model parameters to be tuned, and ε is white noise.
Combining both parts, we have the ARMA model (Eq. 2.4).

$$X_t = \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{2.4}$$

The Autoregressive Integrated Moving Average (ARIMA) model extends the ARMA by introducing an Integrated (I) component, which allows the model to handle non-stationary time series data [22].
The most important aspect of any statistical model is the estimation of its parameters. Several algorithms exist for this purpose, such as Yule Walker, Least Squares, or Maximum Likelihood.

2.2. Machine Learning

This section introduces the necessary ML concepts for the understanding of this work. We assume prior knowledge of the fundamentals of deep learning, namely Neural Networks (NNs), loss functions, Gradient Descent (GD), and backpropagation. There are several excellent textbooks and resources covering these concepts. For example [23] provides a broad but comprehensive introduction to Artificial Intelligence (AI), while [24], [25] tackle deep learning and some topics in computer vision and natural language processing.
In what follows, we will cover CNNs, Residual Neural Networks (ResNets), attention, DMs, autoencoders, and U-Nets.

2.2.1. Convolutional Neural Networks

CNNs are the most prominent network architecture for computer vision and image processing tasks. CNNs exploit three properties of images that make them unsuitable for fully connected networks. First, images have very large input dimensions. A typical image for a classification task might contain 256x256 RGB values, i.e. 196608 values. This means even for relatively shallow fully connected networks, the number of weights quickly becomes computationally intractable. Second, nearby pixels are statistically related, but fully connected layers process each value independently, without any notion of spatial closeness. Third, the interpretation of an image does not change under geometric transformations. An image of a cat is still an image of a cat if we shift it rightward by a few pixels [23].
Convolutional layers process regions of an image independently, using parameters that are shared across the entire image. They exploit the relationship of nearby pixels, use far fewer parameters than fully connected networks, and interpret groups of pixels equally regardless of their position in the image

[23].
Mathematically, the convolution operation can be interpreted as the application of a sliding window to a matrix of pixels. This sliding window is called a filter or kernel. For example, a 3x3 kernel applied to an image with pixels x_{ij} computes a single layer of hidden units h_{ij} as:

$$h_{ij} = \mathrm{a}\left[\beta + \sum_{m=1}^{3}\sum_{n=1}^{3} \omega_{mn} x_{i+m-2,j+n-2}\right] \tag{2.5}$$

where w_{mn} are the entries of the kernel. This is simply a weighted sum over a square 3x3 input region [23]. As seen in Fig. 2.4 the kernel is translated horizontally and vertically to create an output at each position.

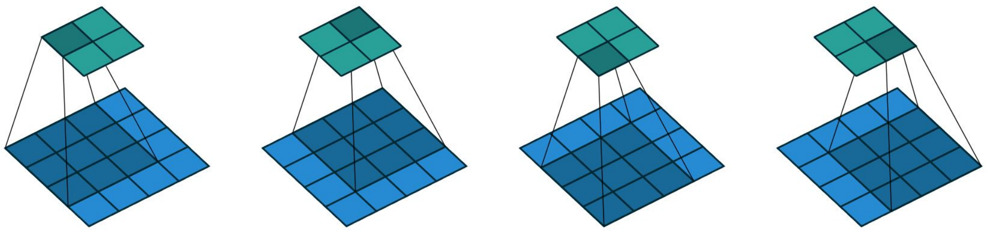


Figure 2.4: Convolution without padding and stride of 1 [26]

In addition to the kernel size, the convolution operation has another two important hyperparameters, namely padding and strides. Padding consists of adding artificial pixels (usually of value 0) to the edges of the image. In this way, we can preserve the spatial dimensions of the input and prevent the loss of information at the borders (without padding, the edge pixels are only taken into the convolution operation once) (Fig. 2.5). Strides refers to the step size with which the kernel moves across the input image during the convolution operation. When applying a convolutional filter, usually the filter is moved one pixel at a time. However, strides allow the filter to skip some of the locations and take larger steps (Fig. 2.6). A larger stride reduces the spatial dimension of the output.

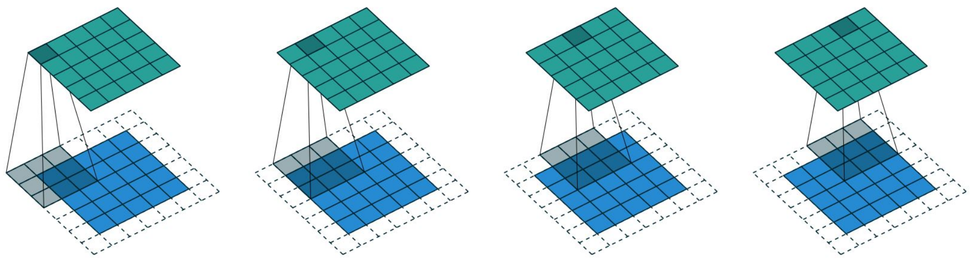


Figure 2.5: Convolution with padding of 1 and stride of 1 [26]

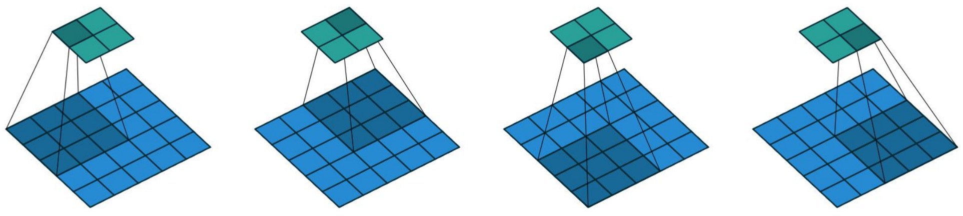


Figure 2.6: Convolution without padding and stride of 2 [26]

The notion of convolution can be generalized to any dimension. For instance in a 3D convolution the kernel would be a cuboid and would slide accross the height, width and depth of the input.

Usually, CNNs also contain pooling layers to scale down the feature maps generated by the convolutional operations. The two most common pooling methods are max pooling and average pooling (Fig. 2.7). In max pooling, a sliding window moves across the input, and at each position selects the maximum value, and discards the rest. Similarly, in average pooling, the output is the average value of the elements within each window. These processes effectively reduce the spatial dimensions of the input feature map while preserving the most important information.

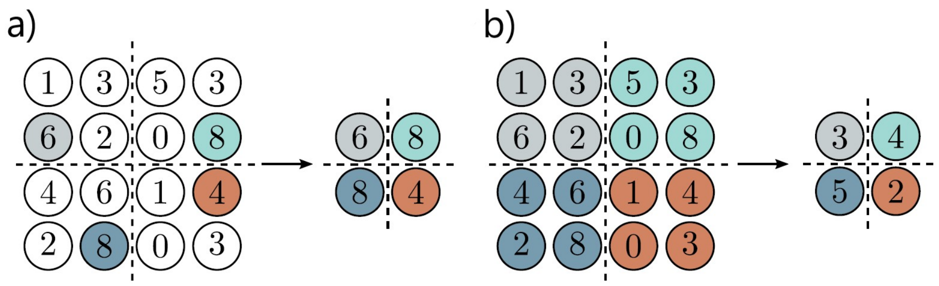


Figure 2.7: Methods for scaling down representations. a) Max pooling. b) Average pooling. Modified from [23]

In some architectures, in particular when the output is an image, it's often needed to increase the spatial resolution of feature maps. There are several methods for upsampling in CNNs. The simplest way is the nearest-neighbor approximation, which copies the closest value to interpolate new pixel values (Fig. 2.8a). Alternatively, we can use max unpooling, the inverse operation of max pooling, in which the values are placed to the locations where they originated (Fig. 2.8 b), or bilinear approximation [27] (Fig. 2.8 c).

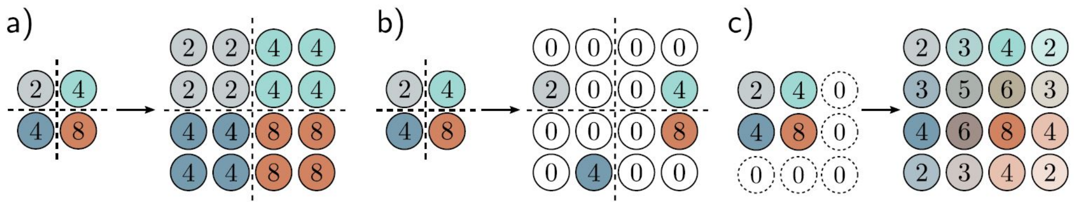


Figure 2.8: Methods for scaling up representations. a) Nearest neighbor interpolation. b) Max unpooling. c) Bilinear interpolation [23]

A fourth approach is to apply the inverse operation of a convolution, called deconvolution or transpose convolution. In a transposed convolution, a learnable filter is applied to the input feature map, similar to a regular convolution. However, instead of sliding the kernel over the input, a deconvolutional layer slides the input over the kernel and performs element-wise multiplication and summation (Fig 2.9). This results in an output that has greater spatial resolution than the input. Similar to conventional convolutions, transpose convolutions can also have stride greater than one and padding [26].

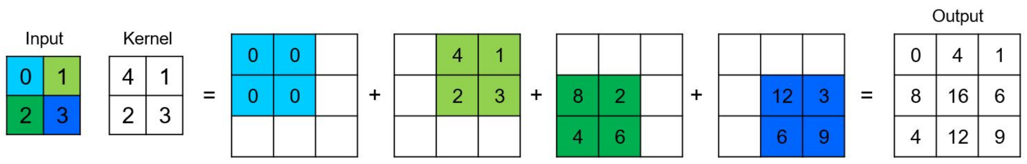


Figure 2.9: Example of transpose convolution.

2.2.2. Residual Networks

NNs have shown remarkable performance in a variety of tasks, ranging from image recognition to natural language processing. However, as these networks become deeper, they face challenges such as

degradation and vanishing gradients. Degradation refers to a decrease in performance observed in NNs when adding more layers. The problem of vanishing gradients occurs when during backpropagation gradients become too small to effectively update the parameters of the initial layers of the network [28].
ResNets were proposed to address these limitations [29]. ResNets employ a novel architectural design that introduces skip connections which facilitate the flow of gradients during backpropagation. This alleviates the vanishing gradient problem and enables training of very deep networks. In particular, the key idea behind ResNets is the introduction of residual blocks (Fig. 2.10). Residual blocks consist of a sequence of layers followed by a skip connection that adds the original input to the output of the layers.

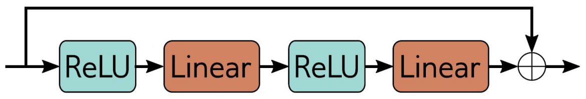


Figure 2.10: Example residual block [23].

Residual blocks have been found not only to solve the problem of vanishing gradients, but also to improve the overall performance of NNs. This relies on the premise that fitting a residual mapping is easier than doing so to the underlying input-output mapping. Let us consider a residual block with input x, and the transformation $F(x)$ applied by the layers within the block. The output y is computed as follows:

$$y = F(x) + x \tag{2.6}$$

Here, the sum of $F(x)$ and x creates a residual connection that allows the network to learn the residual mapping $F(x)$ rather than the desired underlying mapping. This turns out to be a much easier task for different reasons. First, the network can easily learn an identity mapping by setting $F(x) = 0$, which means it can choose not to perform any transformation. Second, by learning the residual, the network focuses on capturing the difference between the input and the target output. This can make the learning task easier because the network only needs to refine or adjust the input features rather than completely reconstruct them.
ResNets do not have the problem of vanishing gradients. However, they suffer particularly from unstable forward propagation and exploding gradients. This is usually handled by adding batch normalization layers (BatchNorm).
BatchNorm is a technique that consists in adding an operation to zero-center and normalize each activation h across a given batch $\mathcal{B}$ [30]. First, the mean m_h and standard deviation s_h are calculated.

$$m_h = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} h_i \tag{2.7}$$

$$s_h = \sqrt{\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (h_i - m_h)^2} \tag{2.8}$$

Then we use these values to standardize the batch activations to have zero mean and unit variance [23]:

$$h_i \leftarrow \frac{h_i - m_h}{s_h + \epsilon} \quad \forall i \in \mathcal{B} \tag{2.9}$$

where ϵ is a small value to prevent division by zero.
Lastly, the normalized variable is scaled by γ and shifted by δ, which are parameters learnt during training.

$$h_i \leftarrow \gamma h_i + \delta \quad \forall i \in \mathcal{B} \tag{2.10}$$

At test time, we do not have a batch from which to compute statistics. To address this, m_h and s_h are calculated over the entire training dataset and frozen in the final network.

2.2.3. Attention

Attention was first introduced in 2014 in the context of Natural Language Processing (NLP) to improve machine translation [31]. A few years later, the introduction of the transformer architecture [32] marked a turning point for the field, and still nowadays transformer-like architectures or models with attention mechanisms are state of the art in several tasks and benchmarks [33]. The enormous success in NLP prompted researchers to try and apply attention mechanisms to computer vision, ultimately leading to the creation of the Vision Transformer (ViT) [34].
The core idea of attention is to allow the model to selectively focus on different parts of the input by using a set of learned weights, which indicate how relevant each part is to each other. In vision in particular, attention enables models to prioritize the most informative regions of an image and discard the less informative ones. The standard form of attention is called dot-product self-attention, or simply self-attention and can be described as follows.
First, the input features are transformed into three sets of vectors: queries (q), keys (k), and values (v); which are learned by standard linear layers.

$$q = W_q x \tag{2.11}$$

$$k = W_k x \tag{2.12}$$

$$v = W_v x \tag{2.13}$$

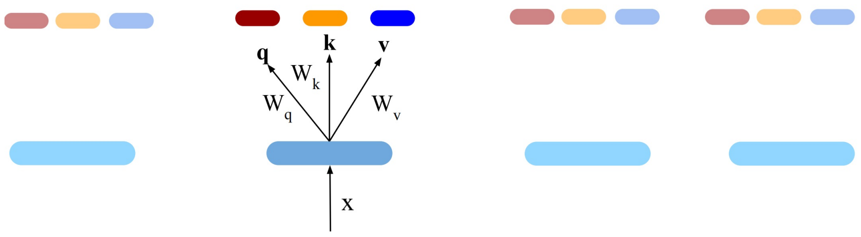


Figure 2.11: The first step of attention is computing the keys, queries and values. Modified from [35]

Then, we compute the dot product between the keys and the queries, and pass the results through a softmax function so that they are positive and sum to one. The values obtained are called attention weights λ.

$$\lambda_i = softmax[q \cdot k_i] = \frac{e^{q \cdot k_i}}{\sum_{i=1}^{4} e^{q \cdot k_i}} \tag{2.14}$$

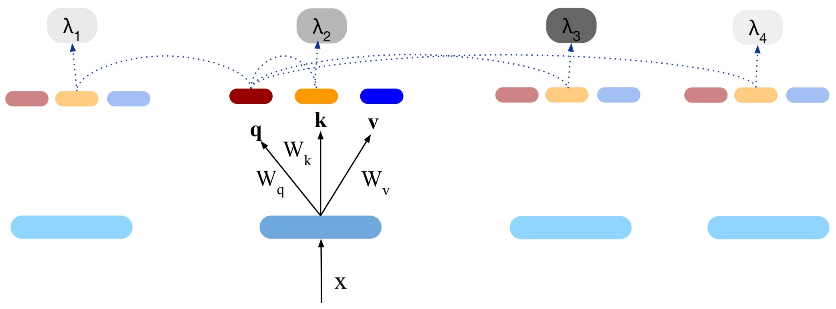


Figure 2.12: The second step consists in computing the attention weights. Modified from [35]

The final output of the attention mechanism is called the context vector z and is a weighted sum of the value vectors, where the weights are the attention weights.

$$z = \sum_{i=1}^{4} \lambda_i \cdot v_i \tag{2.15}$$

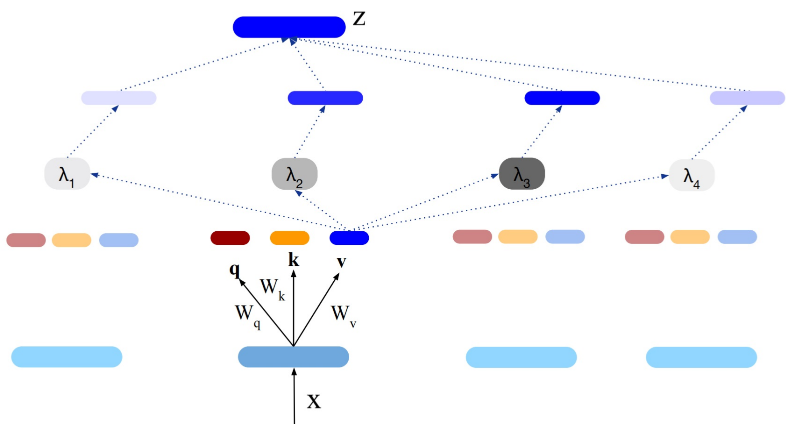


Figure 2.13: The context vector is a weighted sum of the value vectors. Modified from [35]

There are extensions to self-attention that are always used in practice. The first one is called positional encodings. Positional encodings were introduced because the basic form of self-attention discards crucial information: the computation is the same regardless of the order of the input sequence. The way positional encodings address this is by introducing a matrix Π, which encodes positional information. Each column of Π is unique, providing information about the position of each element within the input sequence. This matrix can be either manually designed or learned. Depending on the model, Π can be added to the network input or in every layer [23].
It is also common practice to scale the dot products of the attention computation by the square root of of the dimension of the queries and keys D.

$$\lambda_i = Softmax\left[\frac{q \cdot k_i}{\sqrt{D}}\right] \tag{2.16}$$

Finally, self-attention is always implemented so that it is computed many times in parallel. This is called multi-head self-attention. Now, H different sets of values, keys and queries are computed.

$$q_h = W_{qh}x \tag{2.17}$$

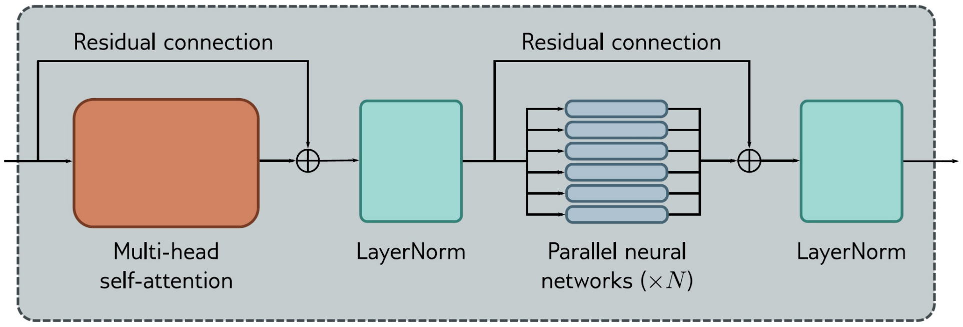
$$k_h = W_{kh}x \tag{2.18}$$

$$v_h = W_{vh}x \tag{2.19}$$

The h^{th} head outputs the context matrix Z_h as follows.

$$Z_h = V_h \cdot Softmax\left[\frac{K_h^T Q_h}{\sqrt{D}}\right] \tag{2.20}$$

Then, the output of the different heads is concatenated and a linear transformation is applied to combine them.
Multi-head self-attention is just one component of a transformer layer, which also includes residual connections, fully connected layers, and normalization layers, as seen in Fig 2.14, for a more in-depth treatment of transformers, the reader is referred to [23].



Residual connection
Residual connection
Multi-head self-attention
LayerNorm
Parallel neural networks ($\times N$)
LayerNorm

Figure 2.14: Transformer layer [23]

The idea of applying transformers to computer vision is not necessarily a promising one. As explained in Section 2.2.1, CNNs are naturally suited for image data due to their inherent properties, which transformers lack. CNNs have built-in translational invariance, while transformers must learn this feature. Moreover, CNNs use far fewer parameters than fully connected networks. In contrast, transformers face a practical bottleneck due to the large number of pixels in images and the quadratic complexity of self-attention mechanisms. Despite these apparent disadvantages, transformers have eclipsed the performance of CNNs for image-related tasks [23].

2.2.4. Diffusion Models

DMs have recently emerged as a powerful class of deep generative models. They have been successfully applied to a diverse range of tasks including image and video generation, weather forecasting, image segmentation, and protein synthesis, among others [36].
DMs aim to generate samples from an unknown data distribution by learning to reverse a nosing process defined by a Markov chain. The idea, inspired by non-equilibrium thermodynamics [37], is to iteratively destroy structure in a data distribution by adding Gaussian noise until the samples are indistinguishable from an isotropic Gaussian distribution. DMs learn a reverse diffusion process that can undo this operation, yielding a highly flexible generative model of the data.
More formally, given an input image x_0 sampled from an unknown data distribution $q(x)$ we define a *forward diffusion process* in which small amounts of Gaussian noise are added to the sample in T steps, producing a sequence of noisy samples $x_1, ..., x_T$. The amount of noise added in each step is controlled by a variance schedule $\beta_t \in (0, 1)$ [38], [39].

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{1-\beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \tag{2.21}$$

$$q\left(\mathbf{x}_{1: T} \mid \mathbf{x}_0\right)=\prod_{t=1}^T q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) \tag{2.22}$$

$$x_t=\sqrt{1-\beta_t} x_{t-1}+\beta_t \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \mathbf{I}) \tag{2.23}$$

Given sufficiently large T and a well behaved schedule, x_T is equivalent to pure noise sampled from $\mathcal{N}(0, \mathbf{I})$.

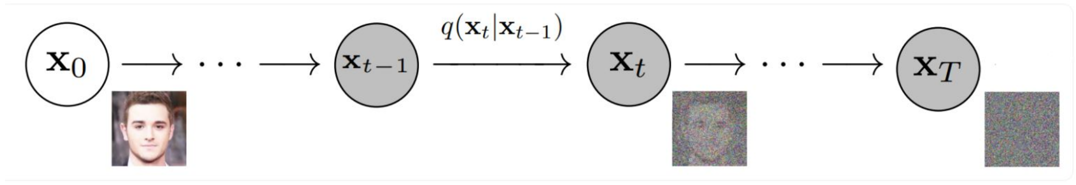


Figure 2.15: Forward diffusion process [40].

A notable property of the forward process is that we can sample x_t at any arbitrary time step t without the need to perform t iterations of (2.23) with the following reparametrization. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \Pi_{i=1}^t \alpha_i$, then we have:

$$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{\bar{\alpha}_t} \mathbf{x}_0,\left(1-\bar{\alpha}_t\right) \mathbf{I}\right) \tag{2.24}$$

$$x_t=\sqrt{\bar{\alpha}_t} x_0+\sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon} \tag{2.25}$$

If we can reverse the forward process and sample from $q(x_{t-1}|x_t)$ we will be able to generate images resembling our training data starting from pure Gaussian noise $x_T \sim \mathcal{N}(0, \mathbf{I})$. Assuming β_t is small enough, the reverse step $q(x_{t-1}|x_t)$ will also be Gaussian, hence we only need to estimate the mean and the variance. Direct estimation of $q(x_{t-1}|x_t)$ is intractable, but we can approximate it with a parameterized model p_θ. We can further decompose this model into two, $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ that respectively parameterize the mean and the variance.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \tag{2.26}$$

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{2.27}$$

Where $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are NNs that can be trained using the Variational Lower Bound (VLB).

$$L_{\mathrm{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0) \tag{2.28}$$

This loss needs to be refactorized into a combination of several Kullback–Leibler Divergence (KLD) and entropy terms to be analytically computable. However, [39] finds that training the DM works better on a simplified loss that only contains the squared error between the sampled and predicted noise.

$$L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right\|^2\right] \tag{2.29}$$

Note that to formulate this loss function, we have substituted the model $\boldsymbol{\mu}_\theta$ by $\boldsymbol{\epsilon}_\theta$ to predict the added noise ϵ_t instead of the mean $\tilde{\mu_t}$. Furthermore, the model $\boldsymbol{\Sigma}_\theta$, originally aimed at predicting the variance $\tilde{\beta}_t$, has been removed and the variance is fixed instead to σ^2. This was found to lead to more stable training and better sample quality [39].

In practice, the training and sampling algorithms of DMs are as follows (Alg. 1 and Alg. 2 respectively [39]). The training loop starts by selecting a random sample $\mathbf{x}_0$ from our data, sampling a random time step t (generally $t \in [1, 1000]$), and sampling Gaussian noise $\boldsymbol{\epsilon}$. Subsequently, we generate the noisy data x_t according to Eq. 2.25. Then, we feed our model x_t and the timestep t (including the time step in the model input is a design choice by the authors, who claim it improves accuracy). The output of the model is compared to the actual noise that we added $\boldsymbol{\epsilon}$ by means of the Mean Squared Error (MSE), which is used as the loss function to take a gradient descent step. This process continues until the network stops improving, that is, training converges. When working with images, the resulting sample might violate the bounds of the pixel space, hence truncation is sometimes necessary.

On the other hand, the sampling or inference of the DM is as follows. We start by sampling pure Gaussian noise $\mathbf{x}_T$, which we will iteratively denoise. In each iteration, we subtract the noise predicted by the model from the current sample x_t to generate an estimate of x_0. Then, we add back most of the noise, obtaining x_{t-1}. We repeat this process until we have a fully denoised sample, at $t = 0$. In order to speed up the generation process it is common to iterate not over the original 1000 steps of the noising process but over a subset of them, usually one of every 20 ($t = 1000, 980, 960...$) [39].

Algorithm 1 Training

```
1: repeat
2:     $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:     $t \sim \text{Uniform}(\{1, \ldots, T\})$
4:     $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:     Take gradient descent step on
6:         $\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2$
7: until converged
```

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \ldots, 1$ do
3:     $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:     $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

So far, we have not concerned ourselves with what particular models are used in practice to predict the noise. This is quite an active field of research and new architectures are published frequently. However given the formulation, it is clear that it needs to be a model in which the output has the same dimension as the input. The most common architecture is the U-Net, which we present in Subsection 2.2.6.
After Ho et al. published their seminal paper on DMs [39], on which the previous explanations are based, several improvements and modifications have been presented. Some of these include improving the noise schedule [41], learning the variance of the reverse diffusion process [42], improving the denoising equation [43], conditioning the reverse process on class labels [5], conditioning the models on text prompts [44]–[46], using a ViT as the backbone [47], [48] and combinations thereof.
All of these innovations yielded incremental improvements to the performance and fidelity of DMs, however the biggest leap forward in the history of DMs came with the invention of Latent Diffusion Models (LDMs) [49]. LDMs follow the same structure of their predecessors, but instead operate in a compressed latent space. This approach was introduced to address the computational inefficiency and memory demands of conventional DMs, especially when dealing with high-resolution images. In practice, this is achieved by mapping the original samples (images) to a lower dimensional space by means of an autoencoder (see Subsection 2.2.5). Then, the DM operates with this latent representation rather than the original pixel values. This approach was found not only to reduce the computational burden but also to produce high-quality samples that are comparable to, or even better than, those produced by pixel space DMs.

2.2.5. Autoencoders and Variational Autoencoders

Autoencoders are a particular type of neural network trained to copy their input. Given a sample, autoencoders first encode it into a lower-dimensional latent representation and then decode it back to its original dimensions, so that it is as similar as possible to the original. An autoencoder learns to represent data in a compressed and meaningful representation that can be used for several applications. Autoencoders have been applied to clustering, recommendation systems, anomaly detection, or as generative models [50]. As mentioned in Subsection 2.2.4 autoencoders can also be used in combination with DMs to reduce their computational requirements.
Autoencoders consist of two main components: the encoder and the decoder (see Fig. 2.16). The encoder part of the network compresses the input into a latent representation. It achieves this by gradually reducing the dimensionality of the input through its network layers. For image data, this generaly involves strided convolutions and pooling layers. This latent representation can be interpreted as a compact ”summary” of the input data, which captures its most important features.

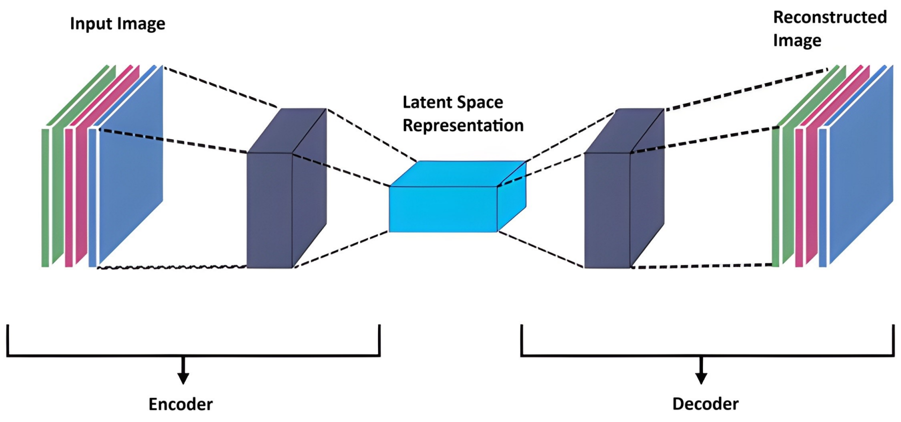


Figure 2.16: Autoencoder high-level architecture. Modified from [51]

The decoder, on the other hand, works to reconstruct the input data from the latent representation. It mirrors the architecture of the encoder but in reverse, progressively increasing the dimensionality of the latent samples back to their original dimensions. To achieve this, the decoder employs transpose convolutions and upsampling layers. The performance of an autoencoder is measured by how closely the decoded outputs match the original inputs, usually measured by the MSE, which is also the training

loss.
One key characteristic of a standard autoencoder is that the mapping from the input to the latent space and back is deterministic. That is, the model tries to minimize the difference between the original sample and its reconstruction by learning a direct encoding and decoding function. However, the latent space does not have any particular structure, it is simply a compressed representation of the input, and the model might not generalize well to unseen samples. To address this, Kingma and Welling introduced the Variational Autoencoder (VAE) [52]. VAEs introduce a probabilistic approach to the latent space. Instead of mapping an input to a single point, a VAE maps an input to a distribution over the latent space. This distribution is generally a Gaussian, which means that the outputs of the encoder branch in a VAE are simply it's mean and variance. The distributions are regularized to be close to a standard normal by including a KLD term in the training loss. In order to obtain a reconstructed sample, we simply need to draw from this distribution and pass it through the decoder, as shown in Fig. 2.17.

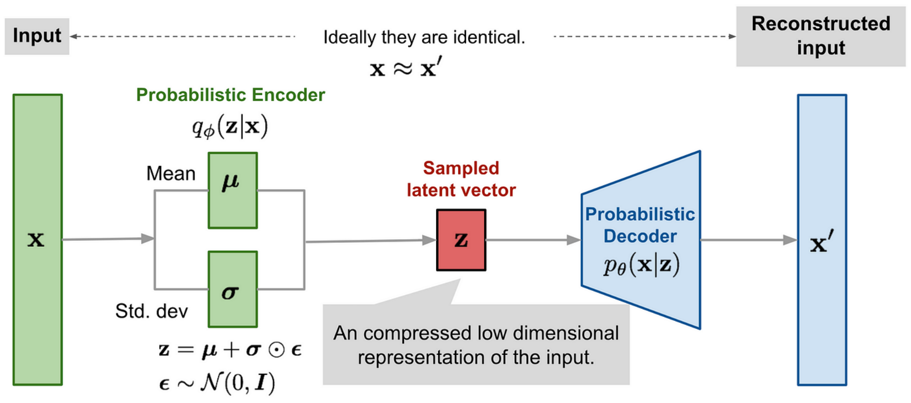

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Figure 2.17: VAE schematic [53]

In general, autoencoders or VAEs are task-agnostic. As mentioned before, the training loss is a measure of the difference between the input and the output and it does not matter whether the autoencoder will be used for weather prediction, medical image processing, or text-to-image generation. For this reason, it is quite common to use pretrained autoencoders, which are freely avalable online. This allows researchers use state-of-the art models that have been trained using clusters of the latest Graphics Processing Units (GPUs), which would be otherwise inaccessible given the large computational requirements. Fig. 2.18, 2.19, and 2.20 show an example of a widely used VAE for diffusion [54], in which the latent space is a factor of 48 smaller than the input dimension.

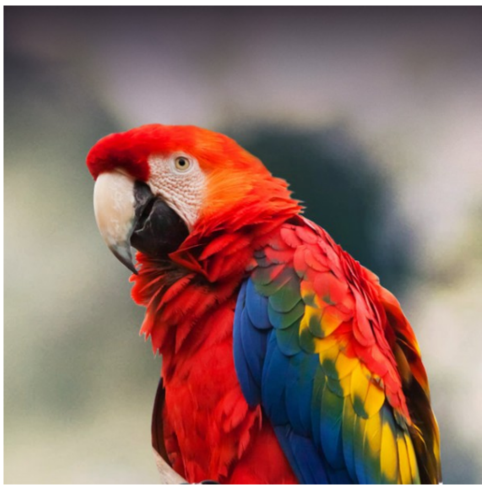


Figure 2.18: Original image of dimension 512x512x3. Courtesy of Lafeber.

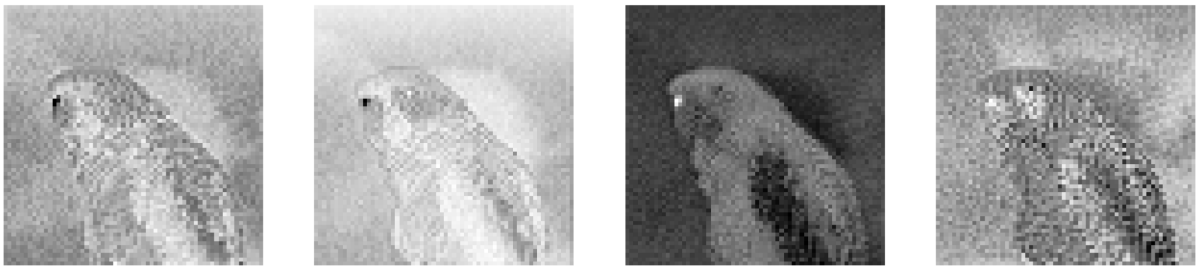


Figure 2.19: Latent representation of dimension 4x64x64. Obtained using the pretrained VAE from [54].

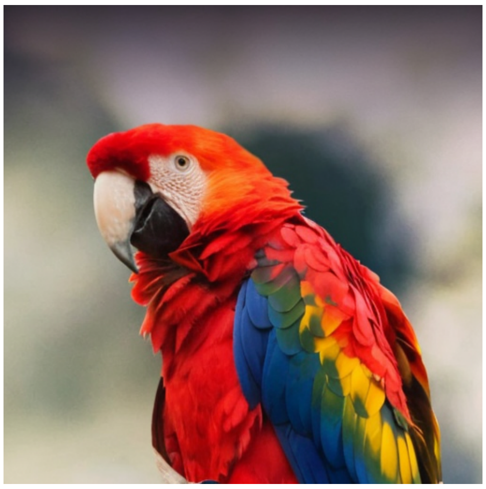


Figure 2.20: Reconstructed image of dimension 512x512x3. Obtained using the pretrained VAE from [54].

2.2.6. U-Nets

The U-Net architecture was originally proposed for biomedical image segmentation [55], but has been successfully applied to a wide range of other tasks. In particular, U-Nets are ubiquitous among DMs where they are employed to parametrize the noise predictor ϵ_θ.

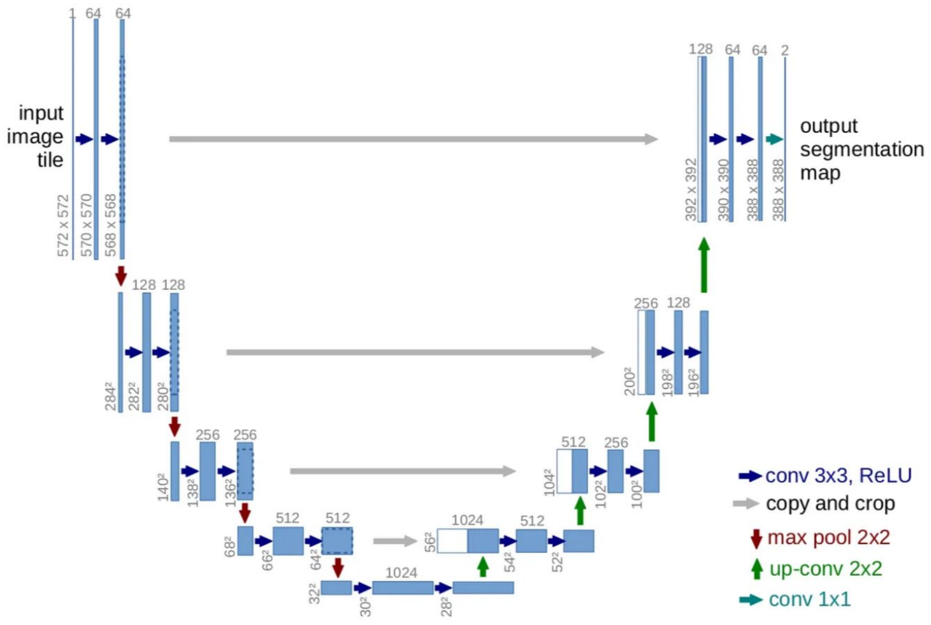


Figure 2.21: Original U-Net from [55].

Similar to autoencoders, U-Nets consist of a contracting path and an expanding path. The contracting path contains a sequence of convolutional and max pooling layers, which reduce the spatial resolution of the image while increasing the number of channels. The expanding path contains a sequence of upsampling and convolutional layers, which successively halve the number of feature channels and double the spatial resolution. Additionally, residual connections are used to concatenate the output of each upsampling block with the corresponding feature map from the contracting path.
Several modifications and enhancements have been proposed to the original U-Net to adapt it to denoising diffusion frameworks. Typically, the U-Net is augmented with self-attention blocks, usually found around the bottleneck. As mentioned in Section 2.2.4, it is also common practice to include the denoising timestep t, and depending on the application the U-Net must be conditioned. Conditioning involves giving the U-Net additional information to guide the denoising process. This ensures that, instead of generating random samples that merely resemble the training data, the model produces outputs that are useful for a given application (e.g. text-to-image models condition the U-Net on a text prompt so that the generated sample depicts the prompt). The particular way to provide conditioning information to the network varies widely between models and applications. The most straightforward method is to append additional information to the input along with the noisy sample.

2.3. Solar Energy

The physical phenomenon that allows the conversion of electromagnetic radiation coming from the sun into electricity is called the photovoltaic effect. An in-depth description of this physical process is beyond the scope of this work, but the reader is referred to [56]. In this section, we instead introduce some concepts related to solar energy that are useful to contextualize and understand this work, namely Solar Irradiance and Clear-sky Models.

2.3.1. Solar Irradiance

The power production of a PV plant is mainly determined by the amount of solar irradiance that reaches its panels. This value is determined by several factors, such as the location, the exact date and time, as well as environmental circumstances. The Sun irradiates approximately $6.3 \cdot 10^7 W$ of energy per square meter. This energy is reduced by the square of the distance, so planet Earth only receives around $1361 W/m^2$, a value called *solar constant* Ec. This value corresponds to the yearly average; however, given the elliptical orbit of the Earth around the Sun, the actual radiation that reaches Earth's outer atmosphere, E_{ext}, on any given day is given by

$$E_{ext} = E_c \left[1 + 0.034 \cdot \cos\left(\frac{360 N_d}{365.25}\right)\right] W/m^2 \tag{2.30}$$

where Nd is the number of the day in the year (January 1st would be 1) [57].

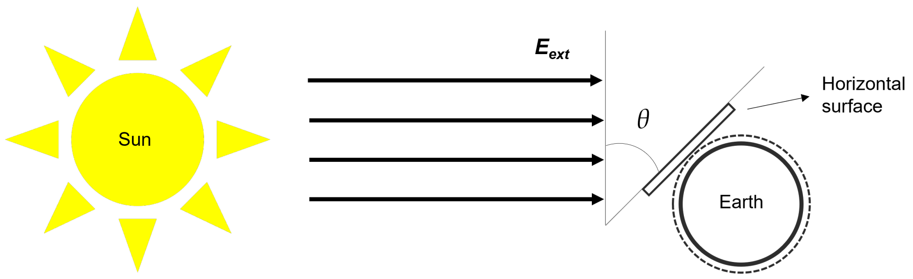


Figure 2.22: Schematic representation of Sun irradiation reaching Earth

When this extraterrestrial radiation reaches the atmosphere, several complex processes occur. The incoming electromagnetic wave is divided into a reflected component, a refracted component, and an absorbed component. Furthermore, the light that makes it through the atmosphere is scattered according to the Rayleigh process. To abstract away some of this complexity, it is common practice to consider that the total irradiation that reaches a horizontal plane (Global Horizontal Irradiance (GHI)) is composed of only two components. The Direct Normal Irradiance (DNI), which describes the solar irradiation that hits the surface directly along its normal axis and needs to be corrected by the Sun zenith angle θ, and the Diffuse Horizontal Irradiance (DHI), which includes all additional irradiation that reaches the surface due to reflection and scattering by particles in the air or clouds.

$$GHI = DHI + DNI \cdot cos(\theta) \tag{2.31}$$

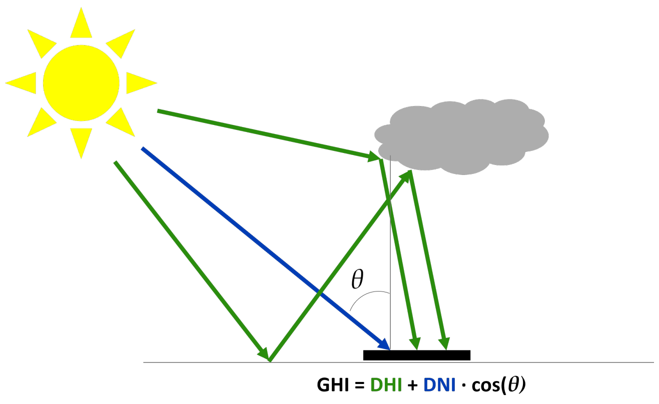


Figure 2.23: Schematic representation of GHI, DHI, and DNI

It is noteworthy that the presence of clouds, if they do not directly obstruct the sun, can actually increase the total irradiance that reaches a solar panel, a phenomenon known as overirradiance [58].

2.3.2. Clear-sky Models

Clear-sky models are analytical models that predict a value for the solar irradiance in a given location, on a certain date and time under the assumption that no clouds are present. In a way, these models provide the theoretical maximum solar irradiance that can be captured at a certain location on a given moment (not strictly true due to the occurrence of overirradiance). There are several methods, some of which also have open source implementations. In this work, we will use one of the most highly regarded ones, developed by Ineichen and Perez [59], freely available through the pvlib Python library [60]. To get an in-depth understanding of this method, the reader is referred to the original source, but it essentially works as follows. First, we input the location (latitude, longitude, altitude) as well as the precise time (year, month, day, hour, minute, second), which allows the model to compute the position of the Sun relative to the point of interest using optical formulae. The model can then roughly predict the amount of solar irradiation that reaches the point of interest. This method, goes one step further and adjusts the results according to location-dependant turbidity profiles. To assert that this model yields accurate results for our dataset we compare it to the measured irradiance on several clear days (e.g. Fig. 2.24). The correlation factor was always in the range 0.9-0.9999, hence we conclude that the chosen clear-sky model can closely describe our observations.

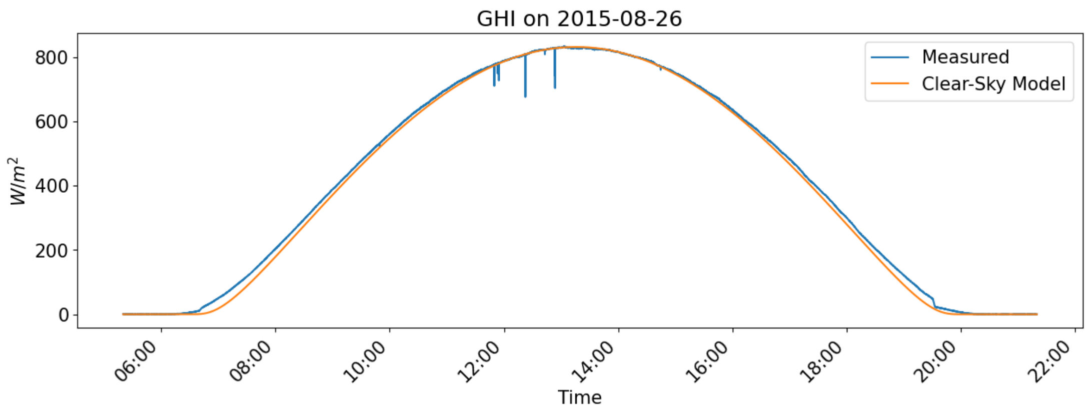


Figure 2.24: Comparison between measured GHI and clear-sky model GHI

The Clear-Sky Index (CSI) is a dimensionless ratio that measures how much the solar irradiation differs from what is expected under clear sky conditions. It is calculated by dividing the actual observed solar irradiance by the theoretical clear sky irradiance.

$$CSI = \frac{GHI_{\text{observed}}}{GHI_{\text{clear-sky model}}} \tag{2.32}$$

The CSI is particularly useful in the solar-energy and meteorological contexts since it provides an intuitive single-number metric that informs about the level of cloud cover and the turbidity of the air. A CSI value of 1 indicates that there is no atmospheric interference like clouds or haze. Values less than 1 indicate the presence of cloud cover or other atmospheric factors reducing the solar radiation reaching the surface. The presence of ground reflection or overirradiance can yield GHI values greater than 1.

2.4. Related Work

In recent years, data-driven methods have become increasingly popular in weather forecasting, covering time horizons from a few minutes to several days [61]. This section delves into published applications of ML to the tasks of precipitation, cloud cover, and irradiance forecasting, which closely match the objective of this project.
One of the pioneering works in the field of data-driven weather prediction was published in 2015 by Shi et al., and introduced the Convolutional Long Short-Term Memory (ConvLSTM) [62]. The ConvLSTM extends the Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN), to incorporate convolutions. This design allowed it to capture spatiotemporal correlations effectively, addressing the limitations of fully connected LSTM in handling spatial data. At the time, this architecture obtained state-of-the-art results in precipitation nowcasting.
Contemporaneously, Klein et. al [63] introduced the ”Dynamic Convolutional Layer”. In contrast to conventional convolutional layers, which use fixed filters learned during training, dynamic convolutional layers use filters that vary from input to input during inference. This is achieved by learning a function that maps the input to a certain convolutional kernel. The authors successfully applied this new architecture to short-term weather prediction, showing improvements compared to several baselines.
Researchers from Google have been very prolific in the field with the development of the MetNet models. MetNet-1 [64], is designed for short-term precipitation forecasting and produces precipitation maps at a high spatial and temporal resolution. It employs a novel architecture that builds upon ConvLSTM and incorporates axial self-attention. MetNet-1 demonstrated superior performance compared to traditional NWP models for forecasts up to 7-8 hours ahead. MetNet-2 [65], from 2021, builds on its predecessor but makes extensive use of residual blocks and employs novel NN elements like exponentially dilated convolutions, which double their receptive field after each layer. MetNet-2 can generate forecasts up to 12 hours ahead with greater skill than current state-of-the-art physics-based models like HRRR and HREF. The most recent addition to the MetNet family, MetNet-3 [66], extends the capabilities of the previous models up to 24 hours ahead. A key innovation in MetNet-3 is its densification technique, which allows for spatially dense forecasts despite being trained on sparse targets, effectively capturing data assimilation processes within its architecture. MetNet-3 outperforms the best single and ensemble NWPs providing forecasts with spatial and temporal resolution of up to 2 minutes and 1 km.
Another notable mention is the work of Le Guen and Thorne [67], who proposed the PhyDNet, a model designed for unsupervised video prediction. PhyDNet introduces a two-branch architecture that disentangles physical dynamics from unknown residual features, like appearance, details and texture. To learn the physical dynamics, the model introduces a new recurrent physical cell, the PhyCell, which performs PDE-constrained predictions, while the unknown factors are learnt by a ConvLSTM, both of them in a common latent space mapped by an autoencoder. In a subsequent work [68], Le Guen et al. expanded their model to learn separate latent spaces for the physical and residual branches by incorporating specific encoders-decoders for each branch, further enhancing the model’s ability to disentangle these dynamics. They apply the updated model to the task of irradiance forecasting using fisheye images and achieve state-of-the-art results in video prediction and 5-minute-ahead irradiance forecasting, highlighting the benefits of incorporating physical dynamics into ML models for real-world applications.
More recently, generative models have been widely adopted for weather forecasting. One of the first successful implementations emerged in 2021 from a collaboration between DeepMind and the UK Met

office [69]. They introduced Deep Generative Model of Rainfall (DGMR), which utilizes Generative Adversarial Network (GAN) to predict future radar fields based on past observations. DGMR is capable of producing high-resolution forecasts up to 90 minutes ahead. The model addresses common issues such as blurry predictions by using spatial and temporal discriminators, ensuring realistic and consistent outputs. Evaluations, including expert meteorologist assessments, demonstrate DGMR's superior accuracy and usefulness over traditional methods and other deep learning models.
The great success of attention in natural language processing sparked interest among researchers to try and apply similar techniques to computer vision, namely ViTs. There are several weather forecasting models that employ ViTs, the most notable ones are FourcastNet [70] and Pangu-Weather [71], [72]. Developed by NVIDIA, FourcastNet combines ViTs with Adaptive Fourier Neural Operator (AFNO) [73], which the authors claim enables efficient spatial token mixing, achieving high-fidelity modeling of fine-grained atmospheric phenomena. The model matches the accuracy of the European Centre for Medium-range Weather Forecasts (ECMWF) Integrated Forecasting System (IFS), while generating forecasts significantly faster (within seconds for weeklong predictions). This speed facilitates the creation of large ensemble forecasts. FourcastNet also excels at predicting extreme weather events, such as tropical cyclones and atmospheric rivers. Huawei's Pangu-Weather, on the other hand, employs an Earth-specific 3D transformer, an architecture inspired by a ViT expanded with positional biases designed to align with Earth's geometry. Additionally, to reduce computational cost, the model uses a window-attention mechanism, which partitions the feature maps into windows. Pangu-Weather is about 50% slower than FourCastNet, but is more accurate on several benchmarks.
Graph Neural Networks (GNNs) can't be considered generative models, but they have also been successfully applied to weather forecasting. For example, [74] is one of the first implementations of message-passing GNNs to predict six-hourly atmospheric states and offers performance comparable to traditional physics-based models. Building on this foundation, researchers from Deepmind created Graphcast [75], a model which uses simultaneous multi-mesh message passing, i.e. message passing along graphs of varying resolution, and an encoder-decoder architecture. This model generates accurate 10-day forecasts that outperform traditional NWP.
Finally, DMs are among the most successful methods for producing realistic and accurate weather forecasts. One of the key advantages of DMs is their ability to produce probabilistic outputs instead of point estimates. That means that instead of a single realization, DMs can provide a range of possible outcomes along with their associated probabilities, helping to understand the uncertainty and risks associated with the forecast.
Recently, there has been an intense acceleration in the number of papers published that present some diffusion-based architecture for forecasting weather variables. For example, [76] presents a fairly straightforward implementation of diffusion for irradiance forecasting. They combine two DMs in pixel space; the first model provides the prediction of cloud cover, while the second is employed as a super-resolution model; it takes the coarse output of the first model and creates a high-definition forecast. It is not uncommon to find several DMs working in series, which is referred to as cascading, and helps reduce the challenge of training a single DM for high resolution outputs [77]. A similar work was published by Asperti et al. [78], this time with the aim of nowcasting precipitation. In this work, the authors conditioned a DM using previous observations of rainfall, a land-sea mask, the geopotential, and wind speed 100 m above sea level, all stacked along the channel axis. This model generates 15 possible future outcomes that are then fed into a post-processing U-Net that combines them to produce the single most probable outcome. The authors found that this approach significantly outperformed contemporaneous deep learning models.
There also exist more complex models that use diffusion somewhere in their architecture, but also leverage other techniques, like attention, Fourier neural operators, or physics-informed losses, to name a few. One such example is LDCast, which was published in April 2023 by the Federal Office of Meteorology and Climatology (MeteoSwiss) [79]. The model contains three components: a forecaster stack, a denoiser stack, and a variational autoencoder. The central block is the denoiser, which has an architecture similar to typical U-Nets but is enhanced with AFNO blocks. The variational autoencoder is used to downsample the precipitation maps in order to perform the diffusion process in a latent space, reducing the computational complexity of training and inferencing the model. Finally, the forecaster, which is essentially a temporal transformer augmented with AFNO blocks, is used to condition the DM using cross-attention. LDCast not only outperforms state-of-the-art models such as DGMR, but also accurately quantifies the uncertainty of its predictions by leveraging the ability of DMs to generate di-

verse samples.
In December 2023, two remarkable models were made public: Diffcast [80] and PreDiff [81]. DiffCast uses a deterministic component to capture the global trend of precipitation and a stochastic DM to account for the residuals (finer details). The backbone of the diffusion is a U-Net enhanced with temporal attention [82] conditioned with a motion prior. The authors design the deterministic component so that it can employ any spatio-temporal predictor (ConvGRU, Transformer, GAN, PhyDnet, etc.) with minimal adaptations. Diffcast achieved state-of-the-art results by being able to capture both the large-scale structure and the fine-grained details of precipitation events.
Prediff is a model by AWS which employs a two-stage pipeline: First, it uses latent diffusion to generate probabilistic forecasts; second, it aligns the forecasts with domain-specific physical constraints through a knowledge alignment network. This approach ensures that the predictions generated are not only likely in the probabilistic sense but also physically plausible. The authors make extensive use of an architecture that they developed in a previous work: the Earthformer [83]. The Earthformer is a Transformer model designed for Earth system forecasting, which uses a novel cuboid attention mechanism to capture spatiotemporal dependencies. Prediff employs an Earthformer U-Net as the backbone for the diffusion process and an Earthformer encoder as the knowledge alignment network. Additionally, the model uses an adversarially trained VAE to encode the input into a latent representation and to convert the latent predictions back to precipitation maps. Prediff not only generates forecasts with high operational utility, but also shows the effectiveness of incorporating physical knowledge into data-driven weather forecasting models.

3
Data

3.1. Data Acquisition

The approach used to obtain the dataset consisted in a setup of fixed fisheye cameras at two solar plants, coupled with irradiance sensors and tracking of power output. The two locations: Cavriglia, in the Tuscany region (Fig. 3.2) and Montsoleil, in the Swiss Jura mountains (Fig. 3.1), are quite different in terms of terrain an weather conditions, providing a wide set of conditions necessary for the model to generalize.

Figure 3.1: MontSoleil PV plant.

The two fisheye cameras are programmed to take 4 images of the sky over the PV plant every 6-8 seconds. Each of the 4 images is captured at a different level of exposure (200, 400, 700, 1100 ms), which are then combined using the Debevec algorithm [84] to create an High Dynamic Range (HDR) image (see Fig. 3.5 and 3.6). In total, the dataset comprises around 4 million images captured between July 2015 and May 2016.

The source of ground truth are irradiance measurements obtained with sensors (Fig. 3.3) located besides the solar panels, with a sampling frequency similar to that of the images. There is additional data available which has not been used, for instance temperature measurements, active and reactive power values, or equipment status flags.

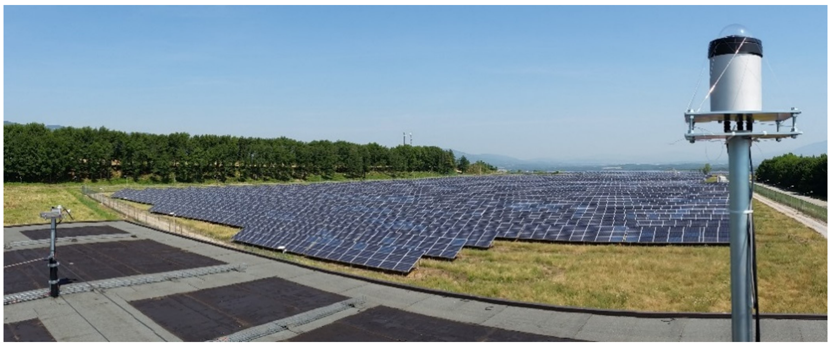


Figure 3.2: Cavriglia PV plant (fisheye camera in the right).

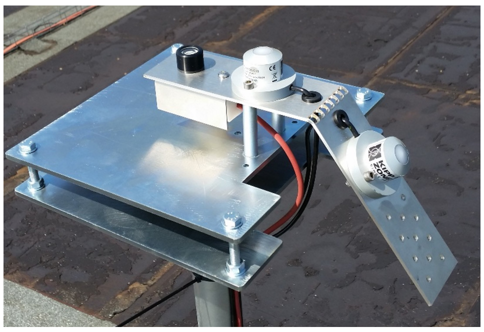


Figure 3.3: Irradiance Sensor.

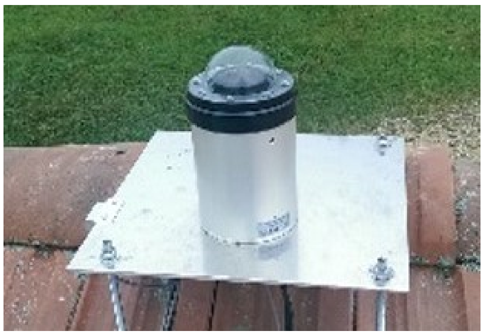


Figure 3.4: Fisheye camera.

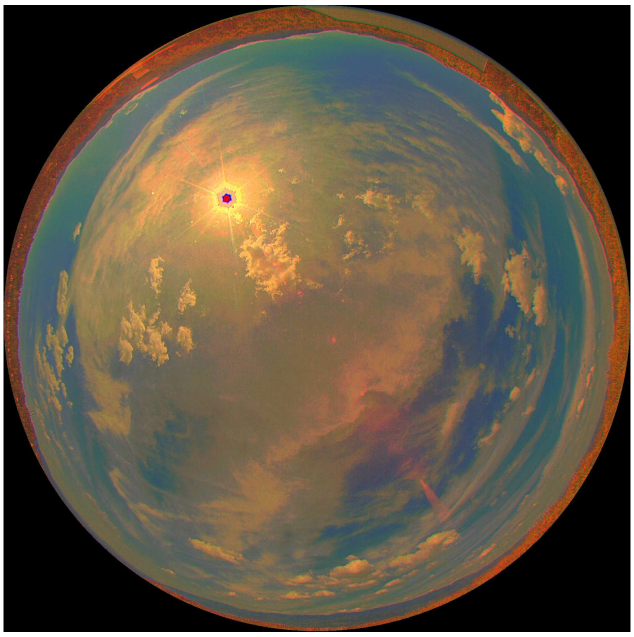
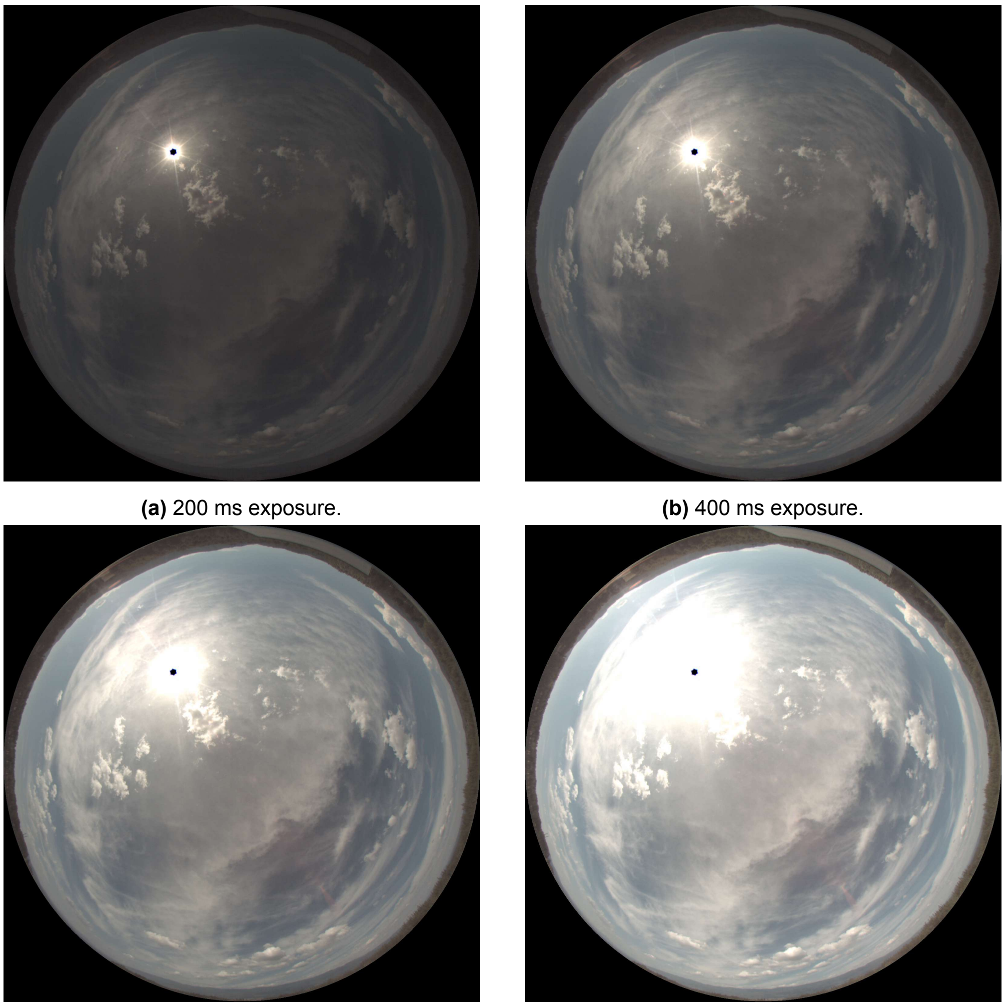


Figure 3.5: Sample image from the Cavriglia site.



(a) 200 ms exposure. **(b)** 400 ms exposure.

(c) 700ms exposure. **(d)** 1100 ms exposure.

Figure 3.6: Different levels of exposure (same sample as in 3.5).

3.2. Exploratory Data Analysis

In any machine learning project, the initial steps in understanding and interpreting the data are crucial. Especially when dealing with real world data, which are often plagued by noise, outliers, missing values, and other inconsistencies. This section contains an initial screening and analysis of the images and irradiance logs, which will be used to get an idea of the quality of the available data and determine the necessary preprocessing steps.

3.2.1. Images

As mentioned in Section 3.1, the image dataset consists of about 4 million images, from two different sites, with a resolution of 1600 x 1600 pixels. Upon visual inspection of the dataset, most of the images show a high-quality picture of the sky. However, dusk and dawn images are often corrupted by color tints and the brightness is too low. It can also be seen that if the Sun is not covered, the pixels close to it are saturated, appearing as a black or purple hexagon or hexagonal star (this shape is caused by the arrangement of atoms of the lens material), as can be seen in Fig. 3.7. This is concerning, since pixels close to the Sun are the most important ones for short-term PV predictions, so crucial information is being lost.

The dataset has several other undesired features. First, some of the images from the dataset show color artifacts, likely originating from the HDR algorithm, as seen in Fig. 3.8. Furthermore, the brightness level is often not consistent, even for images taken a few seconds apart, as shown in Fig. 3.9. Furthermore, several images are corrupted by color tints, especially during dusk and dawn, like the ones in Fig. 3.10. Finally, given the outdoor installation of the camera, it is expected to encounter various environmental factors. These include: interactions with birds (Fig. 3.11), insects on the lens (Fig. 3.12), water from precipitation and ice formation (Fig. 3.13), among others.

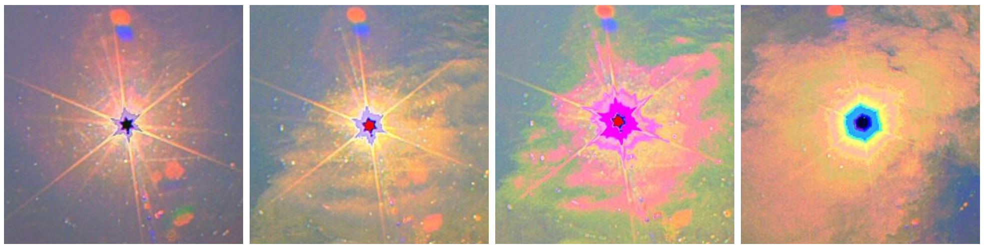


Figure 3.7: Close-up of the sun.

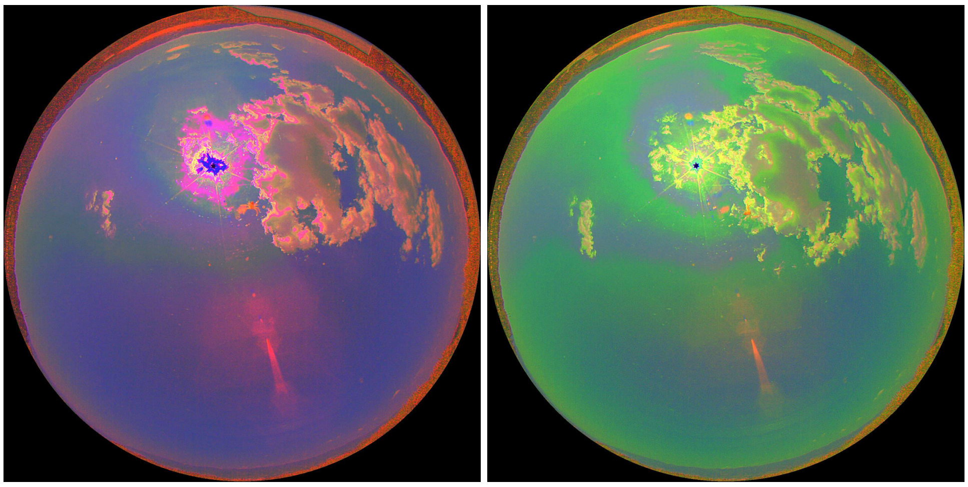


Figure 3.8: Images with color artifacts.

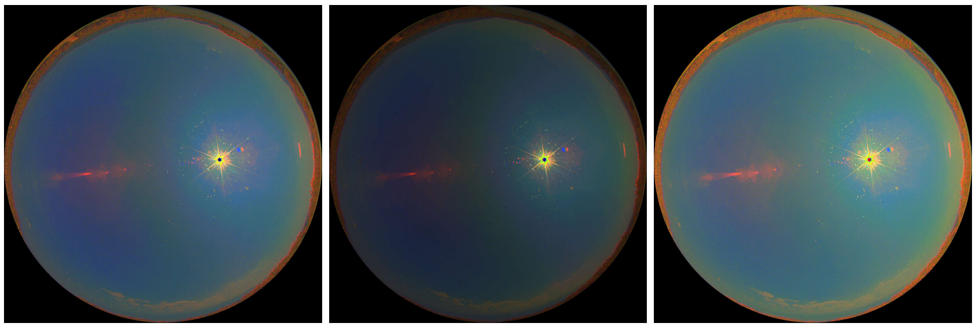


Figure 3.9: Successive images with inconsistent levels of brightness.

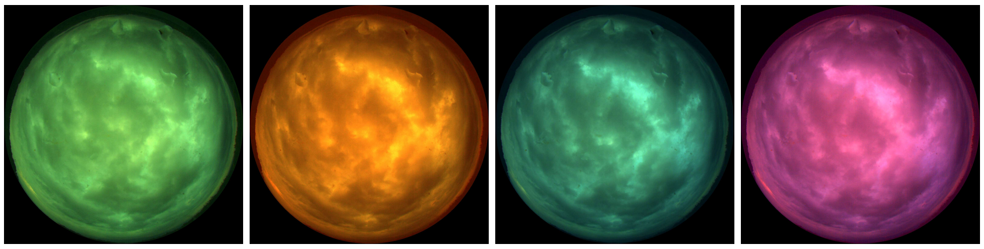


Figure 3.10: Images with bright color tints.

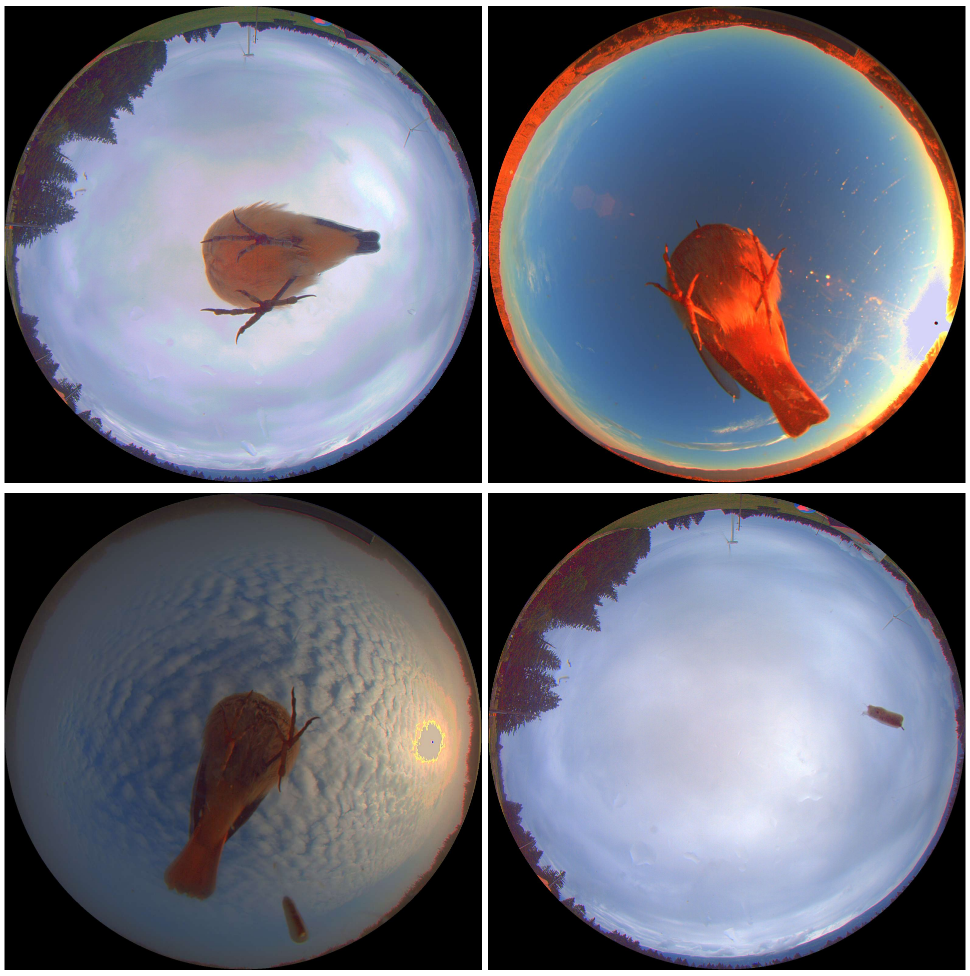


Figure 3.11: Images with birds and bird excrement.

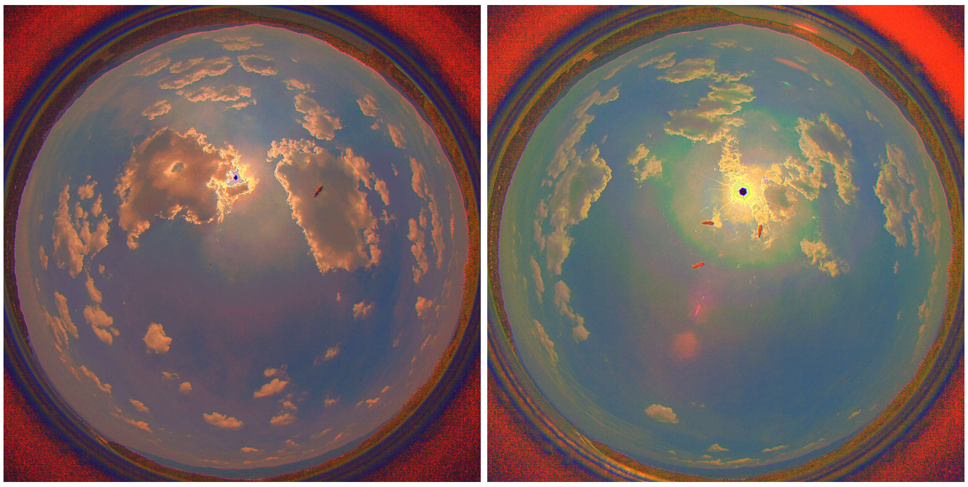


Figure 3.12: Images with insects and sun reflection.

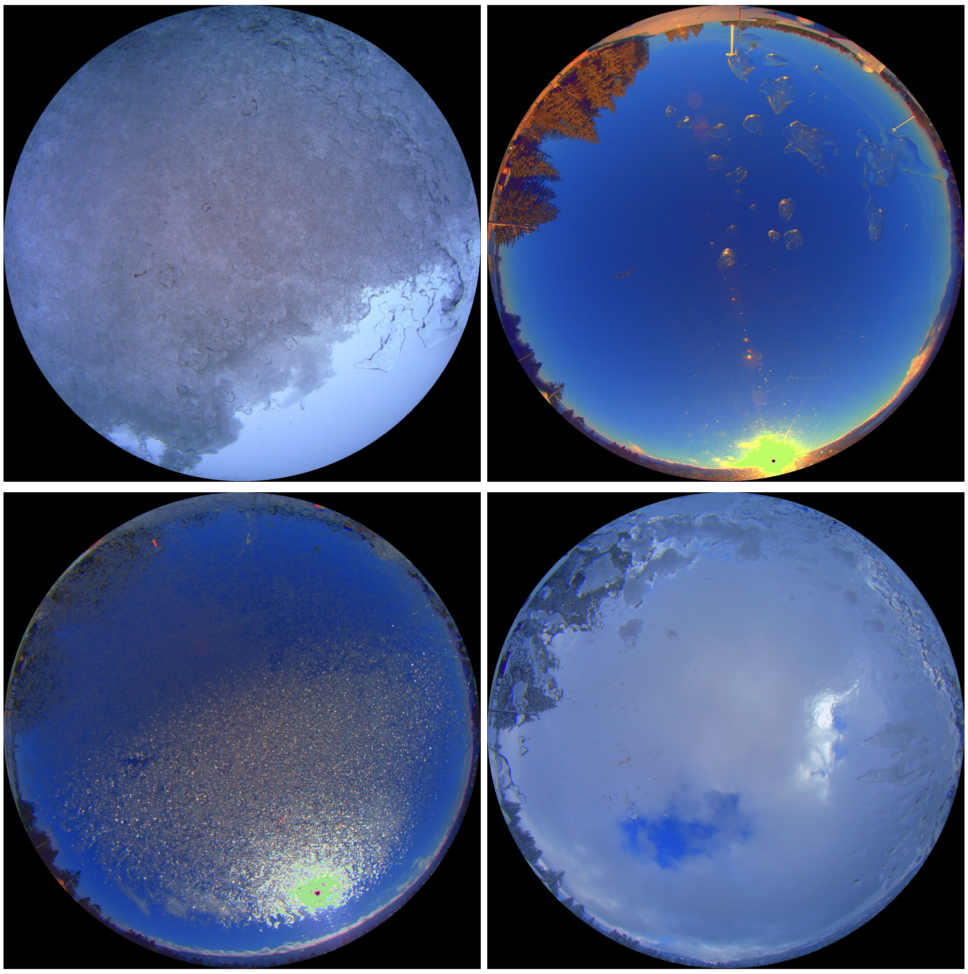
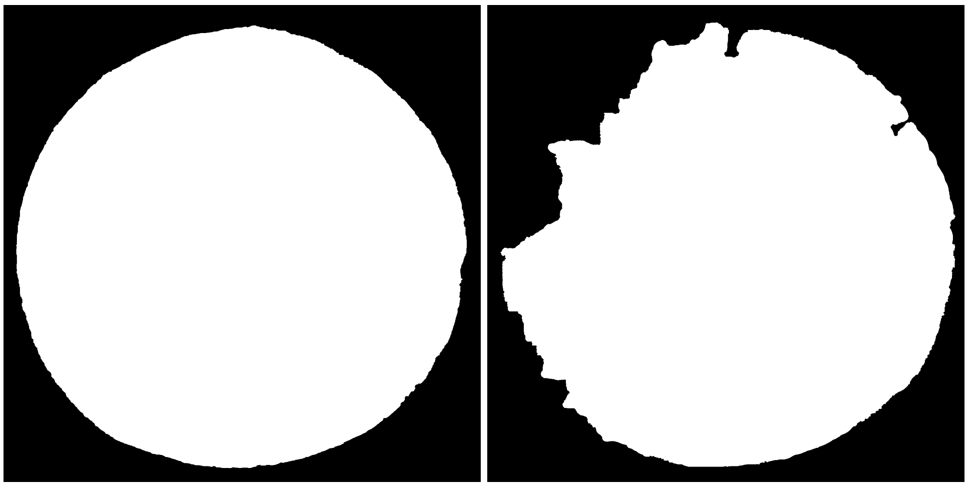


Figure 3.13: Images with snow, water, and ice on the lens.

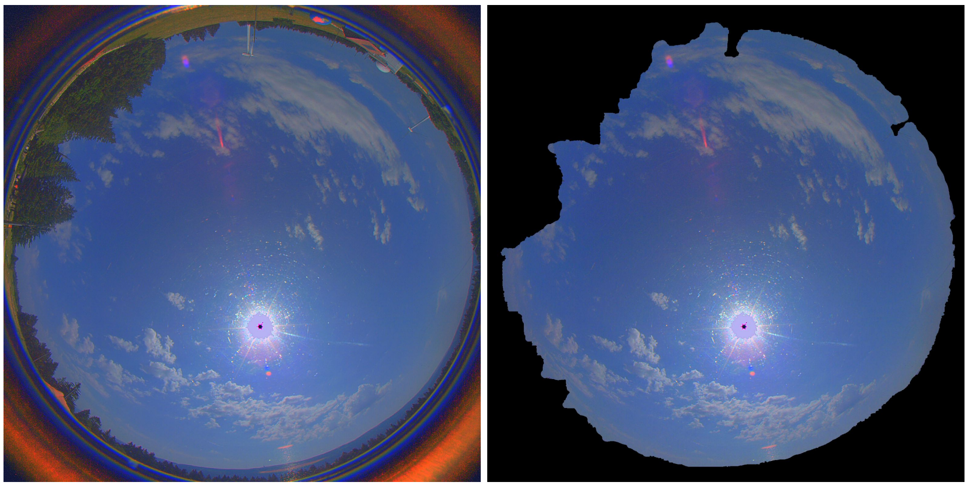
Despite all of the above, it has been considered that the fraction of images with these issues is quite small ($< 1\%$), so the impact on the model performance should be contained, and since there is no straightforward automatic way to filter them out they will remain in the dataset.
With regard to the pre-processing steps for the image dataset, there are several aspects that need to be considered. First, images contain a lot of unnecessary information that is not related to the

task of predicting cloud movements. We are referring in particular to all the trees, mountains, and wind turbines that can be seen on the rim of the images. Secondly, images are too high resolution to process efficiently given the available hardware, a literature search shows that an image of 128^2 or even 64^2 pixels should suffice for the task at hand [85]. This downscaling can be easily accomplished with off-the-shelf bilinear or bicubic resizers available in virtually all machine learning frameworks [86]. To address the first issue, we create a binary mask to be superposed on the image so that only relevant information is visible to the model. Of course, the masks are location-specific, as the landscape around the camera is different for both sites (see Fig. 3.14 and Fig. 3.15).



(a) Cavriglia **(b)** MontSoleil

Figure 3.14: Masks



(a) Image before applying mask **(b)** Masked image

Figure 3.15: Image masking example

Finally, it is common to normalize or standardize images before they are fed to a neural network. This ensures that the input is consistent and has been shown to improve model performance, as well as training stability and convergence [87]. In particular, image normalization is the process of scaling pixel values so that they are in a particular range. This involves linearly scaling the pixels, which are usually in the range [0, 255], to lie in the range [0, 1] or [-1, 1] (the latter more common in diffusion). This can be achieved with Eq. 3.1.

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \cdot (newMax - newMin) + newMin \tag{3.1}$$

Where x is the pixel value, x_{min} and x_{max} are the minimum and maximum values in the image, and $newMin$ and $newMax$ are the limits of the desired range.
Standardization, on the other hand, consists in shifting pixel values so that they have zero mean and unit variance; however, this is usually not found in diffusion.

3.2.2. Irradiance

The irradiance data come in a set of daily logs that contain the GHI measurement with a sampling frequency of around 6 seconds. The setup was configured to only function during hours with daylight; hence the daily start and end time of the measurements varies depending on the season. The irradiance data are fairly clean and high-quality, with only some common minor issues. There are a few times when the setup stopped functioning and values are missing for a few hours. In addition, a small offset was observed and corrected.
Along with the irradiance measurements, we include the clear sky GHI obtained using the pvlib Python library, as introduced in Section 2.3.2. This allows us to calculate the CSI, which gives an indication of the amount of cloud cover at any given moment. For example, in Fig. 3.16 we can see two distinct sky conditions on the same day. From 10:00 to around 13:30 we observe what we will call "Variable" conditions. In this situation, relatively small clouds are constantly moving and covering and uncovering the Sun, which results in steep drops and jumps in the CSI. Additionally, we can also identify the presence of overirradiance with the CSI surpassing 1 at various moments. On the other hand, from 13:30 until the end of the day we can see that the CSI remains fairly stable hovering around 1. This indicates that there are no clouds present and the sky is clear; it is "Sunny". If the CSI was also relatively stable but at a lower value, we would be in an "Overcast" situation, in which thick clouds cover the whole sky and the Sun is not seen for several hours.

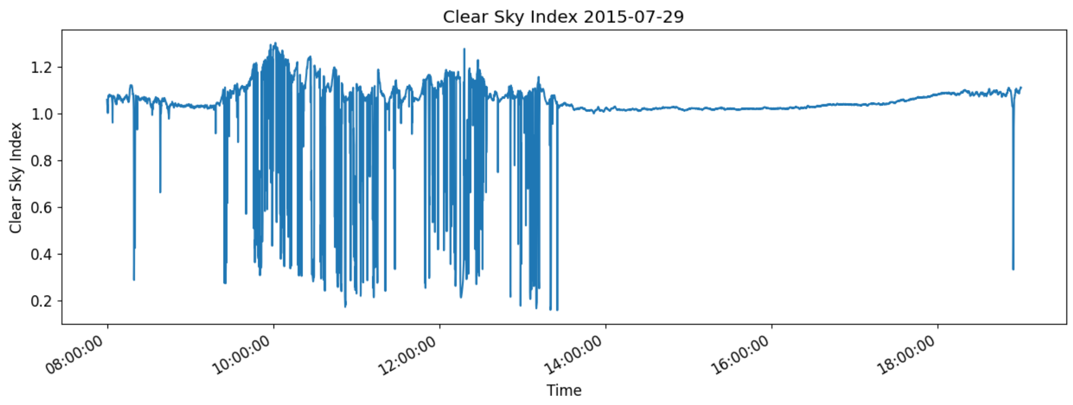
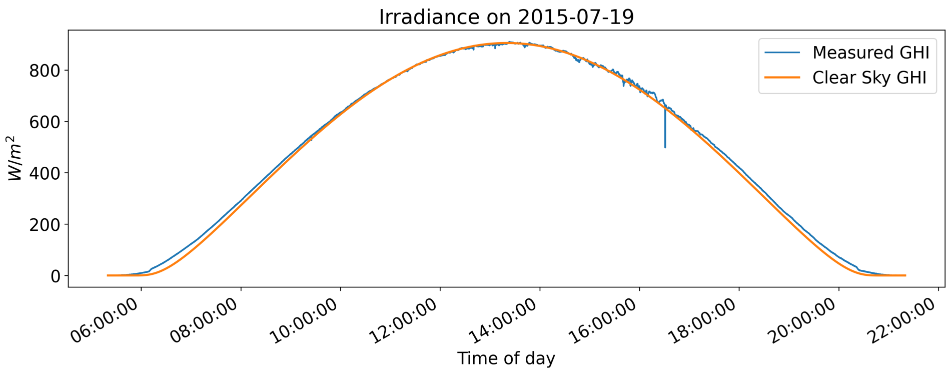


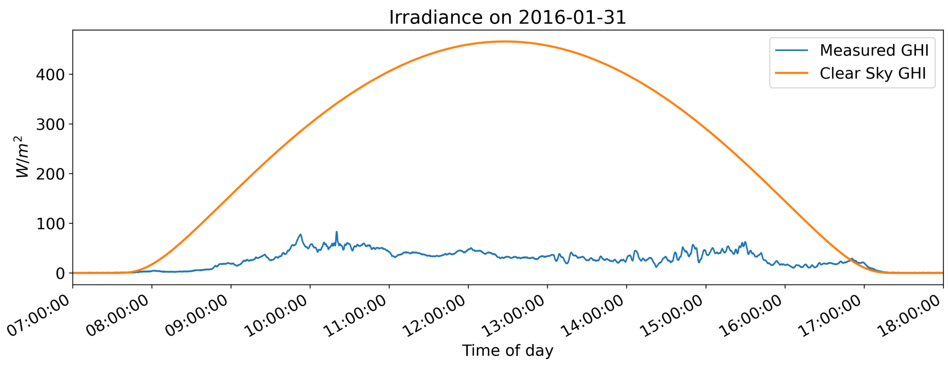
Figure 3.16: CSI

We can leverage the value and variability of the CSI to identify and label days or hours according to the sky conditions. This is very useful to filter and navigate the vast dataset more easily and to get an idea of what sky conditions are rare and which are more prevalent. There are several ways to approach this problem, for instance, we could set a certain threshold (e.g. $CSI = 0.7$) and consider values above it to correspond to sunny conditions and values below it to overcast conditions. Furthermore, if during a certain period the number of crossings of the threshold is very large, we would classify that as a period of variable conditions. This approach has been tried, but the results were not conclusive enough and several misclassifications were present. To address this, we instead consider the moving average of the absolute value of the derivative of the irradiance. Intuitively, the derivative indicates how much the value of a quantity is changing at a certain point, hence steep jumps (drops) in the irradiance will produce large positive (negative) values in its derivative. Since we are interested in changes in both directions (drops and jumps) we take the absolute value of the derivative and apply a moving average to reduce the volatility of the metric. Whenever this value is above a threshold we consider we have variable sky conditions, which as stated in the project motivation, are the main focus of our research. The threshold is set at 10% per minute, which is the maximum acceptable ramp rate for a grid-connected plant.

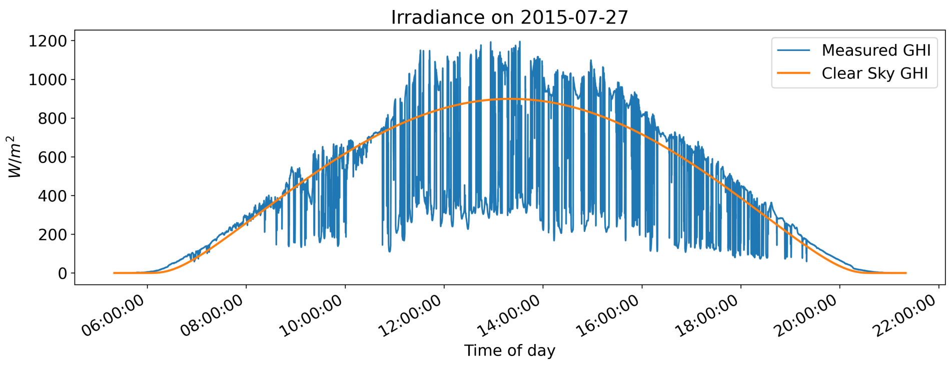
Now we have a method that allows us to classify any period of time in one of the three categories: ”sunny”, ”overcast”, or ”variable”, as can be seen in Fig. 3.17. This will be useful to assemble datasets that contain the desired amount of each sky conditions. As mentioned in Section 2.1.3, persistence often yields very high accuracy because most of the time irradiance does not change abruptly. Given that our focus is not on raw accuracy, but rather in accurately predicting sharp changes in irradiance, we can artificially increase the share of variable conditions in the train and test datasets.



(a) Sunny day.



(b) Overcast day.



(c) Variable day.

Figure 3.17: Selected samples of the irradiance profile for different conditions.

4 Method

4.1. Overview of Proposed Approach

As mentioned in the Introduction, the main contribution of this work is the proposed two-step GHI prediction pipeline consisting of a diffusion-based next-frame predictor followed by an image processing NN. Fig. 4.1 shows an overview of the model. It is noteworthy that the two submodels are separable and do not necessarily depend on each other, they can be trained and used independently.

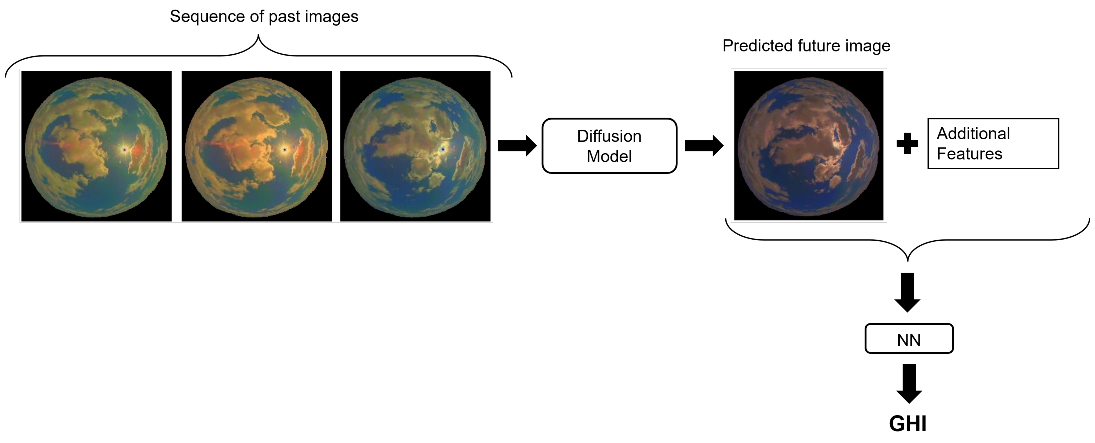


Figure 4.1: High-level schematic of model

The first part, consists of a custom diffusion pipeline, which takes as conditioning information a sequence of past images. The backbone is a U-Net enhanced with self-attention in the deep layers and is trained according to Algorithm 1 in Section 2.2.4. The implementation leverages the open source Python ML library Pytorch [88], as well as the diffusion-specific HuggingFace toolbox Diffusers [89]. Inference of the model (denoising loop) follows Algorithm 2.
Regarding the second part of the model, we are dealing with a multimodal regression task. Multimodal learning is a type of deep learning that uses a combination of various modalities of data (e.g. images, text, audio...). In our case, we are working with images, datetimes and numerical values. The main input feature is the future image of the sky, which we concatenated along the channel axis with a binary mask indicating the position of the Sun (see Section 4.2). Additionally, we include information about the date and time using geometric encodings (see Section 4.2), and past irradiance observations.

4.2. Additional Features

In addition to the images and the previous irradiance, some additional features were thought to be potentially beneficial to the performance of the model. Namely, we will create two additional features: the Sun position in pixel coordinates and a geometric encoding of the time of the year.
The first task involves mapping the location of the Sun from the outside world to a location in a 2D image. This task is remarkably challenging, especially given the pronounced distortion present in images taken with a fisheye lens. However, some existing software can be leveraged to significantly reduce the burden on the project timeline and the necessary technical expertise.
In essence, the process involves computing the sun position (zenith and azimuth angle at the observer location) as a function of the observer's position on Earth and the local time. This is achieved with an off-the-shelf implementation of the algorithm presented in [90]. Then, the spherical coordinates are transformed to Cartesian, and a precomputed mapping is applied to transform the coordinates in the outside world to a pixel position in the image. In order to guarantee a correct mapping, calibration is essential.
The calibration consists of two steps. First, the internal calibration contains information on which coordinates in the world (relative to the location and orientation of the camera) correspond to which pixel in an image. Then, the external calibration captures the information about how the camera is located and oriented in the world. Internal calibration was performed upon installation of the camera using the OCamCalib toolbox [91], [92]. External calibration is done using a sequence of images from a sunny day, in which the sun can be easily detected, and using Kabsch's algorithm [93].
This process allows us to obtain the position of the Sun in an image given the time when the image was taken and the location (Cavriglia or MontSoleil), as can be seen in Fig. 4.2.

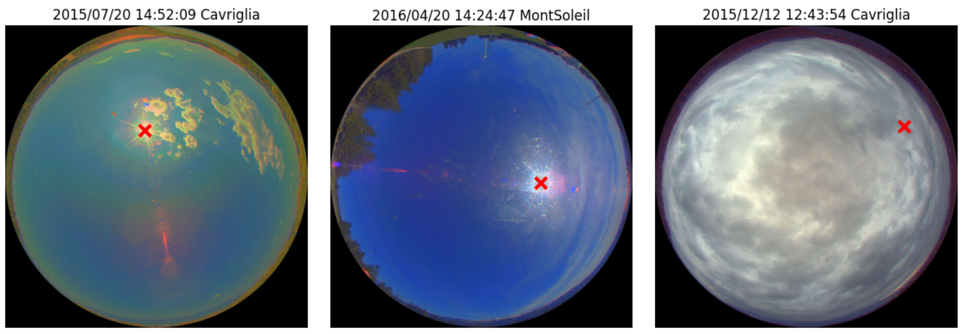

2015/07/20 14:52:09 Cavriglia
2016/04/20 14:24:47 MontSoleil
2015/12/12 12:43:54 Cavriglia

Figure 4.2: Random samples of Sun location process. Sun position indicated with a red cross.

In order to provide machine learning models with temporal information, it is common to use some kind of encoding. One of the simplest and most common ways employs basic geometric functions such as sine and cosine. This kind of encoding captures the cyclical nature of temporal information, which numerical encoding or one-hot encoding cannot represent. For instance, while days of the week range from 1 to 7, this simple numerical representation does not convey that Sunday and Monday are adjacent, but on opposite sides of the range, potentially leading to misinterpretations by neural networks.
To address this, sine and cosine encodings are used to map temporal features onto a continuous circular space, preserving the periodic nature. The implementation is straightforward; for example, the encoding of the hour of the day would use Eq. 4.1 and Eq. 4.2 and can be seen in Fig. 4.3.

$$\mathbf{Feature}_{\mathbf{sin\_hour}} = sin\left(\frac{2\pi \cdot hour}{23}\right) \tag{4.1}$$

$$\mathbf{Feature}_{\mathbf{cos\_hour}} = cos\left(\frac{2\pi \cdot hour}{23}\right) \tag{4.2}$$

Where $hour$ corresponds to the hour of the day and ranges from 0 to 23.
In this project, the temporal information that has been selected and encoded is the day of the year (ranging from 0 to 364), and the minute of the day (ranging from 0 to 1439), which yields four separate features (both sin and cos are included).

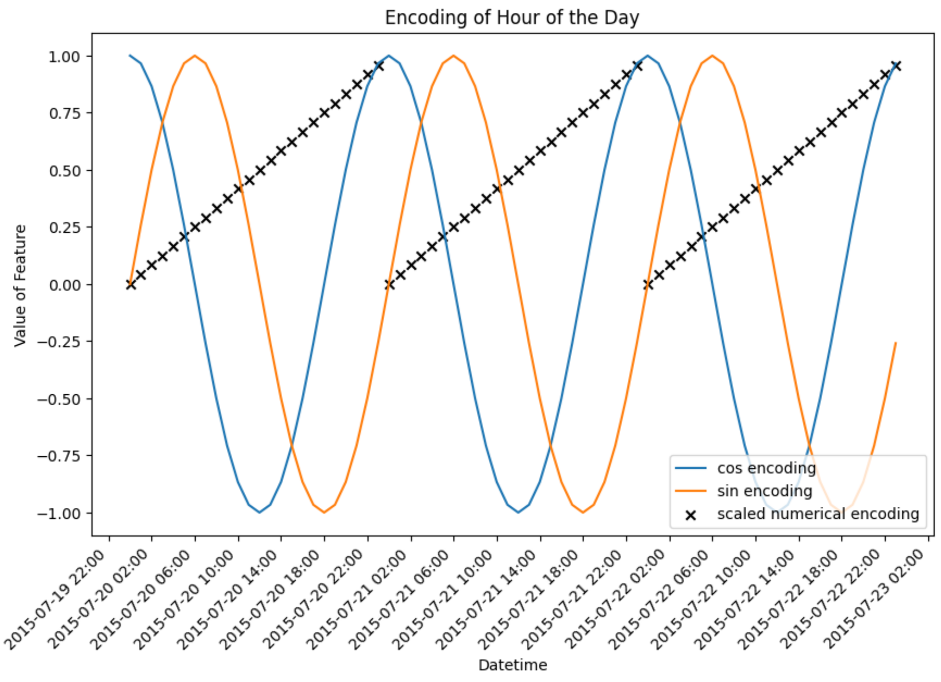


Figure 4.3: Comparison between geometric and numerical encodings.

4.3. Implementation

4.3.1. Next Frame Predictor

As mentioned above, the next frame predictor is trained using algorithm 1. The loop begins by randomly selecting a data sample and a timestep, and sampling Gaussian noise. We then generate a noisy sample that corresponds to the selected timestep in the forward diffusion process (recall Equation 2.25), this noisy sample is the model input. The model's output is compared to the actual added noise using the MSE as the loss function. We perform gradient descent on this loss function for a given number of epochs or until training converges. Inference of the model is slightly different. It follows algorithm 2, shown schematically in Fig. 4.4 and Fig. 4.5, for the pixel and latent space, respectively.

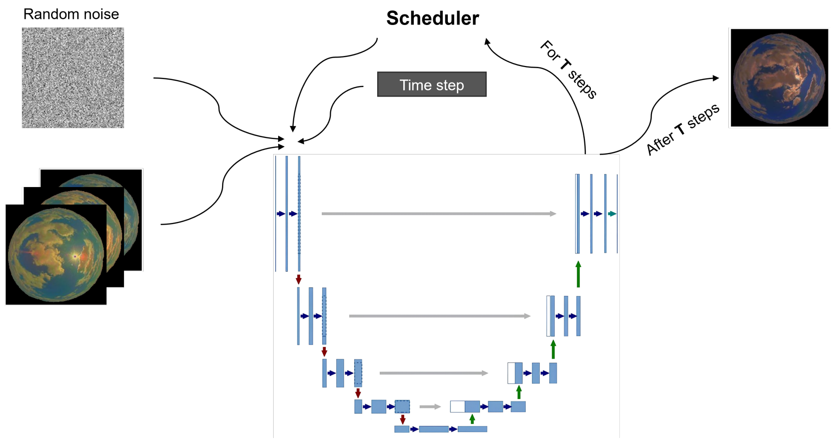


Figure 4.4: DM inference

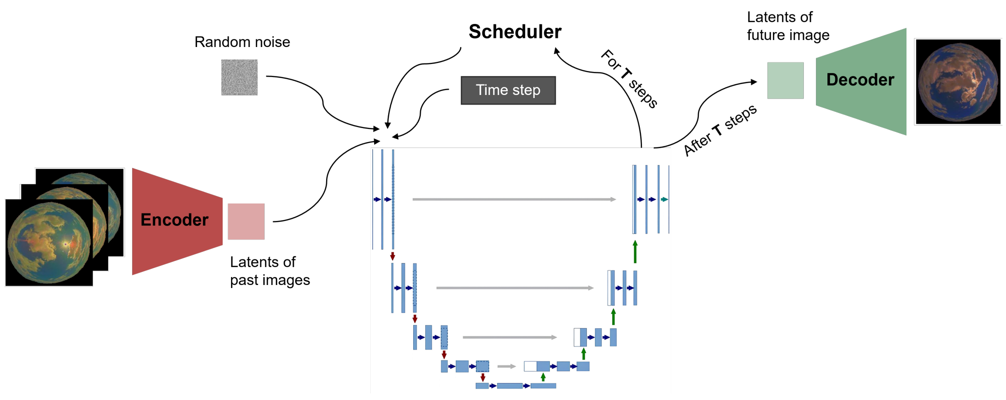


Figure 4.5: DM inference in latent space

The scheduler orchestrates the denoising loop. In the first iteration, random noise is sampled from a standard normal distribution and the conditioning sequence is concatenated along the channel axis. This tensor is passed through the U-Net, which outputs a noise prediction. Intuitively, this noise is what the model ”thinks” we have added to the sample. The scheduler then subtracts this prediction from the noisy sample (pure noise in the first iteration) and adds back the noise corresponding to the next time step (recall that we only take one in every N time steps in the reverse diffusion process). In the last iteration, the U-Net predicts the noise that was added to the now almost-clean image, which is subtracted to produce the final output, this process can be seen schematically in Fig. 4.6. Inference in latent space is analogous, but instead of operating on the pixel values, we use a latent representation (referred to as latents) obtained by means of an autoencoder.

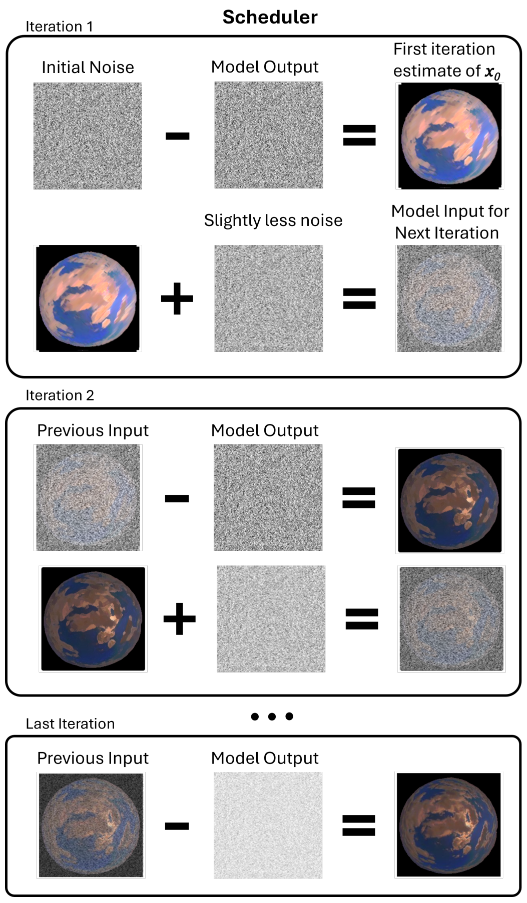


Figure 4.6: Scheduler Iterations

The backbone and main component of virtually all DMs is the U-Net. As introduced in Section 2.2.6, U-Nets basically consist of a contracting and an expanding path, and optionally contains a few middle blocks that do not alter the dimensionality. In its most basic form U-Nets only employ convolutions, max-pooling, and residual connections. However, models that achieve good performance often incorporate attention and residual connections (different from the residual connections that link parallel contracting and expanding blocks). In order to organize this complexity, we define the following building blocks, categorized by whether they reduce the spatial dimension of the input tensor (Contracting blocks), they keep it the same (Same blocks), or increase it (Expanding blocks).

- Contracting blocks
  - DownBlock
  - AttnDownBlock
- Same blocks
  - MidBlock
  - AttnMidBlock
- Expanding blocks
  - UpBlock
  - AttnUpBlock

The parameters of each block, not only depend on which type of block it is, but also where in the network it is located, since this will determine the number of input and output channels as well as the spatial dimensions of the tensor that needs to pass through. Nevertheless, the general structure is relatively similar: a few ResNets, some attention layers (if it's an attention block), and a downsampling or upsampling step as necessary. For instance, a standard contracting block (DownBlock) with two residual networks would have the structure shown in Fig. 4.7 (normalization layers not included for simplicity). It is worth noting that the first residual layer also takes as input information about the diffusion time step.

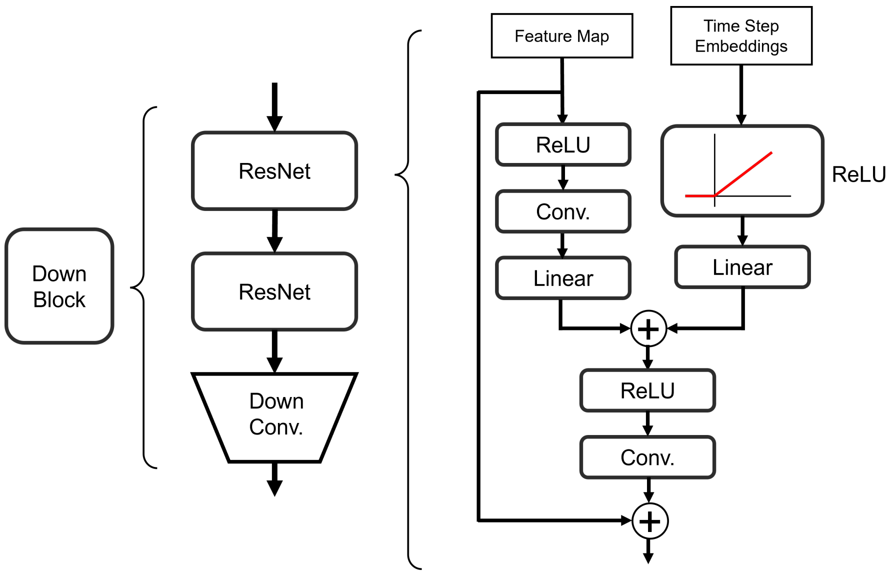


Figure 4.7: DownBlock schematic representation

On the other hand, an expanding block with attention (AttnUpBlock) and also two residual networks would look as follows (Fig. 4.8). Notice that expanding blocks also take as input the residual connection coming from the corresponding contracting block, so this also needs to be accounted for.

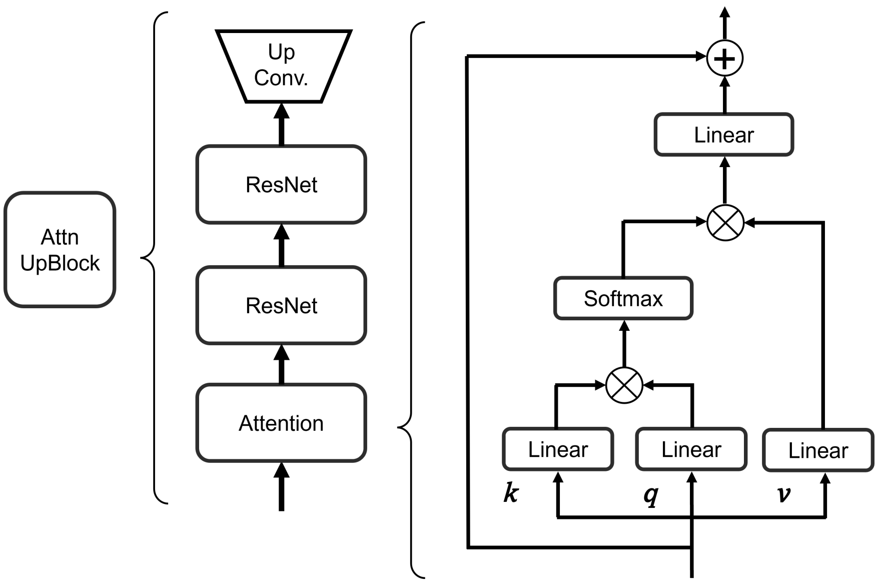


Figure 4.8: AttnUpBlock schematic representation

With these building blocks, we can create U-Nets of varying levels of complexity. In particular, we can decide the number of ResNets per block, the number of blocks in the contracting and expanding path, whether to include attention and the number of output channels of each block. The latter has not been specifically mentioned so far but is also an important hyperparameter, which can dramatically increase the overall parameter count of the network.

4.3.2. Image Processing Network

As introduced in Section 4.1, the second submodel of our pipeline is dealing with a multimodal image regression task that uses both images and supplementary information. There exist numerous architectures and techniques that could be employed, but the most promising and well-known employ a similar structure. First, we pass the image (in our case along with the binary mask) through a sequence of convolutional and pooling layers. This reduces the dimensionality of the tensor while preserving the most important information. Once we have reduced the dimensions to a number that is manageable by fully connected layers, we flatten the cubic tensor into a 1-dimensional vector to which we concatenate the encoded datetime information and the previous irradiance values (after potentially preprocessing them with a Multilayer Perceptron (MLP)). Finally, this vector is passed through a few Fully Connected (FC) layers that culminate in a single artificial neuron that outputs the predicted GHI, as shown in Fig. 4.9.

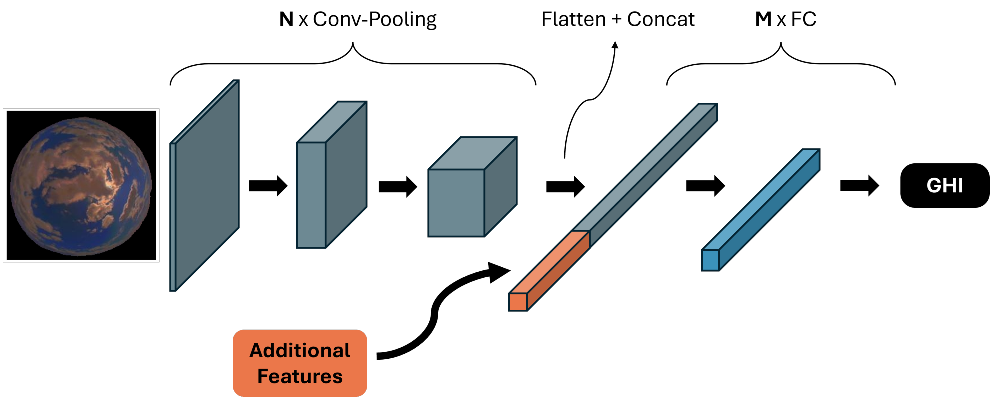


Figure 4.9: Regression CNN schematic representation

We can fully define such a CNN using two lists. The first list indicates the number of out channels in each convolutional-pooling block. For example [8, 16, 32] indicates that the network contains 3 such blocks (Fig. 4.10) with 16, 32, and 64 out channels, respectively. As such, the dimensions of an input tensor with four channels (three color channels plus the binary sun-location mask) with a resolution of 128 would go through the next transformation steps. Recall that the pooling operation reduces the spatial dimensions by a factor of two.

$$[4, 128, 128] \rightarrow [8, 64, 64] \rightarrow [16, 32, 32] \rightarrow [32, 16, 16]$$

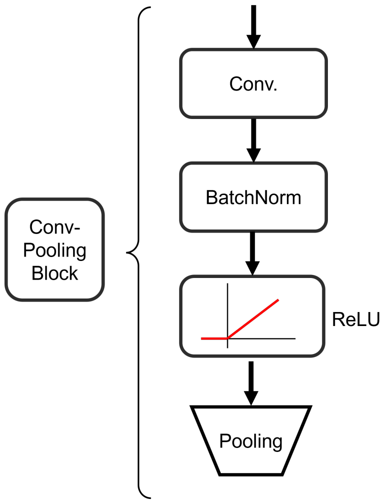


Figure 4.10: Convolutional-pooling block.

The second list, indicates the number of output neurons for each of the fully connected layers. Note that the number of input features on the first FC layer is fully defined by the structure of the convolutional part. In our example, the first FC layer would need to have $32 \cdot 16 \cdot 16 + 5 = 8197$ input features; the flattened dimension of the image tensor plus the five additional features. Similarly, the outermost layer will always only have one neuron, since we are dealing with a regression task.

5 Results

5.1. Next-Frame Predictor

The most important step for accurate short-term predictions of solar irradiance essentially involves forecasting the sky conditions directly above the location of interest, particularly the position and appearance of clouds that might obstruct the Sun. In our proposed two-step pipeline, the next-frame predictor plays a crucial role by generating future sky images based on sequences of past observations. This predicted image is then fed into a CNN, along with additional features to forecast the GHI. In this section we present the development of the diffusion-based next-frame predictor, we motivate the sampling interval as well as the prediction horizon, we determine the optimal architecture, benchmark its performance against relevant methods, and analyze some of its features.

5.1.1. Selection of Sampling Interval and Prediction Horizon

Selecting an appropriate prediction horizon and the time interval between conditioning frames is not trivial, and there likely is no universally correct answer. These choices depend heavily on the nature of the dataset and the specific forecasting goals. In this subsection, we try to determine a reasonable choice given our intended application and the particularities of our dataset.

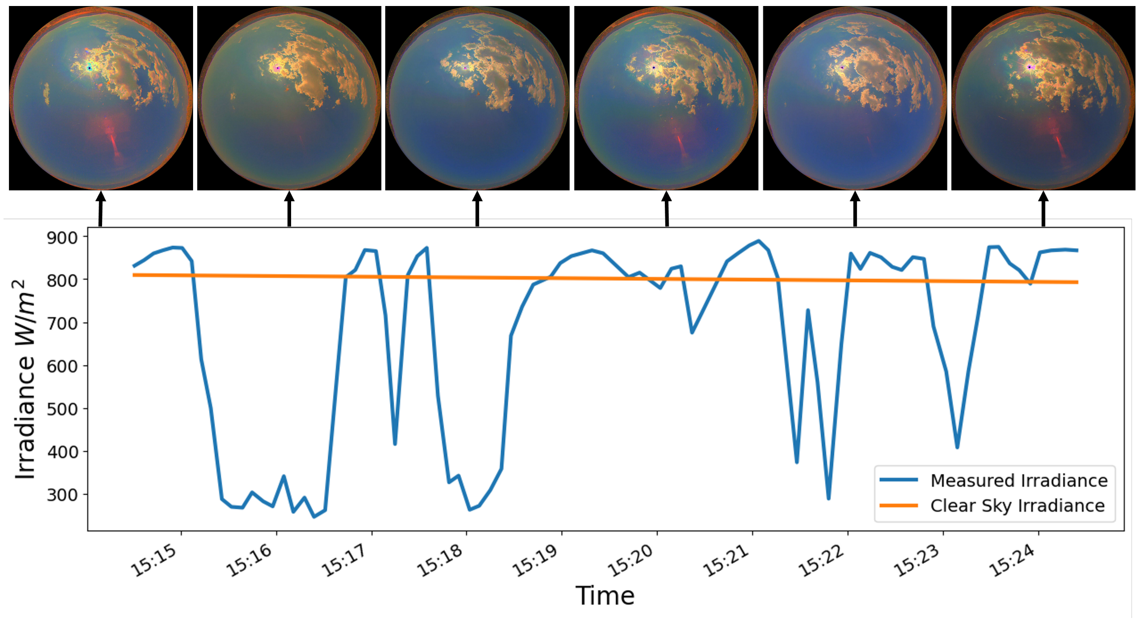


Figure 5.1: Example of irradiance profile with corresponding sky images.

As mentioned in Section 1.1, the main focus of this work lies in very short term prediction. In particular we try to predict incoming fluctuations in irradiance, which pose problems for grid-connected PV plants. Fig 5.1 illustrates the irradiance level over a ten-minute period, along with a subsample of the corresponding sky images. It can be observed that sharp fluctuations occur within the span of a few seconds. In fact, in just this short sequence, irradiance values fluctuate dramatically, swinging between several peaks and valleys. Based on this, we conclude that selecting a prediction horizon beyond one or two minutes is not practical. For instance, if we set the prediction horizon at 5 minutes and try to predict irradiance at 13:20 based on the state at 13:15, we encounter a highly dynamic situation. The irradiance starts near clear-sky levels, drops to around 30%, rises back to a maximum, experiences a small dip, rises again, and then falls once more, eventually returning to its starting level. Even a perfect model, which could predict that the irradiance at 15:20 matches that of 15:15, it would miss the important fluctuations in between, events that can significantly affect power quality, voltage stability and hamper grid code compliance.
In light of the above, we have decide to set the prediction horizon at one minute. There remain a few more choices to fully specify the conditioning sequence: the number of frames it contains and the spacing between them. As for the number of frames, it is evident that at least two are necessary, as no movement can be inferred from a single frame. In order to be able to extract circular trajectories and acceleration at least three frames would be necessary. Although it is unlikely that clouds will exhibit significant acceleration or follow curved paths, the distortion caused by fisheye lenses can create the appearance of clouds moving in circular patterns. To account for this potential distortion, we err on the side of caution and select three frames. Finally, regarding the spacing between frames we adopt a symmetric approach by using the same one-minute spacing. Visual inspection revealed that one minute intervals allow for sufficient differences between frames, ensuring the sequence captures gradual cloud drift, but also maintains coherence and correlation across the sequence.

5.1.2. Pixel Space DM

We start our search for the next-frame predictor by training a pixel-space DM. This model does not rely on a VAE but instead the denoising loop operates on the pixel values themselves. Visual inspection of the generated samples shows promising performance, hence we proceeded to optimize the U-Net architecture to maximize the model's ability to forecast future sky states. In what follows, we will use the blueprint introduced in Section 4.3, where the architecture of a U-Net can be accurately defined by two lists: the block structure of the contracting path and the number of out channels per block (e.g. [DownBlock, DownBlock, AttnDownBlock], [32, 64, 128]). Since the U-Nets are symmetrical, the structure of the expanding path mirrors that of the contracting path, eliminating any ambiguity.
Given the computational burden of training large U-Nets, we must make some compromises with the width and depth of hyperparameter tuning. In particular, we perform a grid search [94] that includes a total of 8 U-Nets with different levels of complexity. The three hyperparameters that we modify are the number of blocks, the type of block (standard or with attention), and the number of channels per block. Note that there is not a unidirectional relationship between the number of blocks and complexity. Including more blocks generally increases the total parameter count of the network, but the dimensionality of the feature map is compressed to a smaller spatial dimension; hence, subsequent operations, such as attention, can operate on a smaller tensor. This can yield a considerable speedup, given the quadratic computational complexity of the attention mechanism. Furthermore, the number of parameters on each block is proportional to the number of output channels; nevertheless, the number of channels does not increase the computational complexity of attention quadratically but linearly; hence, there is a nuanced interplay between the three parameters. We train each of the networks for one epoch. While this does not fully train the models, a visual inspection of the training and validation curves consistently showed that networks that performed comparatively worse after one epoch were very likely to remain inferior throughout the entire training process, as seen in Fig. 5.2. We therefore make the assumption that early performance is a strong indicator of the overall training outcome.

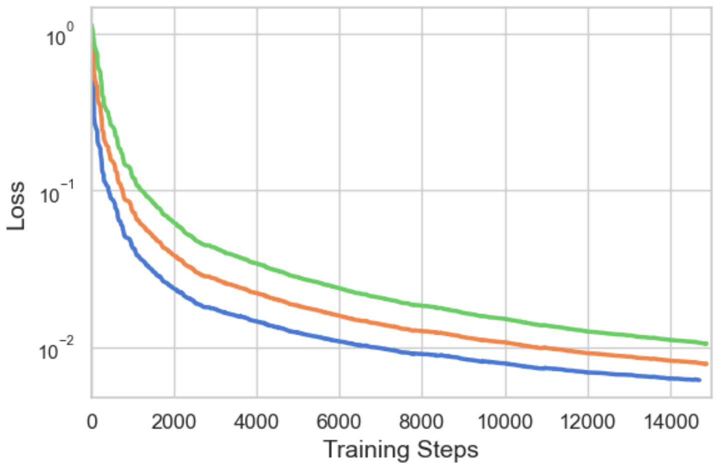


Figure 5.2: Training curves of different U-Nets (subset for clarity)

Table 5.1 summarizes the configurations and their corresponding validation losses after one epoch.

Blocks	Out Channels	# Parameters	Validation loss
6xDownBlock	[128, 128, 256, 256, 512, 512]	108 M	0.259
6xDownBlock	[64, 64, 128, 128, 256, 256]	27 M	0.288
3xDownBlock, 3xAttnDownBlock	[128, 128, 256, 256, 512, 512]	120 M	0.255
3xDownBlock, 3xAttnDownBlock	[64, 64, 128, 128, 256, 256]	30 M	0.309
4xDownBlock	[256, 256, 512, 512]	108 M	0.215
4xDownBlock	[128, 128, 256, 256]	27 M	0.264
2xDownBlock, 2xAttnDownBlock	[256, 256, 512, 512]	119 M	0.281
2xDownBlock, 2xAttnDownBlock	[128, 128, 256, 256]	29 M	0.275

Table 5.1: Diffusion U-Net grid search

From the results, we observe that the model with four DownBlocks and output channels [256, 256, 512, 512] achieves the lowest validation loss of 0.215, despite having a similar parameter count to more complex models. We see that increasing the capacity of the network by using more channels can lead to better performance, even without attention. As a matter of fact, including attention mechanisms did not consistently improve the validation loss (see Fig. 5.3). For example, the model with three DownBlocks followed by three AttnDownBlocks and output channels [128, 128, 256, 256, 512, 512] had a higher validation loss (0.255) compared to the simpler model without attention but with a similar parameter count. On the other hand, including several attention blocks did make the training process significantly slower. While there is certainly much more that could be tested and fine-tuned in terms of architecture, given the resource-intensive nature of training these models we decided to commit to the best performing configuration and move forward.

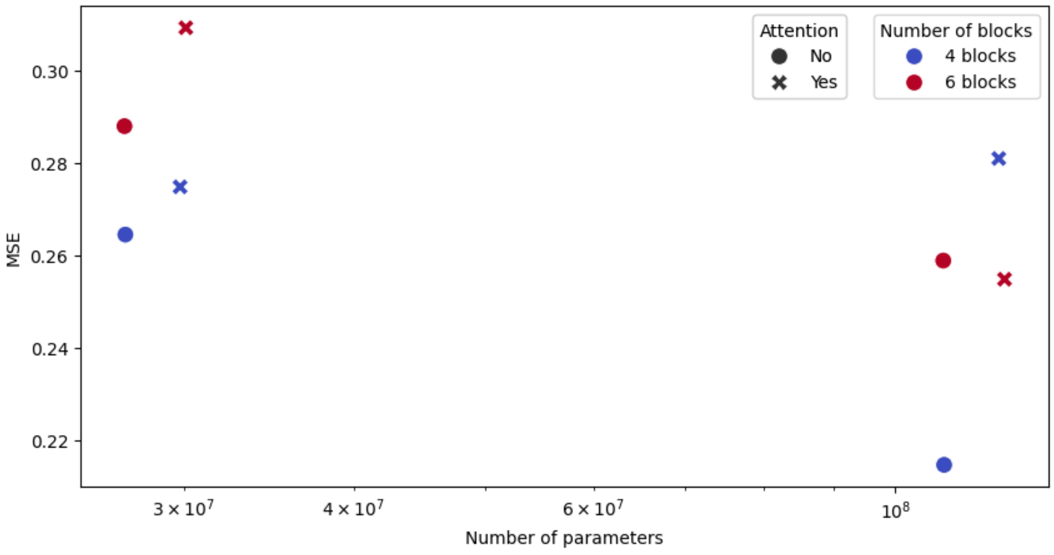


Figure 5.3: MSE vs number of parameters.

5.1.3. LDM

In this subsection, we present the results of our experiment with the LDM for next-frame prediction. Despite the theoretical promise of LDMs in handling high-resolution images, our application of LDMs to lower-resolution fisheye sky images, which could be effectively employed by a CNN, did not yield satisfactory results.
As detailed in Section 4.3, we employed a latent diffusion framework to predict future sky conditions based on past observations. The input consisted of sequences of images downscaled to 128x128 pixels to accommodate computational constraints and to align with the requirements of the subsequent image processing CNN. As explained in Section 2.2.4, LDMs operate by compressing images into a lower-dimensional latent space using a VAE. The diffusion process is then applied within this latent space, which reduces the computational requirements.

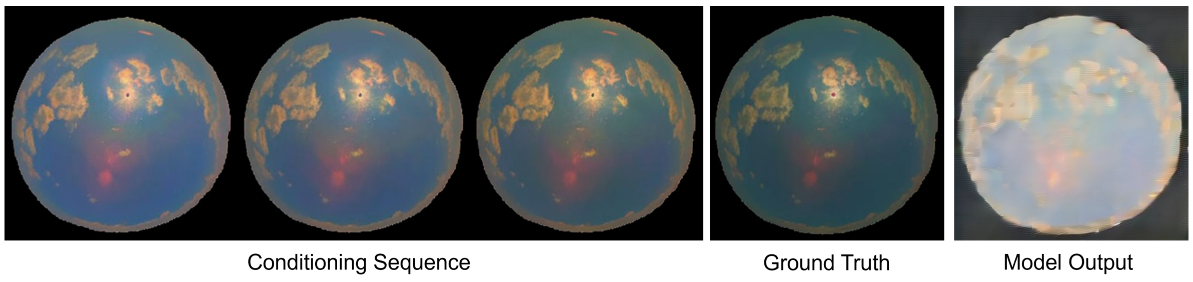
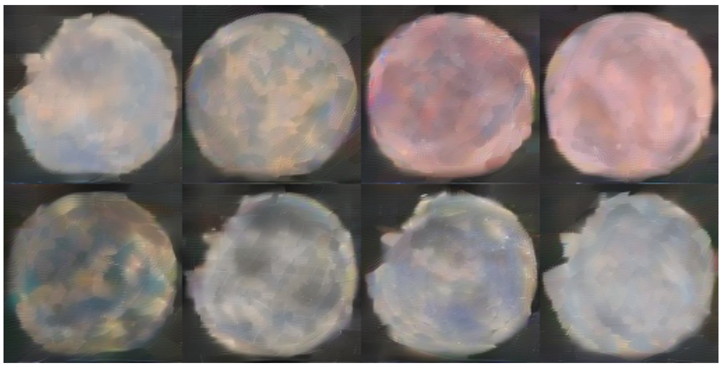
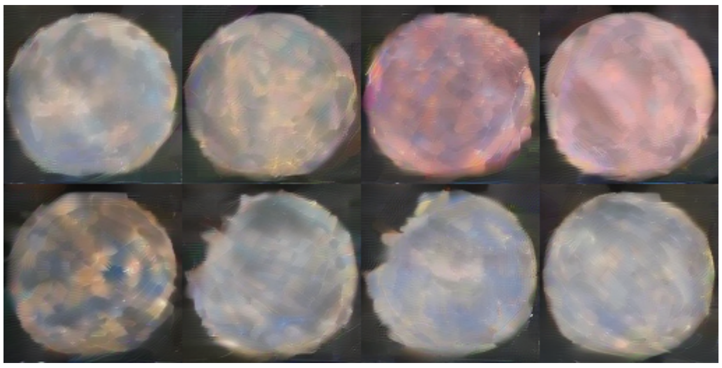


Figure 5.4: Sample of LDM inference.

Upon evaluation, we observed that the generated images from the LDMs capture the general circular structure characteristic of fisheye lenses, but lack critical details essential for determining the level of solar irradiance. In fact, the model learned that there are two different sky masks (corresponding to the two plants) and always generates either of the two, even in unconditional inference. Furthermore, the model is relatively accurate in its choice of colors, generating shades of blue, gray, and pink; all of them well represented in our dataset. However, as mentioned above, the images show a significant lack of detail, particularly in the representation of cloud formations, sky textures, and brightness variations. As can be seen in Fig. 5.5, this issue does not improve with longer training, the image quality remains the same in subsequent epochs, not only in terms of visual inspection, but also in terms of the training loss.

We hypothesize that the primary factor contributing to the model's underperformance is the low resolution of the input images. Latent diffusion models are designed to handle high-resolution images efficiently by performing the diffusion process in a compressed latent space. However, starting with already low-resolution images (128×128 pixels) and further compressing them via the VAE results in an excessively small latent space. As discussed in Section 2.2.5, in VAEs the latent space is essentially a compressed version of the input data, hence its dimensionality determines the level of detail that can be retained. With high-resolution inputs, the latent space is large enough to hold information about intricate patterns. In contrast, low-resolution inputs lead to a latent space that is too compact, effectively bottlenecking the information flow and preventing the model from generating detailed predictions.


(a) Epoch 1


(b) Epoch 2

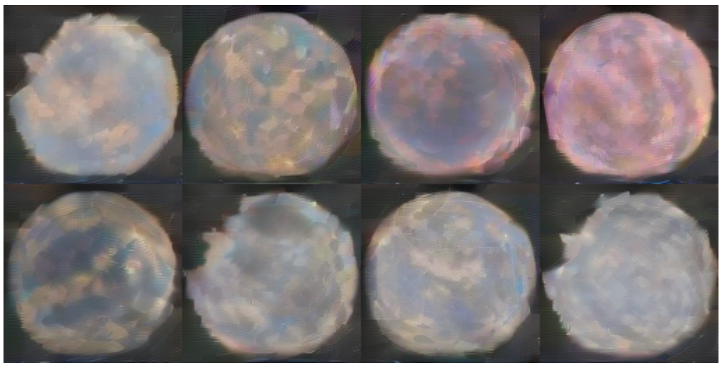

(c) Epoch 3

Figure 5.5: Latent generation samples

There are potential solutions to address the limitations encountered. The most straightforward is increasing the input image resolution. However, this not only would increase the already high computational demands, but also the downstream CNN can't take advantage of the higher resolution. We would need to downsize the image back to 128 pixels, rendering the additional complexity pointless. Alternatively, modifying the VAE to produce a higher-dimensional latent space might help preserve more information. This would involve reducing the compression rate by redesigning the encoder and decoder networks to maintain more features. However, since we are leveraging pretrained autoencoders, we have limited flexibility to alter their architectures. Designing and training a custom VAE from scratch would demand substantial time and computational resources, and most importantly offset the advantages of using latent diffusion. While theoretically possible, we consider this research direction not very promising and that it might divert excessive time and resources away from our main objective. Therefore, we have decided to shift our focus towards pixel space diffusion models, which operate directly on the image pixels without the need for latent space compression.

5.1.4. Benchmark of Next Frame Predictor

In this section we compare the fully trained and best performing diffusion architecture with other relevant methods. In particular, we benchmark against optical flow, ConvLSTM, and frame persistence.
Optical flow, thoroughly introduced in Section 2.1.2, estimates the motion vectors between consecutive frames to predict the future state of clouds. In our experiments, we leverage the open source implementation of the Farnerback method available via the OpenCV library [18]. As mentioned in 2.4, ConvLSTM was one of the first successful methods to integrate CNNs with recurrent architectures, and paved the way for subsequent advances in video frame prediction, making it a critical baseline for comparing the performance of our proposed method. Similar to the DM, we train the ConvLSTM model on the whole dataset until converged, following the implementation in [95]. Finally, frame persistence simply consists in copying the last available frame as a prediction.
We employ two common metrics to compare images: MSE and Structural Similarity Index (SSIM). The MSE, which has also been used as the loss function to train the ML algorithms, computes the squared difference between the predicted and actual pixel values. The SSIM, on the other hand, evaluates the similarity between two images focusing on luminance, contrast and structure. We do not provide a mathematical description of this metric, but the interested reader is referred to [96]. It is worth mentioning, however, that the MSE ranges from 0 to infinity, lower values being better, while the SSIM ranges from -1 to 1, values closer to 1 being better. Note that some methods scale pixel values to the range [0, 1] while other methods keep the original range [0, 255]. To allow a fair comparison we rescale the pixel values to [0, 255] when necessary.
The results are shown in Table 5.2. As can be seen, in terms of MSE, persistence outperforms the rest of the models by a considerable margin. We attribute this superior performance to the inherent simplicity of the Persistence approach. Given that brightness and color variations heavily influence the MSE metric, the straightforward nature of Persistence, which merely replicates the last frame without accounting for any dynamic changes, leads to lower MSE values. Similarly, optical flow, which also employs the previous image as a base for its predictions achieves a relatively low MSE. On the other hand, the models that have to create the predicted images "from scratch", i.e. ConvLSTM and Diffusion, fare worse in this comparison.

Model	MSE	SSIM
Diffusion	298.64	0.9006
OF	271.94	0.8396
ConvLSTM	430.84	0.8260
Persistence	235.00	0.8590

Table 5.2: Comparison of different next-frame prediction models

However, we theorize that the SSIM provides a better quantification of quality of the prediction, especially keeping in mind the downstream task of using it to predict irradiance. By incorporating structural information, SSIM ensures that models accurately replicating cloud patterns receive higher scores, even if there are minor discrepancies in color or brightness. In our results, the DM achieved the highest SSIM score, showing it's ability to preserve the structural integrity of the images, and to forecast cloud patterns effectively. Consequently, we believe the diffusion model will be the most effective for irradiance forecasting (see Section 5.3).

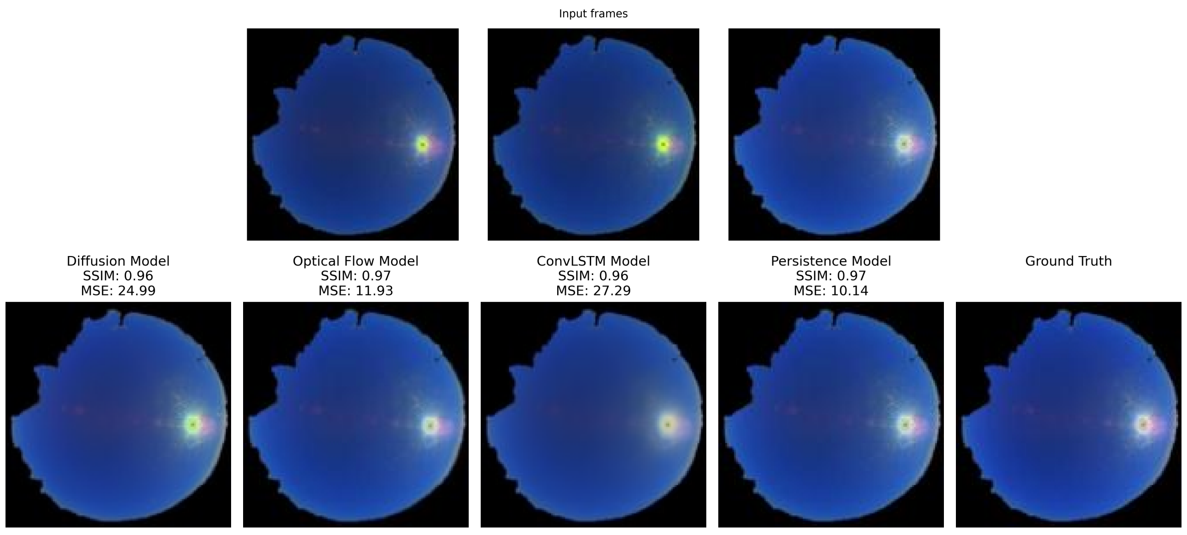


Figure 5.6: Next frame prediction sample

Fig. 5.6 is a good example of the above mentioned issue. As can be observed, the input frames are a sequence of sunny images, without any movement. Under visual inspection, all the models seem to do a relatively good job at predicting the next frame, this would be expected given the simplicity of this task. Nevertheless, we can observe significant differences in the metrics. The optical flow and the persistence models achieve a much lower MSE, while the convLSTM and the Diffusion fare poorly in this comparison. Under further inspection of the images we can confirm that indeed the two deep learning models show minor inconsistencies in color and brightness, which compound over the whole image, deteriorating this metric. Given that the dataset contains several such sequences, this explains the overall worse performance of the ML-driven methods in terms of MSE.
When looking at more dynamic sequences, we can observe the superior performance of our proposed diffusion architecture. For instance, Fig. 5.7 and 5.8 show respectively a mostly clear sky with fast moving clouds and a relatively cloudy image with large clouds with complicated changing shapes. In both of these, we can visually observe the superior performance of diffusion. Once again, this assessment is not corroborated by the MSE metric, but it is by the SSIM. For instance in the latter sample the diffusion model achieves a SSIM of 0.76, while the rest of the models barely obtain 0.6. While the diffusion model produces a smoother picture, lacking sharp details, it understands the overall movement and changing cloud patterns and comparatively succeeds at mapping the future position and shapes of clouds. On the other hand, the optical flow model fails to produce a realistic image, warping the previous frame too much. This is caused by the large and rotating motion vectors present in this sequence. The ConvLSTM model also performs poorly, as it excessively blurs the center of the image. This likely occurs because the model is trained using MSE loss. When the model is uncertain about the next frame, it tends to generate a washed-out and blurry prediction, a strategy that minimizes the overall MSE, but results in a ”non-committal” output. Finally, the persistence model performs surprisingly well for a dynamic sequence, it achieves its typically low MSE while obtaining the second best SSIM.

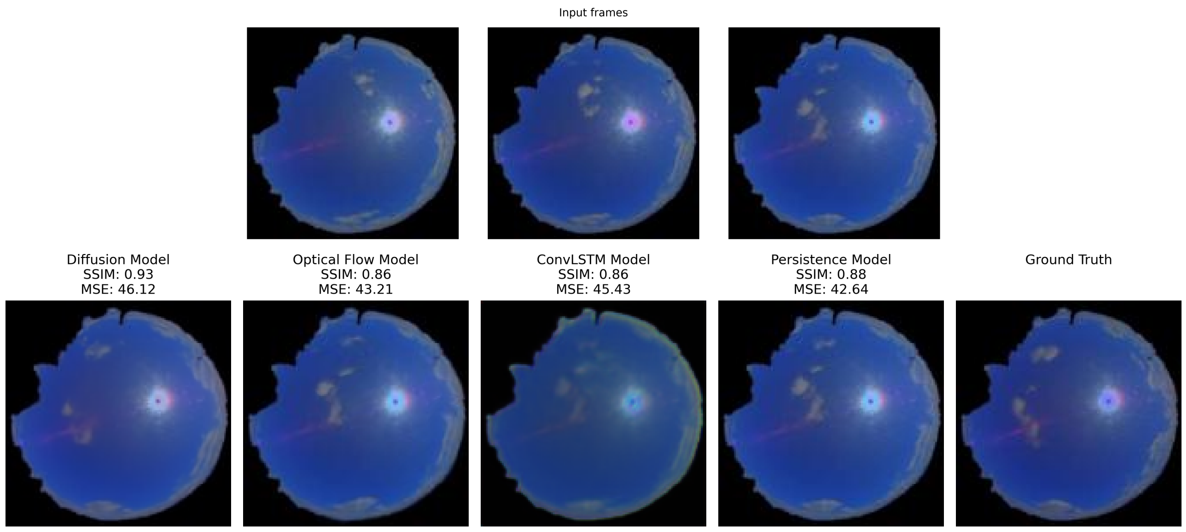


Figure 5.7: Next frame prediction sample

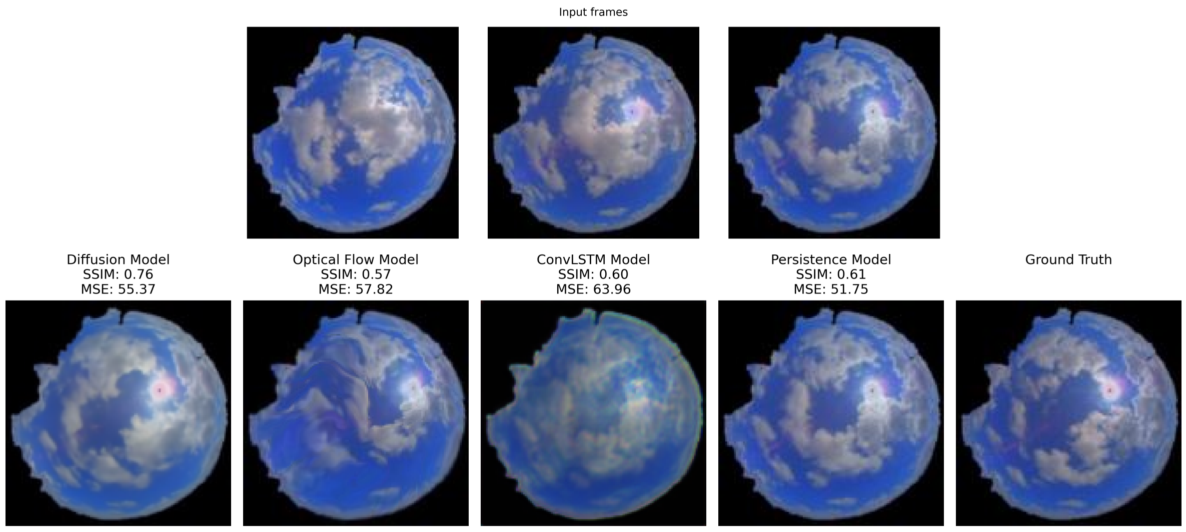


Figure 5.8: Next frame prediction sample

Similarly, Fig. 5.9, illustrates the better understanding of cloud dynamics by the diffusion model. If the reader directs its attention to the bright spot in the lower part of the image, where there is some sunshine coming through, it can be observed how all the models try to map its location with varying levels of success. However, as reflected by the SSIM metric, the diffusion model performs the best, even though its image still lacks some fine details.

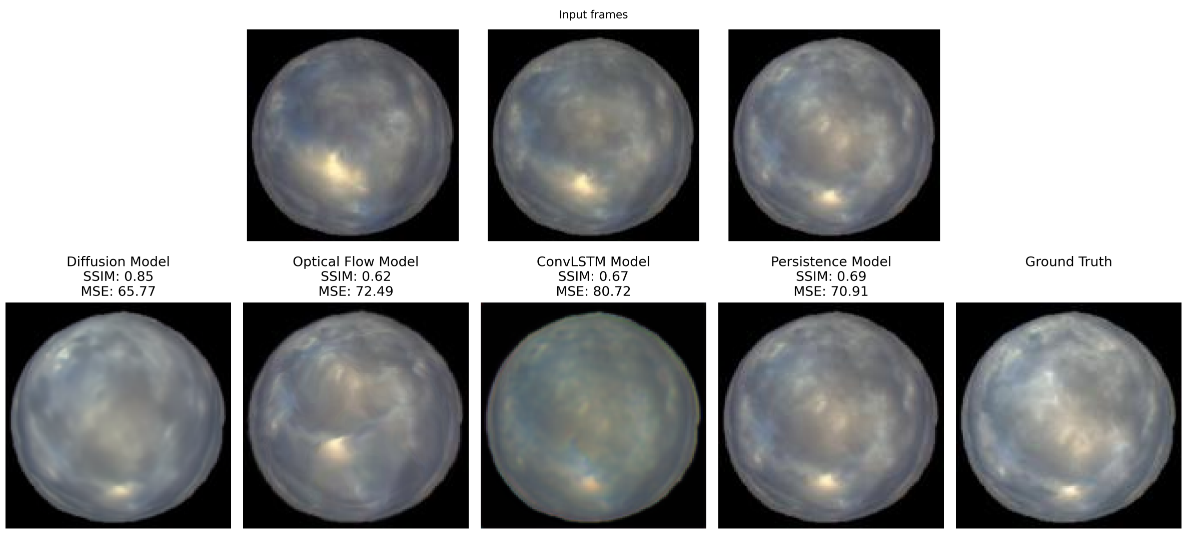


Figure 5.9: Next frame prediction sample

Another unique feature of the diffusion model is its ability to conceal the Sun even if it was shown in the previous frames or reveal it even if wasn't. For instance, Fig. 5.10 shows an overcast sequence in which the sun is not visible at all. Nevertheless, there is an area of thinner clouds where some sunlight breaks through, which is moving towards the Sun. The diffusion model is the only one able to perceive this, and correctly reveals the Sun when the gap in the clouds reaches its location.

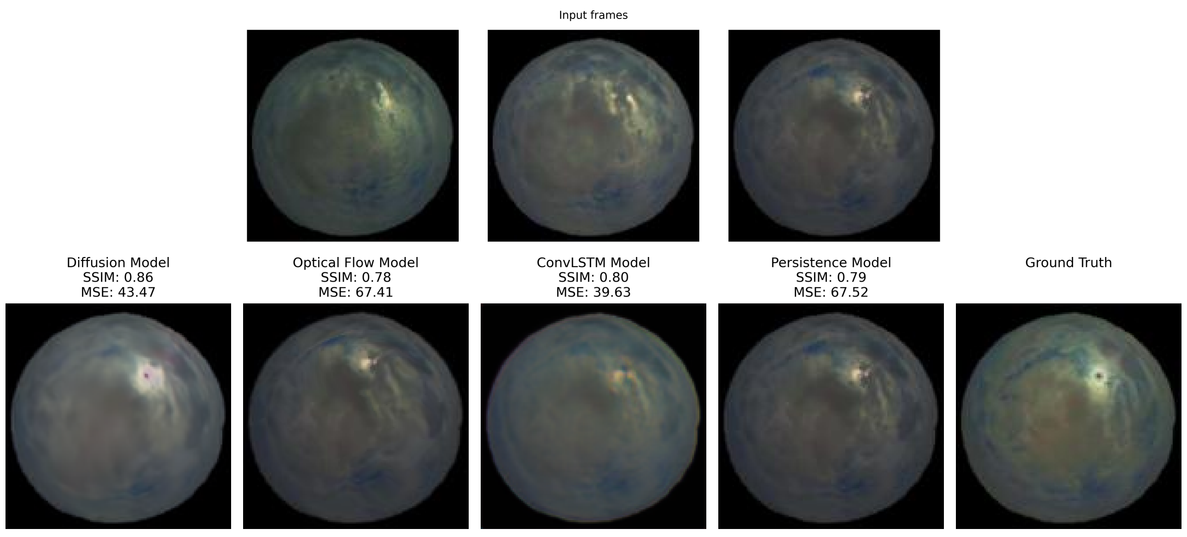


Figure 5.10: Next frame prediction sample

The opposite situation can be observed in Fig. 5.11 and Fig. 5.12. Both are sunny sequences, but there are clouds moving which will soon occlude the Sun. Again, the diffusion model is the only one able to remove the distinctive dark dot that indicates the Sun is directly visible. This feature of the diffusion model is of paramount importance for us, given that the main objective of this work is to be able to accurately predict sharp irradiance oscillations.

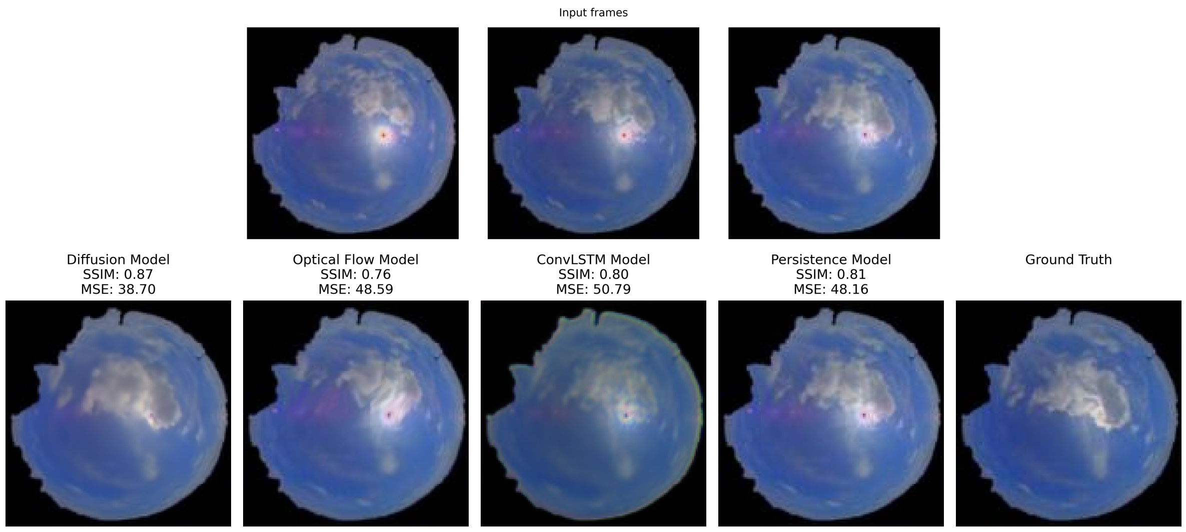


Figure 5.11: Next frame prediction sample

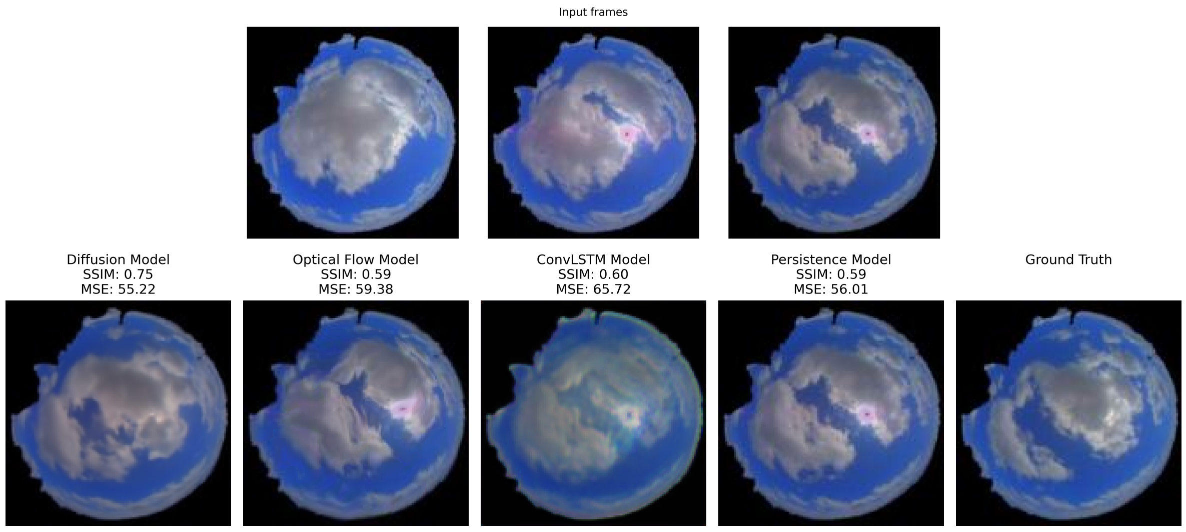


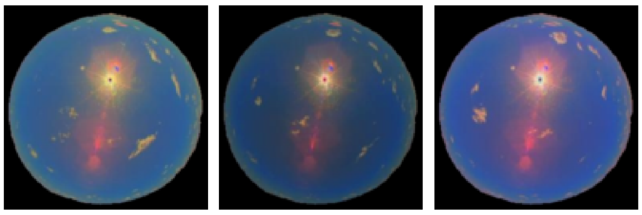
Figure 5.12: Next frame prediction sample

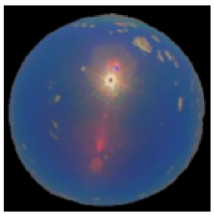
5.1.5. Stochasticity in Diffusion Inference

As described in Section 2.2.4, the images generated by a DM are not deterministic. The starting point of the denoising process is precisely random noise, sampled from a standard normal distribution, which makes each inference loop unique. This is one of the characteristics that made DMs so popular in the first place; their ability to generate diverse samples, as opposed to GANs, which are prone to mode collapse. In this section, we aim to determine whether some of the stochasticity remains after the model is conditioned on a past sequence of frames, or if the conditioning information has such a strong influence on the denoising process that samples generated from the same sequence become essentially indistinguishable.

As can be seen in Fig. 5.13, 5.14, and 5.15 there is indeed some variability in the generated frames, even after conditioning the model. While the conditioning information helps guide the denoising process, the inherent randomness from the initial noise persists, leading to subtle but noticeable differences between samples.

Fig. 5.13 shows a clear sky with a few moving small clouds. The biggest challenge in this sequence is the inconsistent level of brightness, which as we have seen has a big influence on the MSE. It is a priori impossible to know whether the future frame will be comparatively brighter or darker than the previous. We can see this high uncertainty reflected on the results. The model generates frames with varying levels of brightness, depending on the sampling of the initial noise.


(a) Conditioning Sequence


(b) Ground Truth

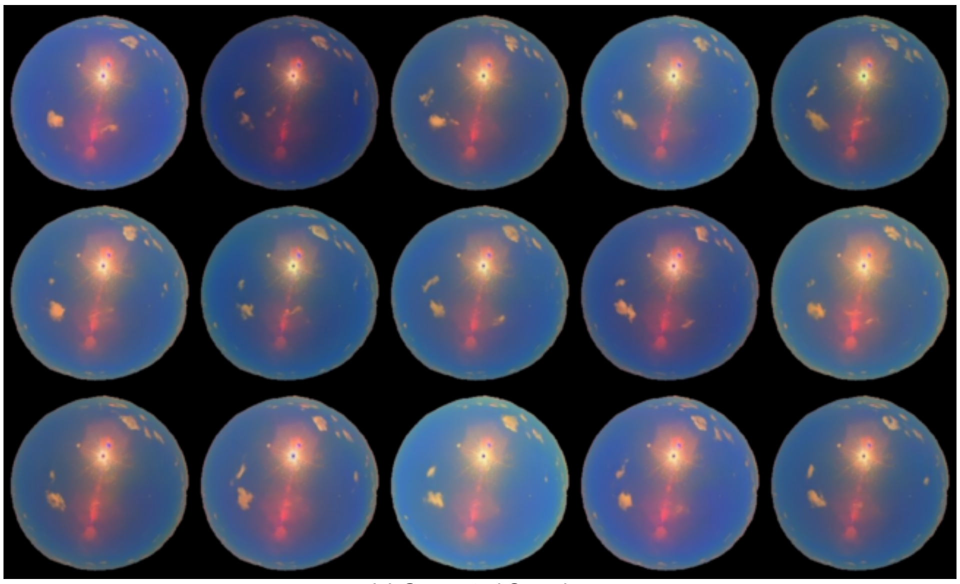
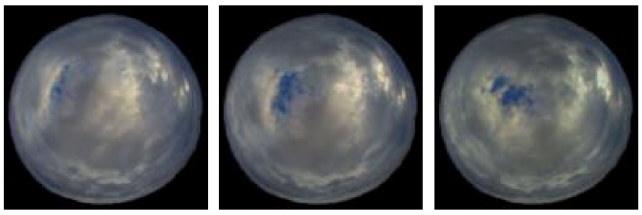
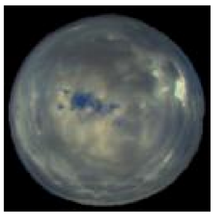

(c) Generated Samples

Figure 5.13: Example of stochasticity in diffusion inference


(a) Conditioning Sequence


(b) Ground Truth

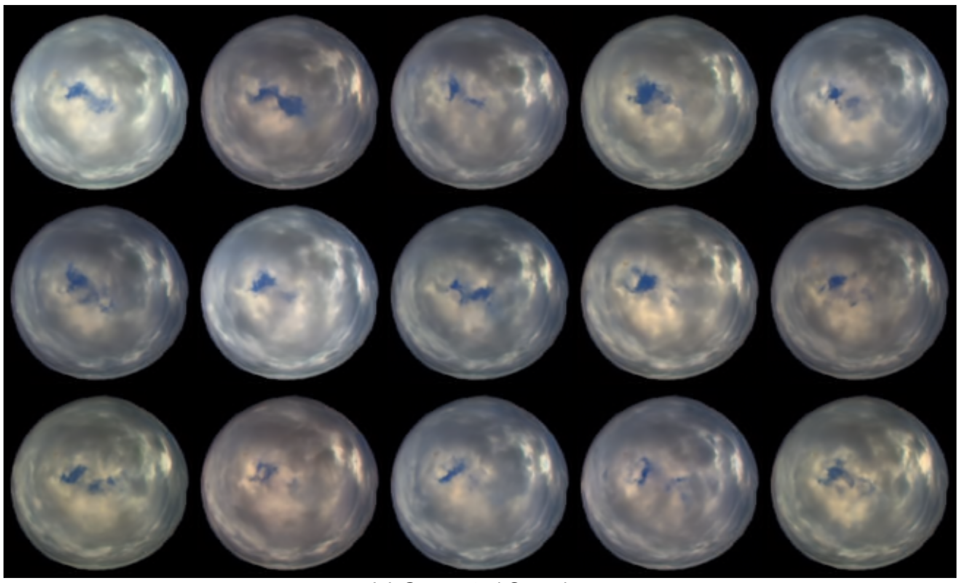
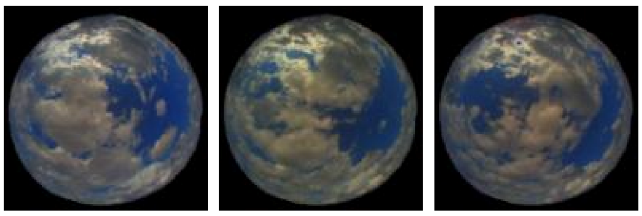
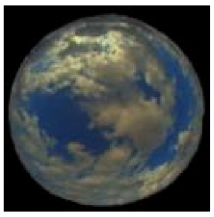

(c) Generated Samples

Figure 5.14: Example of stochasticity in diffusion inference


(a) Conditioning Sequence


(b) Ground Truth

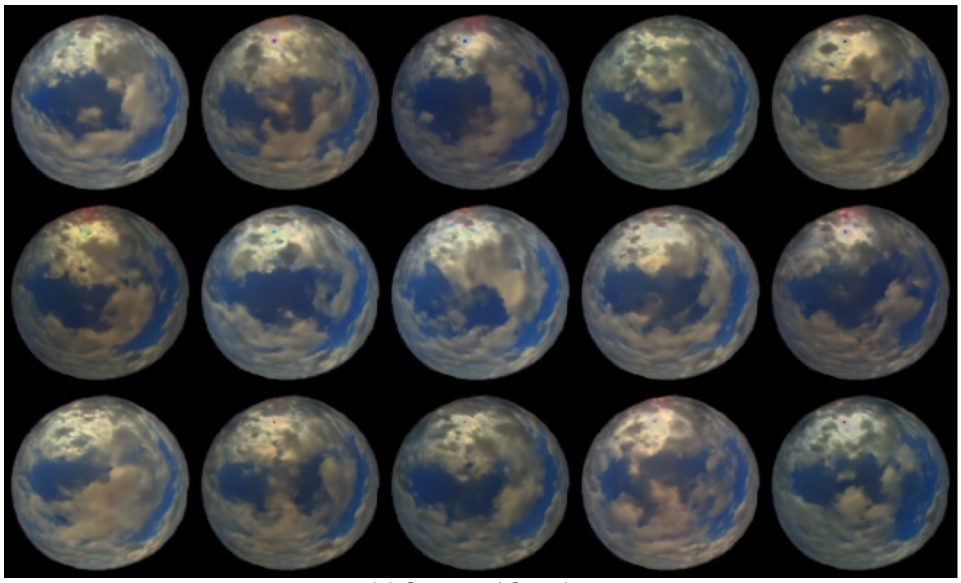

(c) Generated Samples

Figure 5.15: Example of stochasticity in diffusion inference

Fig. 5.14 shows a sequence that starts out very cloudy, with the clouds gradually clearing as the sky opens up. As shown, the model makes various predictions about how this scenario will unfold, resulting in significantly different sky conditions in the final frames.
Lastly, Fig. 5.15 shows a particularly challenging situation, one susceptible to contain a significant jump in the GHI. As can be observed, the Sun is not directly visible in the conditioning sequence, however the clouds are moving in a way that could potentially reveal it. As shown in the ground truth, in the end the sun remains occluded, but we can see the model predicting both outcomes.
We believe this stochasticity is not an undesirable property of our model but rather a valuable feature that enhances its utility. In the context of power systems operation and energy forecasting, where future states are inherently uncertain, probabilistic predictions are often preferred over deterministic ones. Probabilistic predictions provide a range of possible outcomes, which enable better risk management (e.g. by allowing grid operators to prepare for different situations).
We do not go any further in this section but we will leverage this stochasticity once we assemble the full pipeline to produce probabilistic irradiance predictions (see Section 5.3.2).

5.1.6. Recursive Inference

As discussed in Section 5.1.1 we set the prediction horizon at one minute. This decision was motivated by the fact that within a one-minute interval, clouds can move, deform, or dissipate, resulting in sharp fluctuations of irradiance. However, in practical applications, particularly in the energy sector, a one-minute forecast offers limited utility. For example, if we consider energy markets, the minimum prediction horizon should be over 15 minutes. Because the last trading window in which market participants can place orders closes 15 minutes before delivery. Similarly, grid operators and energy producers benefit more from predictions that extend 5 to 10 minutes into the future, allowing them more flexibility to make decisions regarding grid stability and managing contractual obligations.
To address this, we explore the concept of recursive inference. Recursive inference consists on feeding the model its own predictions to generate forecasts further into the future. In our case, this entails generating a frame one minute into the future and using it as part of the input for the subsequent prediction

(see Fig. 5.16). By iteratively inferencing the model in this manner, we can extend the forecast horizon without modifying the model's architecture or retraining it for longer prediction horizons.

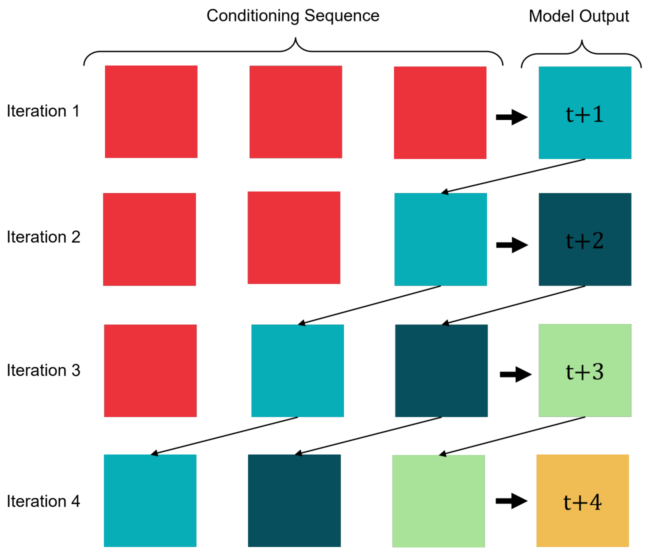


Figure 5.16: Recursive inference schematic.

As expected, when we recursively feed the model its own predictions, we observe a progressive degradation in the quality of the generated frames. Fine details, such as cloud edges and textures, gradually fade, resulting in smoother and less defined images. Furthermore, the position and shape of clouds diverge more and more from the truth with each iteration. These two phenomena are well documented in the literature. Models trained on MSE loss often produce smoothed forecasts that lack sharp details. This is because the MSE treats all deviations equally and penalizes large errors more heavily due to the squaring operation. When the model is uncertain about the exact outcome (e.g., the precise position of a cloud), it tends to predict the mean of all possible outcomes to minimize the expected loss, which results in blurred representations where sharp features are averaged out. With each iteration, this uncertainty increases, leading to progressively blurrier predictions. This is particularly pronounced in situations where the sky conditions are complex, when the sky features numerous clouds of varying thicknesses and textures, as seen in Fig. 5.17. In these situations there is presumably more intrinsic uncertainty, hence the averaging effect is aggravated. On the other hand, sequences of clear sky, such as the one depicted in Fig. 5.18, with very limited uncertainty, feature a much slower degradation.

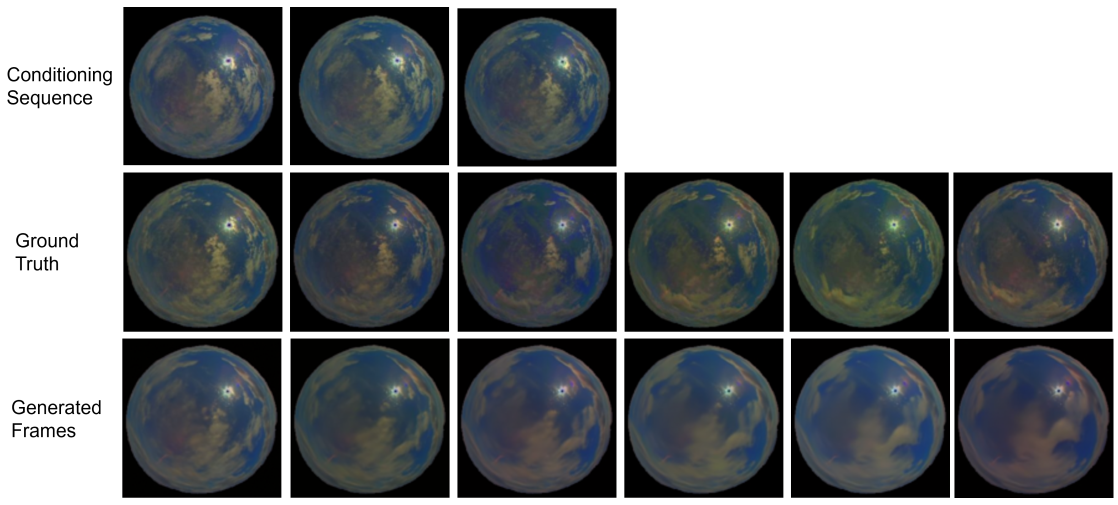


Figure 5.17: Example of recursive inference in challenging conditions

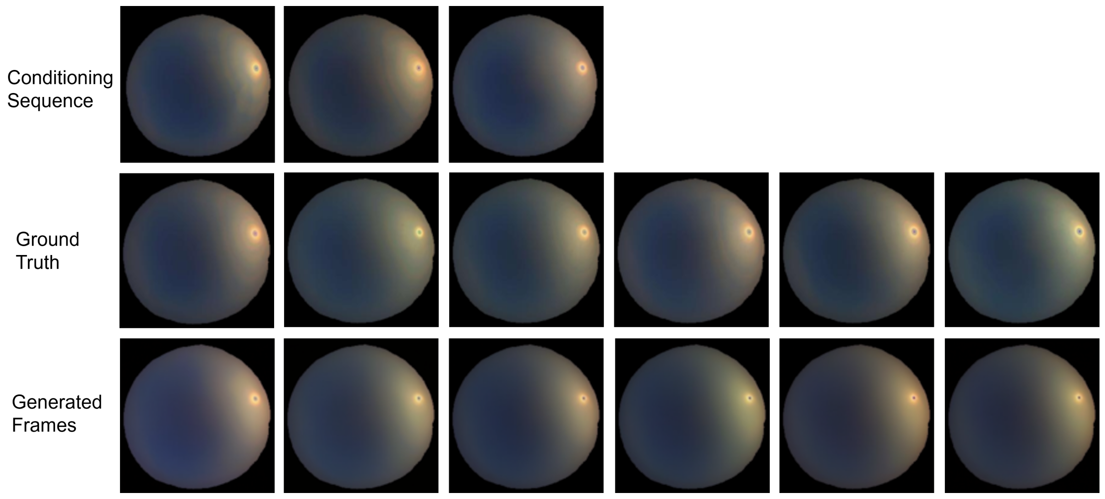


Figure 5.18: Example of recursive inference in stable conditions

As can be seen in Fig. 5.19, despite the increasing lack of fine-grained details, the model does capture the general direction and velocity of cloud movements. However, over time, the ground truth and the model predictions visually diverge in terms of the specific sky conditions depicted. While the model maintains a reasonable representation of overall cloud dynamics, the exact arrangement and appearance of clouds increasingly differ from reality. We expected this behaviour and presume it is rooted in two well-known facts. First, feeding the model its own predictions introduces a shift in the input distribution. The model was trained on real images, and its predictions may not fully capture the statistical properties of the true data distribution. As the model continues to generate new inputs based on its own outputs, these deviations accumulate, leading to increased errors and loss of detail. More importantly, we hypothesize, is the fact that small discrepancies in early predictions compound and magnify in later steps. Chaotic systems, like the weather, are characterized by the fact that small differences in the initial conditions can lead to vastly different outcomes. Similarly, when we iteratively inference the DM small inaccuracies compound and lead to considerably different results.

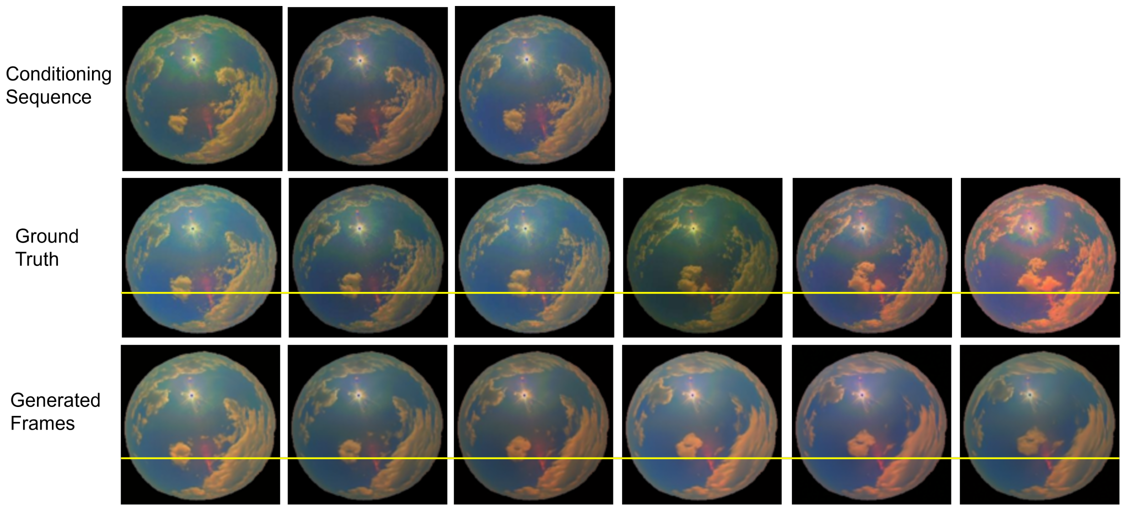


Figure 5.19: Example of recursive inference. Yellow line provided as reference to illustrate cloud movements.

5.2. Image to Irradiance prediction

In the second stage of our proposed pipeline, we employ a CNN to transform the synthetic future sky images into precise GHI (used interchangably with ”irradiance”) predictions. This NN processes the generated frames in conjunction with additional features to try to model the relationship between a given image of the sky and the irradiance level. In what follows, we quantify the impact of the additional features, we search for the optimal network configuration, and we explore different training approaches.

5.2.1. Additional Features

The additional features that we have developed and presented in Section 4.2 were mostly motivated by intuition and prior experience of the involved researchers. In this section, we perform an ablation study to determine whether the additional features indeed incorporate useful information that boosts the performance of the network. Recall the additional features are the following:

- **Binary mask** that contains the precise location of the Sun
- **Previous Irradiance** corresponding to the last available image (in the standard case 1 minute before the prediction).
- **Day of the year** encoded using geometric functions into two features.
- **Hour of the day** encoded using geometric functions into two features.

We design the ablation study shown in Table 5.3 to assess the contribution of each additional feature to the performance of the CNN in predicting irradiance. The validation MSE serves as the evaluation metric, with lower values indicating better accuracy. The network we use for this is a relatively simple CNN consisting of three convolutional-pooling layers followed by two FC layers, following the structure presented in Section 4.1. We believe that this network provides generalizable insights on whether the additional features contain useful information or not, which can later be applied to more complex architectures.

Binary mask	Previous irradiance	Hour of Day	Day of Year	Validation MSE
				3281
X				2981
X	**X**			1832
X	**X**	**X**		1831
X	**X**	**X**	**X**	1712
X	**X**		**X**	1788
	X	**X**	**X**	1920

Table 5.3: Ablation study on additional features, **X** indicates that the feature is present

From the results, it is clear that each of the features we designed contributes to the model's performance, although to varying degrees. Using only the sky image as input, the baseline model achieved a validation MSE of 3281, which we will use as a reference point. When the binary mask was added, the validation MSE decreased to 2981. Explicitly indicating where the sun is located is arguably equivalent to providing information about the time of the day. As expected, the model can leverage this information to interpret the sky image more effectively. We further hypothesize that by knowing where the Sun is located in the image, the model can focus on the regions most relevant to solar intensity, enhancing performance.
Including the previous irradiance measurement alongside the binary mask led to a significant reduction in validation MSE, bringing it down to 1832. This substantial improvement highlights the strong temporal correlation of irradiance values over short intervals. As mentioned multiple times throughout this work, the previous irradiance serves as a powerful predictor and persistence models often perform extremely well.
Interestingly, when the hour of the day was added to the input (along with the binary mask and previous irradiance) the validation MSE remained the same. A plausible explanation is that the hour of the day is already implicitly represented in the binary mask, hence including it explicitly does not provide additional information. When we incorporate information about the day of the year (configuration with all the additional features), the validation MSE decreased to 1712, the lowest among all configurations.

Since adding the hour of the day did not initially yield an improvement, we test the configuration that includes all the features but this one. Interestingly, we find that excluding this feature is now detrimental to performance. This results suggests that even though the hour of the day may be redundant with the binary mask, when combined with information about the day of the year it does enhance the model's performance. This is likely due to the fact that seasonal information is crucial to determine, given the time of day, whether there is sunlight and how intense it might be.
Finally, we conduct one last experiment to accurately quantify the impact of including the binary mask, since this feature took significant time and effort to create compared to the rest. We find that excluding the binary mask does indeed result in poorer performance.
While the binary mask is concatenated along the channel axis of the image, the remaining features are concatenated along the 1-dimensional feature map right after the flattening that precedes the FC layers. This is the most common and straightforward way to incorporate additional information into an image processing network. However, there exists an approach that can, in some cases, yield slight performance improvements. It consists on preprocessing the additional features with a relatively small MLP, which balances the number of tensor positions associated with the processed image and the additional features. Without this step, the tensor before the FC layers can be almost entirely made up of entries that correspond to the output of the convolutions, while only a few entries (in our case 5) correspond to the additional features.
Furthermore, we also investigate a reframing of the objective of the CNN, which has also been found to improve performance in some cases. This consists on adding the previous irradiance to the output of the network before computing the loss. What we achieve with this is that the network learns to predict increments or decreases in irradiance rather than the irradiance value itself.
As can be seen in Table 5.4, none of these methods significantly increased performance; therefore, we discard them favoring the simpler approach.

Network	Validation MSE
Baseline	1712
Feature preprocessing	1713
Incremental predictions	1745

Table 5.4: Results using additional feature preprocessing and incremental predictions.

5.2.2. Architecture Search

The purpose of this section is to find the best possible network architecture to process the images generated by our DM. We will focus on convolutional architectures since they have proven good performance in image regression, are relatively easy to implement, and can be trained fast. In order to organize the architectural search we define two families of networks: CNNs and ResNets. Note that this distinction slightly abuses terminology since ResNets are a specialized form of CNNs. However, for the sake of clarity, we'll refer to standard CNNs without residual connections as "CNNs" and those with residual connections as "ResNets. The architecture in the first family of networks can be precisely defined with two lists: one that contains the out channels for each of the conv-pooling blocks and one that contains the number of neurons in each of the FC layers (see Section 4.3). For the second family of networks, we follow a well-known nomenclature for ResNets, which consists of simply naming each network with the number of layers it contains (e.g. ResNet18, ResNet24,...) [29].
For the CNN grid search, we start with a relatively simple architecture, the one used for the ablation study on additional features, and modify the most crucial hyperparameters. The number of channels for the convolutional layers, the number of pooling layers, the number of FC layers, and how many artificial neurons they contain. However, we keep them relatively shallow and with parameter counts around 1 million. For deeper and more complex networks we rely on the ResNet family, which will not suffer from vanishing gradients. In particular, we tested 7 ResNets, with the number of layers ranging from 10 to 50. For all of the experiments we train the network until it is fully converged and report the validation loss. The results of the experiments can be seen in Table 5.5 and Table 5.6. We also tried modifying other parameters of the convolutions, such as the padding, or kernel size and average pooling instead of max pooling, but none of these yielded significant improvements.

Conv-Pooling	FC	# Parameters	Validation loss
[32,64,8]	[512, 256, 1]	1.21 M	1712
[32,64,64,32]	[512, 256, 1]	1.25 M	1691
[32,64,16]	[256, 128, 1]	1.11 M	1811
[32,64,8]	[512, 256, 128, 1]	1.5 M	1819

Table 5.5: Validation loss for different CNN architectures.

ResNet	# Parameters	Validation loss
ResNet10	4.66 M	2417
ResNet14	6.46 M	2237
ResNet18	11.2 M	2284
ResNet22	12.7 M	2336
ResNet26	17.5 M	2395
ResNet34	21.3 M	2271
ResNet50	23.8 M	2329

Table 5.6: validation loss for different ResNets.

Analyzing the results, we observe that among the CNN architectures, the model with convolutional layers [32, 64, 64, 32] and FC layers [512, 256, 1] achieved the lowest validation MSE of 1691, surpassing the ”baseline” model by a small margin. Adding an extra Conv-Pooling block to further reduce the height and width of the tensor while increasing the depth yields a slight performance improvement. On the other hand, shifting some of the parameters from the FC layers to the Conv-Pooling blocks is detrimental.
For the ResNet architectures, the best-performing model was ResNet14, with a validation MSE of 2237. Despite the more depth and sophistication of ResNets, none of the architectures outperformed the simpler CNNs. Furthermore, increasing the depth of the ResNet models did not consistently improve performance; in fact, some deeper networks performed worse than shallower ones.
These findings suggest that, for our specific task of irradiance prediction from sky images, relatively simple CNN architectures without residual connections suffice, and more complex models might be at risk of overfitting.

5.2.3. Training on Generated Frames

So far, we have trained all image processing networks on true photographs of the sky. The reader will have noticed that this is not exactly our use case. The images that the CNN will take as input are not true photographs, but frames generated by the DM. In this section, we bridge this gap between training data and the intended application. We experimented with augmenting the training dataset by incorporating generated images from the DM. The objective is to determine whether including synthetic data can improve the network's performance on generated images without significantly compromising its accuracy on true images.
Our findings are illustrated in Fig. 5.20, which depicts the trade-off between performance on true images and generated images as the proportion of synthetic data in the training set varies. The network trained exclusively on true images performed best when evaluated on true photographs, as expected. However, introducing small amounts of generated images during the training of the network did not deteriorate the performance on true data by a lot, while it did improve the network performance when evaluated on the generated dataset. Notably, a moderate inclusion of synthetic data of 5-15% enhanced the network's ability to generalize to generated images without a substantial decrease in performance on true images. Increasing this share even further, to around 50%, we observe that the performance in generated data keeps improving, but this time at the cost of a notable degradation of performance in the

true dataset. Interestingly, increasing the proportion of generated images beyond a certain threshold led to a decrease in performance across both true and synthetic images.

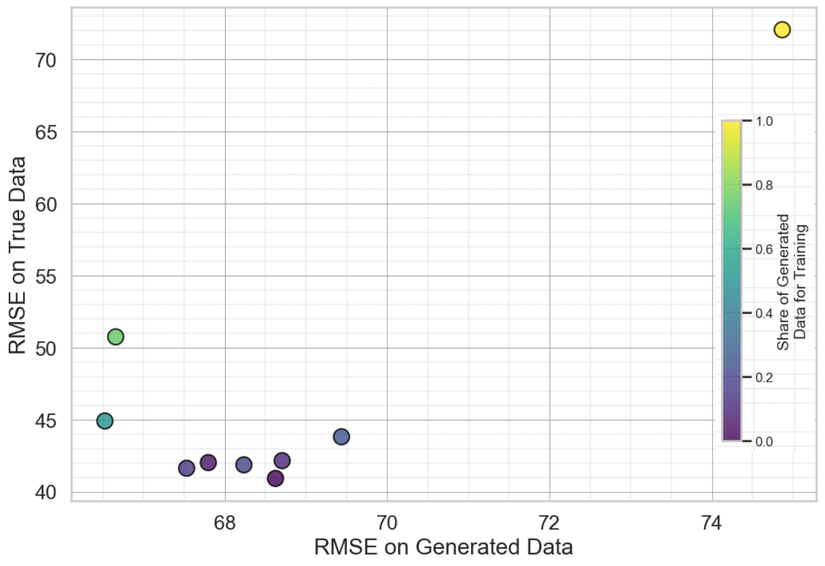


Figure 5.20: Pareto plot of RMSE in generated and true images

It is important to note that slight inconsistencies might be observed (e.g. the outlier network trained on 20% generated images), which can be attributed to factors such as the randomness in network initialization and the stochastic nature of sampling during training. However, we conclude that while the training dataset must predominantly consist of true images, including a small share of generated images so that the network can be acquainted with their particularities can yield slight performance improvements.

5.3. Case Studies

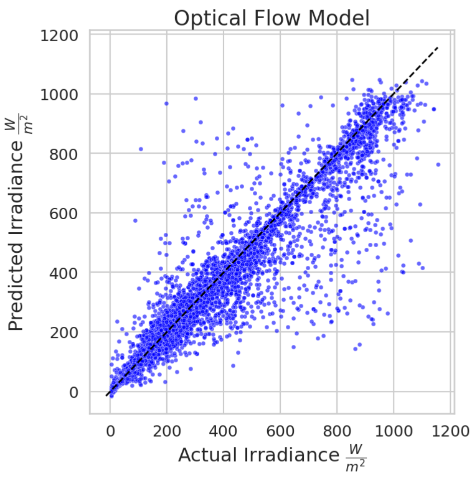
5.3.1. Point predictions

In this case study, we analyze the performance of four irradiance prediction models: ConvLSTM, Diffusion, Optical Flow, and Persistence. We evaluate their effectiveness using three performance metrics: Mean Absolute Error (MAE), MSE and the coefficient of determination R^2.

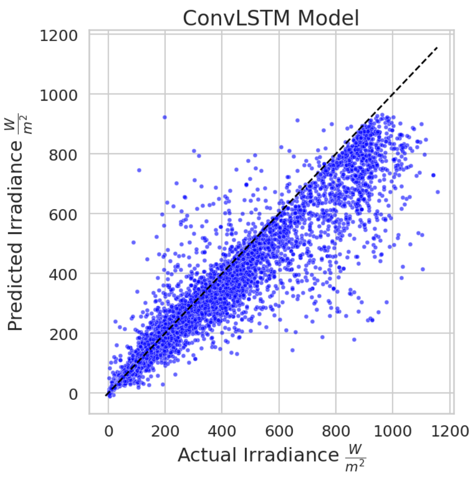
The overall performance of the models across the dataset is summarized in Table 5.7 and Fig. 5.21. As expected, the persistence model performs remarkably well, achieving the lowest RMSE and a near-perfect R^2 of to 0.96. Regarding the ”intelligent” models, optical flow show reasonable performance but is outperformed by Diffusion while ConvLSTM performs the worst by a significant margin. As observed in Fig. 5.21 and 5.22, all the models but persistence show a bias towards positive residuals (i.e. underestimating irradiance).

Model	MAE	RMSE	R^2
ConvLSTM	43.2	87.1	0.88
Diffusion	30.4	70.8	0.92
Optical Flow	33.8	78.4	0.90
Persistence	30.8	73.0	0.92

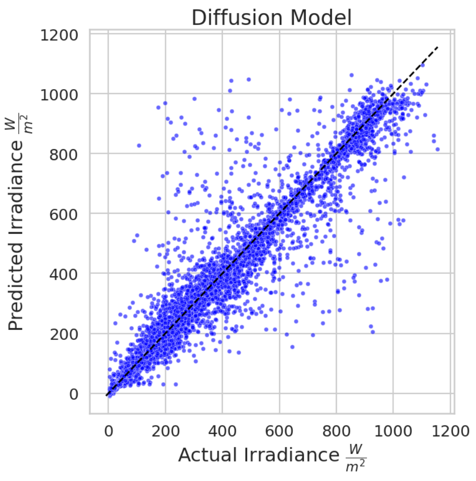
Table 5.7: Overall performance comparison



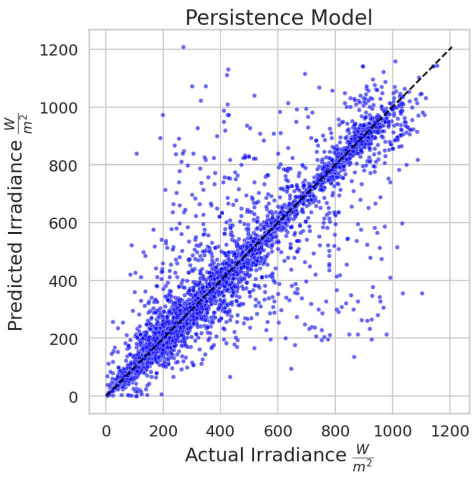
(a) Optical Flow



(b) ConvLSTM

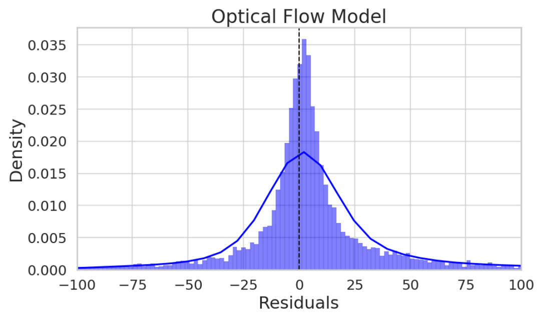


(c) Diffusion

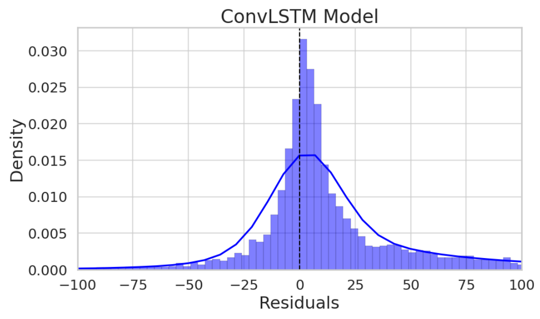


(d) Persistence

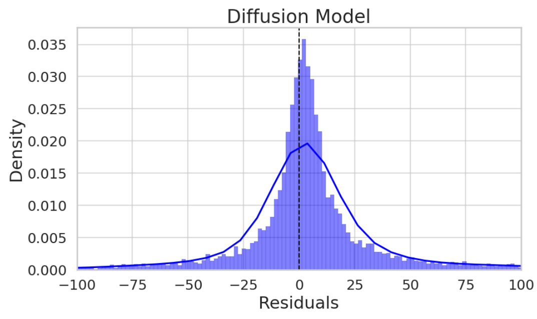
Figure 5.21: Scatter plot of true vs predicted irradiance



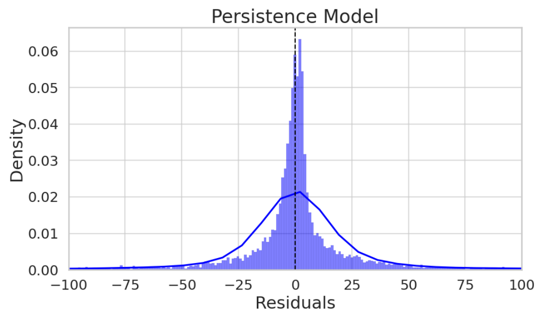
(a) Optical Flow



(b) ConvLSTM



(c) Diffusion



(d) Persistence

Figure 5.22: Distribution of residuals

As mentioned throughout this work, we expect persistence to be a very strong predictor in the overall dataset given that the vast majority of time irradiance changes are smooth. Nevertheless, persistent situations do not pose a challenge for grid stability, hence we analyzed the models on sequences exhibiting more than a 15% change between consecutive irradiance measurements. The results are presented in Table 5.8. In this case, the diffusion model performs the best achieving the lowest MAE and the highest R^2 among all models. When we remove the persistent situations from the test set, the persistence model naturally experiences a substantial drop in performance, with the highest MAE and the lowest R^2, underlining its intrinsic limitations during dynamic situations. Surprisingly, ConvLSTM, which performed worst in the global benchmark comes now second, followed closely by optical flow.

Model	MAE	RMSE	R^2
ConvLSTM	151.1	199.2	0.31
Diffusion	138.3	191.9	0.36
Optical Flow	155.1	210.7	0.23
Persistence	184.6	237.2	0.03

Table 5.8: Performance during large swings

Finally, as presented in Section 5.2.3, there were some performance improvements to be attained when introducing some synthetic samples during training the CNN. We quantify this phenomenon for the more interesting subset of the data (sequences with drops or jumps). For sake of simplicity, we just analyze the models found on the Pareto front, where optimal trade-offs between performance metrics are found. The results, shown in Table 5.9, indicate that incorporating synthetic data into the training process indeed leads to marginal gains in the model's overall performance. When a small proportion of synthetic data was introduced (15%), the model exhibited an improvement of around 2.5% in terms of MAE. As we increased the percentage of generated images even further (50%), the model continued to show improvements, though to a smaller degree. In conclusion, acquainting the model with synthetic data when we train it is beneficial, since the images that the model will see during inference are synthetic. Nevertheless, we quickly reach a point of diminishing returns, in which adding more and more generated images does not actually boost performance. As we have already demonstrated, when the share of synthetic data becomes too large the performance of the model is notably weak (recall Fig. 5.20).

Model	MAE	RMSE	R^2
Diffusion	138.3	191.9	0.36
Diffusion 15%	135.8	182.1	0.42
Diffusion 50%	134.5	176.6	0.46

Table 5.9: Performance of DM-CNN pipeline with different fractions of synthetic data during training

Overall, the proposed diffusion pipeline consistently achieves the best performance across all evaluations, matched only by the persistence model when considering the entire dataset. However, when we focus on sequences with sharp drops or jumps, which are the primary concern of this project, any predictive model outperforms persistence.

5.3.2. Probabilistic predictions

In this section, we present a probabilistic framework designed to leverage the stochasticity of diffusion models for the irradiance prediction task. We aim to use the variations in the DM outputs to forecast a probability distribution instead of point predictions. The framework involves constructing a mixture model that combines prior knowledge with a data-driven component.
For a given input (i.e. an image), we can run the diffusion model multiple times to obtain a set of stochastic predictions:

$$\{\hat{\rho}_i\}_{i=1}^N$$

where:

- $\hat{\rho}_i$ is the i-th prediction for the input. In particular we predict the CSI, which can be easily converted to irradiance by multiplying it by the clear sky model irradiance.
- N is the total number of predictions generated.

We propose a mixture model that combines a prior distribution $P_0(\rho)$ with a data-driven component based on the stochastic predictions:

$$P(\rho) = a_N \cdot P_0(\rho) + (1 - a_N) \cdot \frac{1}{N} \sum_{i=1}^{N} \mathcal{N}\left(\rho \mid \hat{\rho}_i, \sigma_n^2\right) \tag{5.1}$$

where:

- $a_N \in [0, 1]$ is the mixing coefficient.
- $P_0(\rho)$ is the prior distribution of the target variable r.
- $\mathcal{N}\left(\rho \mid \hat{\rho}_i, \sigma_n^2\right)$ is a Gaussian distribution centered at $\hat{\rho}_i$ with variance σ_n^2.
- σ_n^2 represents the variance (uncertainty) associated with each prediction.

The prior P_0, represents existing knowledge about the distribution of ρ. We opt for a relatively straightforward choice, and parametrize it using a normalized histogram of the dataset (see Fig. 5.23). Then each prediction $\{\hat{\rho}_i\}$ is associated with a Gaussian distribution centered at the prediction itself and with variance σ_n^2. The mixing coefficient a_N balances the influence between the prior and the data-driven component. A higher a_N places more weight on the prior, indicating that the model predictions do not provide significant new information, while a lower a_N emphasizes the data-driven component, which means that the model predictions are informative.

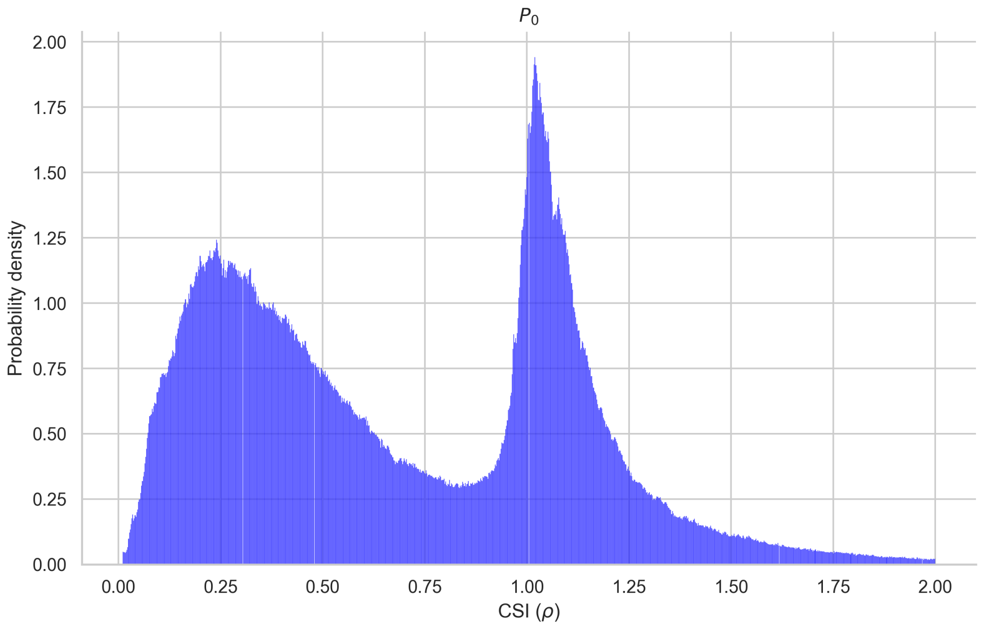


Figure 5.23: Histogram-based Prior

To fully specify the mixture model, we need to determine the values of a_N and σ_n^2. We achieve this by maximizing the likelyhood of the observed data under the model (minimizing the Negative Log-Likelyhood (NLL)). We use the numerical optimization toolbox provided by the scientific computing library SciPy [97]. To gain a deeper understanding of our probabilistic framework, we plot the NLL surface, as shown in Fig. 5.24. The surface shows a U-shape along the σ_n^2 axis, for very small values of σ_n^2 the NLL is relatively high, but as σ_n^2 increases the NLL improves until it reaches an optimal point. Beyond this optimal point, the NLL begins to increase again as σ_n^2 becomes too large. There is an intermediate value of σ_n^2 that balances the uncertainty in the model, resulting in the best fir to the data. As for a_N, when this parameter decreases from 1 towards 0, we see that the NLL improves, which means that the model performs better when the data-driven component is weighted more heavily than the prior. However, as a_N becomes too small, especially in combination with a very small σ_n^2, the NLL begins to increase sharply. Relying too much o the stochastic predictions without enough variance to capture the uncertainty leads to poor performance. It must be said that each variation of the framework has a slightly different NLL surface, hence each configuration needs to be optimized separately.

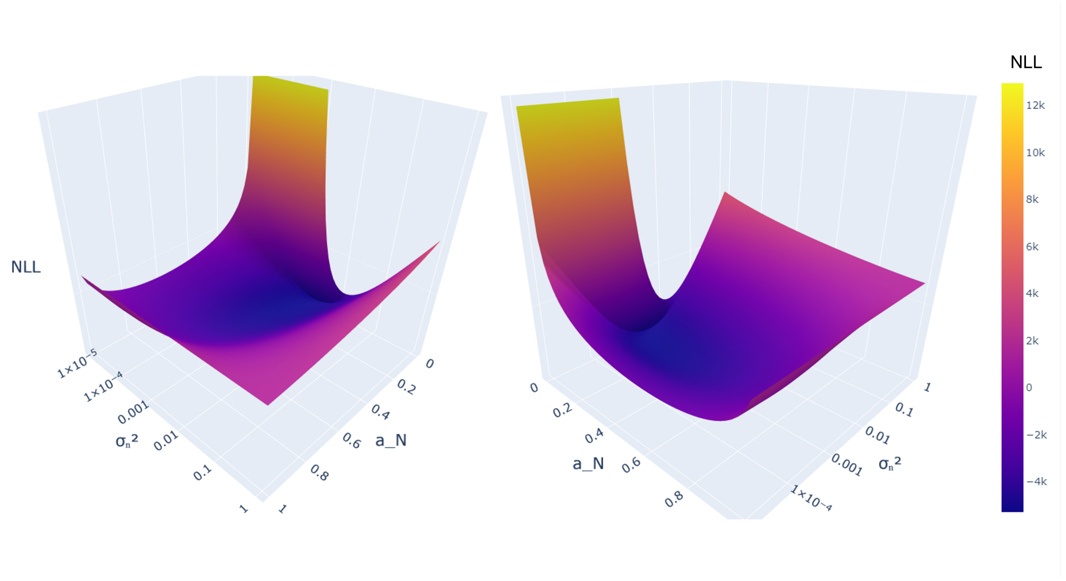


Figure 5.24: NLL surface for probabilistic model tuning

Once the model is fully tuned we can use it to predict probability distributions instead of point estimates. The process is as follows. Given a conditioning sequence we first inference the DM repeatedly (N times) to generate a set of possible future frames. We run each of the generated frames through the CNN, which yields a set of irradiance predictions. Using Eq. 5.1 and the optimal parameters, we can compute the final probability function. Some samples can be seen in Fig. 5.26 and Fig. 5.26.

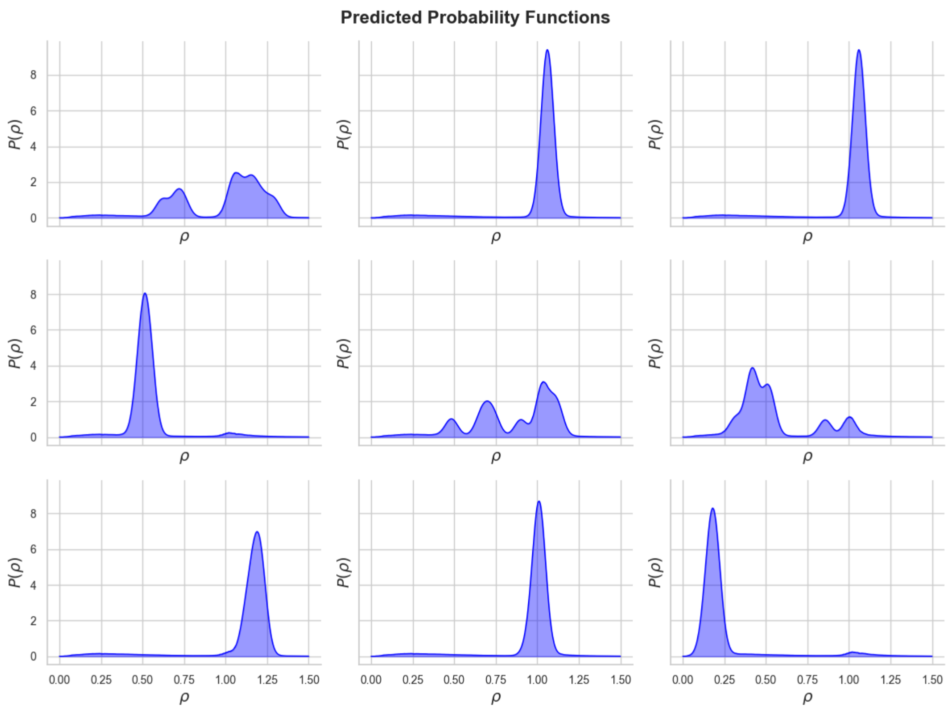


Figure 5.25: Different probabilistic predictions

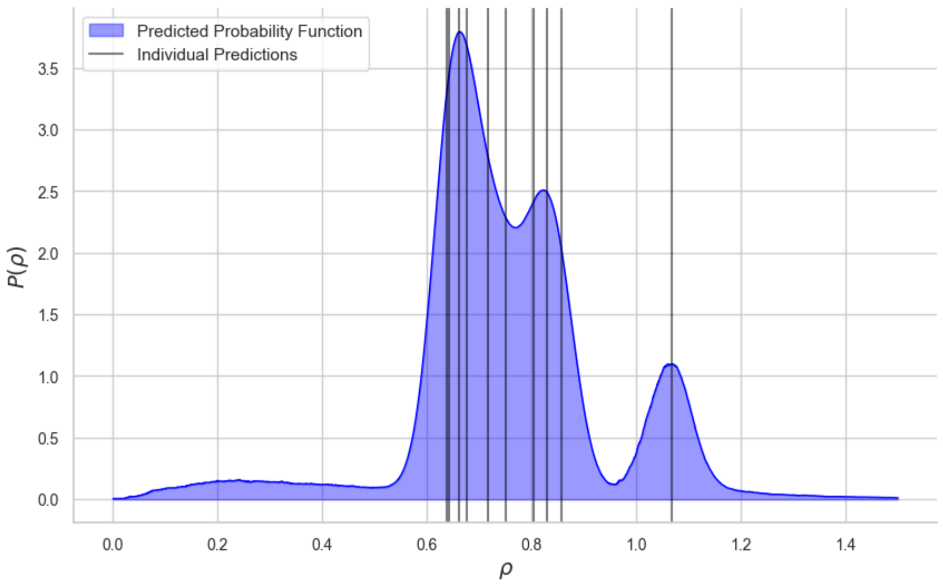


Figure 5.26: Example of probabilistic prediction with individual predictions

We conduct an experiment to evaluate how varying the number of stochastic samples affects the performance of our probabilistic framework. The experiments were performed using the diffusion model and the optical flow model. We find the optimal parameters for each model a_N and σ_n^2 by minimizing the NLL as described before.

Model	N	Optimal a_N	Optimal σ_n^2
Diffusion Model	1	0.1657	0.00283
Diffusion Model	5	0.1335	0.00195
Diffusion Model	10	0.1273	0.00145
Diffusion Model (Median)	1	0.1624	0.00253
Optical Flow Model	1	0.1795	0.00298

Table 5.10: Probabilistic model tuning results

As can bee observed, as N increases from 1 to 10 in the diffusion model, the optimal mixing coefficient a_N decreases. This implies that the model relies less on the prior distribution P_0 and more on the data-driven component, derived from the stochastic predictions. Similarly, when the number of samples increases, the optimal variance σ_n^2 decreases, indicating that the model requires narrower distributions if there are more samples. Among the models with a single sample, we can see how the diffusion with median has the least reliance on the prior, as well as the lowest σ_n^2.
The optimized models are tested by means of the NLL on an unobserved dataset. The results, shown in table 5.11, indicate that the models with the least reliance on the prior also perform comparatively better. Therefore, the diffusion model with 10 stochastic samples outperforms the rest, followed by the diffusion model with 5 samples and the diffusion model with the median. All the models perform significantly better than just using the prior. It has to be said that the better performance comes at the cost of additional computational resources. Generating a single sample takes 1.5 seconds in the available hardware (Nvidia Quadro P5000 GPU with 16 GB memory), hence gefnerating a single prediction with the best performing model takes around 15 seconds.

Model	N	NLL	NLL per sample
Diffusion Model	1	-4366.16	-0.873
Diffusion Model	5	-5050.08	-1.01
Diffusion Model	10	-5376.05	-1.07
Optical Flow Model	1	-4110.72	-0.822
Diffusion Model (Median)	1	-4626.61	-0.925
Prior P_0	-	1710.42	0.342

Table 5.11: Probabilistic model performance results

Another interesting metric to analyze is the entropy (H). The entropy measures the uncertainty in the model's predictions. Unlike the NLL, which directly evaluates how well the model fits the observed data, the entropy simply quantifies how confident the model is about its predictions. The entropy for a continuous probability distribution with probability density function $p(x)$ over a domain D is calculated with Eq. 5.2, and has units of nats if we use the natural logarithm.

$$H(X) = -\int_D p(x)\log(p(x))\,dx \tag{5.2}$$

It is important to emphasize that a low entropy value indicates the model is highly confident in its predictions, not necessarily that those predictions are correct. A model can be confidently wrong, meaning it may produce low-entropy predictions that are consistently inaccurate. Fig. 5.27 shows predictions with different levels of entropy.

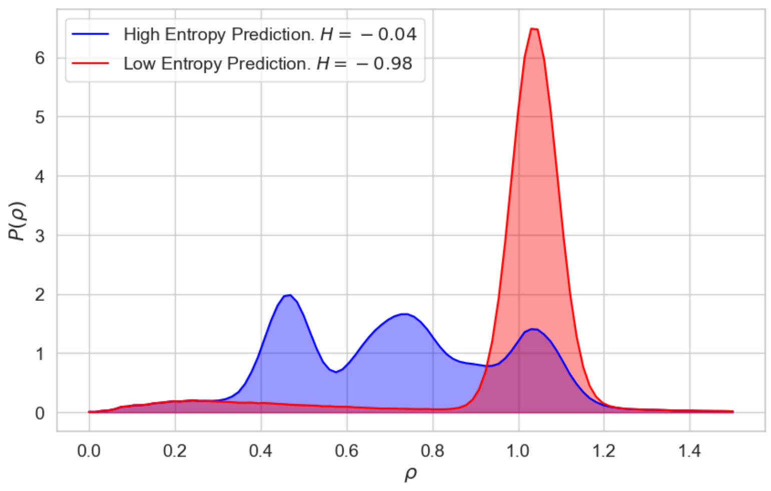


Figure 5.27: Example confident and uncertain predictions

In this context, we evaluate the average entropy of the different models. As shown in Table 5.12, the diffusion model with 10 stochastic samples has the highest entropy. This suggests that while this model fits the data better (as indicated by the NLL), it is less confident in its predictions, producing more spread out distributions. On the other hand, the diffusion model with 5 samples achieves the lowest entropy indicating that it generates narrower distributions. All the models with a single sample lie somewhere in between. To further contextualize these entropy results, it's useful to compare them with the entropy of a uniform distribution, which is 0.406, and the entropy of the prior distribution, which is 0.22. The fact that all model entropies are lower than these baselines reflects that the models are more certain in their predictions than a random guess (uniform distribution) or prior-only assumption.

Model	N	H [nats]
Diffusion Model	1	-0.91
Diffusion Model	5	-1.03
Diffusion Model	10	-0.74
Diffusion Model (Median)	1	-0.96
Optical Flow Model	1	-0.85
Uniform distribution [0, 1.5]	-	0.406
Prior	-	0.228

Table 5.12: Average entropy of different models

In conclusion, the models that rely less on the prior (i.e. those with more samples) tend to perform better in terms of NLL, meaning they are overall more accurate. However, the relationship between the number of samples and entropy is less straightforward. In fact, while more samples lead to better accuracy, they do not always results in higher confidence. The diffusion model with 5 stochastic samples seems to strike the best balance between accuracy and confidence while requiring half the computational resources of the 10-sample model, making it a strong candidate for real-world applications.

6 Discussion and Conclusion

6.1. Answers to Research Questions

- What are current state-of-the-art techniques in weather forecasting using ML and how do they compare to traditional methods?

Some of the current state-of-the art techniques in ML-driven weather forecasting involve complex models such as ViTs, GNNs or DMs, often with parameter counts in the hundreds of millions or billions. These models have demonstrated improvements over traditional NWP methods in terms of accuracy, but the most important advantage is their significantly lower computational requirements. While these ML models can be very challenging and resource-intensive to train, once trained, they allow for relatively cheap and fast inference. Over the last few years the field has experienced a surge in interest, and the body of research is growing rapidly. However, our particular use case, characterized by short term predictions with high spatial resolution was found to be relatively unexplored.

- Do latent diffusion models improve the accuracy of predicting short-term irradiance? How do they perform in stable and challenging conditions?

Diffusion models proved to perform very well in the task of short-term irradiance prediction. One highlight is their competitiveness in the natural dataset, heavily biased towards persistence sequences. Even then, DMs were not only able to match persistence models but to improve upon them by a small margin. When it comes to sequences with fluctuations, again diffusion models proved to be the best amongst the benchmarked models.

- How to design a diffusion model for the task of irradiance nowcasting? How to use past observations to condition the denoising process?

DMs were originally designed for image generation rather than regression tasks, hence they cannot be employed as a standalone model for irradiance nowcasting. To bridge this gap, we employ a two-step pipeline: first, the DM acts as a next-frame predictor that generates a future sky configuration based on past observations; second, we use a CNN to compute the irradiance from these predicted frames. As for the conditioning information to guide the denoising process, there exist a few options depending on the particular architecture of the U-Net and the intended application. We employed a relatively straightforward approach by concatenating all past images with the noisy sample in the input.

- Can we inference the model recursively to arbitrarily extend the prediction horizon? How does this affect its predictive performance?

We can recursively use the model's own predictions to extend the prediction horizon; however, this approach leads to a progressive deterioration of the generated samples. Fine details, like cloud edges and textures become less defined, and the position and shape of clouds increasingly diverge from reality. Therefore, while recursive inference is possible, it will progressively deteriorate the model's predictive accuracy.

- How does the inherent uncertainty of the denoising process affect the results? Can it be leveraged to quantify uncertainty in the model predictions?

The inherent uncertainty of the denoising process was found to be a valuable feature of DMs. It allowed us to develop a probabilistic framework capable of producing full probability distributions instead of point predictions. This is particularly beneficial in power system operation, where understanding the range of possible future scenarios and being able to quantify the uncertainty associated with each prediction is crucial to make informed decisions.

6.2. Discussion of Results

The first critical observation comes from the poor performance of the LDM. LDMs have been extremely successful in text-to-image pipelines aimed at generating high resolution outputs given a prompt. However, the application of these models to produce relatively low-resolution images that could be efficiently handled by a CNN did not yield satisfactory results. The LDMs managed to capture the general idea of the dataset, including the location-specific masks and the most represented shades, but failed to provide detailed cloud contours and textures, essential to assess the level of irradiance. As we have discussed, we believe that compressing already small images (128x128 pixels) with a VAE resulted in a latent space that was too compact, limiting the amount of information that it can possibly encode. Therefore, the model was only able to capture the general structure of images but not intricate details. This underperformance led to a shift in the focus of this project leading us to pixel space DMs, which operate directly on image pixels without relying on latent spaces.
In contrast to LDMs, pixel space DMs were much more effective for our task. The model produced images that were sometimes indistinguishable from the ground truth. When comparing our next frame predictor with other relevant methods, we found that the MSE was not the most suitable metric to quantify performance given our use case. The MSE was heavily influence by slight differences in the brightness and color of the images, which were sometimes not visually apparent. The SSIM on the other hand, was a more reliable benchmark, that could accurately quantify whether two images were similar in terms of the position and shapes of clouds. The DM consistently outperformed the other models and showed various desirable features. In dynamic scenarios, with rapidly moving clouds, the DM accurately mapped future cloud positions and outlines, while the other models struggled either by warping previous frames too much or by producing blurry noncommittal predictions. Furthermore, the DM's ability to conceal or reveal the sun was unique among the models, highlighting its peculiar potential for this application.
Regarding the image processing CNN, we found that feature engineering contributed significantly to the performance of the network. In particular, giving the network the previous irradiance resulted in a major performance boost. The rest of the features, such as the binary mask and datetime information, also were beneficial, although to a smaller degree. Interestingly, increasing the complexity of this network beyond a certain point did not provide any benefits, as illustrated by the fact that simple Conv-Pooling architectures outperformed much larger ResNets. Finally, we found that including some generated frames during training improved the network's performance on synthetic data (the actual use case). However, as the proportion of synthetic data increased beyond a certain threshold, the accuracy of the network collapsed.
Moving on to the case studies, in which we evaluate the full pipeline, we first focus on point predictions. When comparing our method with other relevant benchmarks we find several important patterns. The persistence model shows great overall performance, achieving high RMSE and R^2 scores. This results was somewhat expected, as persistence is very effective on most of the dataset. However, this good performance should not be overemphasized, as persistent situations do not challenge grid stability significantly. When we move toward more dynamic situations, persistence falls short. In such situations the diffusion model performs best, highlighting its superior ability to forecast complex scenarios. As expected given the findings in Section 5.2.3, we can push the performance of our model a few percentage points further by incorporating generated images to the training dataset. However, we could likely improve the performance of the other models, ConvLSTM and Optical Flow, if we trained their CNN components on images generated by those models. It's reasonable to expect that their performance could achieve similar gains as the ones obtained by diffusion.
Regarding the second case study, we leverage the inherent stochasticity in diffusion inference to develop a framework capable of predicting full probability distributions rather than single-point predictions.

The framework uses a mixture model to combine prior knowledge with data-driven predictions. We balance the two components based on the mixing coefficient a_N. The results show that as the number of stochastic samples increases, the optimal a_N becomes smaller, indicating that the model increasingly relies on it's own predictions rather than prior knowledge. Similarly, the variance, which represents the uncertainty associated with each prediction, decreases with more samples. In terms of performance, measured by the NLL, the diffusion model with 10 samples outperformed all other configurations. However this comes at the cost increased computational time.
In summary, we think our proposed pipeline is very effective for the task of irradiance nowcasting. In the presented configuration, with a prediction horizon of one minute and three conditioning frames, our method outperforms traditional computer vision algorithms and alternative ML-driven methods. While the persistence model excels in stable conditions, it struggles with sequences that contain fluctuations, which actually challenge grid stability. Our pipeline matches the performance of persistence in the overall dataset, while also displaying promising results in dynamic sequences. In addition to the strong performance in point-predictions, our pipeline is capable of predicting full probability distribution, a very valuable feature for real-world applications.

6.3. Limitations and Future Work

Throughout the course of this research, we came up with several ideas that could enhance the performance and applicability of our models. Some of these possibilities were identified at the beginning, while others only became apparent as we progressed. In this section, we discuss these ideas, acknowledge the limitations of our method and provide insights and recommendations for future work.
As seen in Section 5.1.4, when evaluating the performance of the Next Frame Predictor, we observed that the SSIM provided a more representative assessment of image quality compared to the MSE. The MSE tends to be heavily influenced by minor variations in brightness and color, which may not significantly impact the quality of the image in terms of using it to derive the irradiance present. An interesting direction for future work would be to train the network using SSIM as the loss function, or perhaps a combination of SSIM and MSE. Incorporating metrics like SSIM into the loss function could lead to improved visual quality in the predicted frames, and alleviate the issue of blurriness in the output images. As we have discussed, models trained on MSE sometimes learn to ”hack” this metric by producing washed-out predictions that are never drastically wrong but do not contain any useful information. By including the SSIM, the model might produce sharper and more detailed predictions that are more informative to derive irradiance.
For a final application, we suggest a comprehensive hyperparameter search, since ours was limited given the timeline of the project and the large computational demands of training DMs. Parameters such as learning rates, network architectures, and regularization techniques can have a big impact on model performance. We therefore believe that we did not reach the full potential of our proposed pipeline. There exist frameworks that automate this task using efficient algorithms, and smart features like pruning of unpromising trials, that can significantly reduce the time required to find the best possible configuration (e.g. Optuna).
Our implementation conditions the next frame predictor DM by stacking the past sequence of images along the channel axis of the noisy sample. This method allows the network to access all conditioning information at the initial layer, but this might not be enough to fully capture all the temporal information present in the sequence. Exploring more sophisticated conditioning mechanisms could improve the model's ability to capture these dependencies. For instance, some of the state-of-the-art text-to-image models employ a mechanism called cross-attention, where the query from one sequence attends to key-value pairs from a different sequence, and they give the conditioning information at several points throughout the U-Net. Using this, or developing custom conditioning strategies tailored to our specific application could also be a fruitful research direction.
In our experiments, we use three previous images separated by the same time interval that we aim to predict ahead: 1 minute. This however, is just one of many possible choices. Future studies could explore alternative frame spacing strategies, like including more conditioning images or varying the intervals between them. For example, we can think of a logarithmic spacing scheme, which would provide a few frames from the distant past and increasingly more frames closer to the present, potentially offering a richer temporal context for the network to model complex cloud patterns.
Although we performed recursive inference of the model, time constraints limited our ability to fully

explore this path. Recursive inference involves using the model's own predictions as inputs for subsequent predictions, such that the prediction horizon can be arbitrarily extended. This is in theory possible, but as we demonstrated, compounding errors make the recursive samples degrade quickly, especially in dynamic situations. A deeper exploration of this topic would be valuable. For instance, it might be interesting to train models specifically designed to predict further into the future and compare their performance with predictions generated recursively. Another promising direction could involve developing a model that generates not just a single frame but two frames: one representing the near future and another for a slightly later time point. This could potentially slow down the error accumulation, mitigating the rapid degradation of recursive samples.

In our implementation, the Next Frame Predictor and the irradiance prediction model are trained separately. We consider that a very promising area of research could involve training both models jointly, which would allow the Next Frame Predictor to receive feedback from the irradiance prediction loss, potentially leading to generated frames that are more informative for the downstream task. We think this end-to-end training approach, while technically challenging, could align the objectives of both models, resulting in improved performance.

A big challenge and an ongoing concern throughout this work is the fact that most of the time irradiance changes are smooth, which contributes to the strong performance of the persistence model. Early exploratory data analytics showed that less than 10% of the data contained sequences with fluctuating irradiance levels. Future work could focus on training models exclusively on dynamic sequences where irradiance changes are identified. An upstream gating mechanism could be developed to classify situations as either ”persistence” or ”dynamic.” In cases classified as ”persistence,” the system would rely on the persistence model, while in ”dynamic” cases, the specialized models would be employed. This approach could improve prediction accuracy by adapting the model to fit the specific characteristics of each situation.

All of the above highlights the wide range of opportunities for improving the models developed in this work, which could lead to significant advancements in their accuracy and applicability.

References

[1] F. J. Nijsse, J. F. Mercure, N. Ameli, *et al.*, "The momentum of the solar energy transition," *Nature Communications*, vol. 14, 1 Dec. 2023, ISSN: 20411723. DOI: 10.1038/s41467-023-41971-7.

[2] J. Yin, A. Molini, and A. Porporato, "Impacts of solar intermittency on future photovoltaic reliability," *Nature Communications*, vol. 11, 1 Dec. 2020, ISSN: 20411723. DOI: 10.1038/s41467-020-18602-6.

[3] B. Martins, A. Cerentini, S. L. Mantelli, *et al.*, "Systematic review of nowcasting approaches for solar energy production based upon ground-based cloud imaging," *Solar Energy Advances*, vol. 2, p. 100 019, Jul. 2022. DOI: 10.1016/j.seja.2022.100019.

[4] R. Marquez and C. F. Coimbra, "Forecasting of global and direct solar irradiance using stochastic learning methods, ground experiments and the nws database," *Solar Energy*, vol. 85, no. 5, pp. 746–756, 2011, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2011.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X11000193.

[5] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," May 2021. [Online]. Available: http://arxiv.org/abs/2105.05233.

[6] X. Li, Y. Ren, X. Jin, *et al.*, *Diffusion models for image restoration and enhancement – a comprehensive survey*, 2023. arXiv: 2308.09388 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2308.09388.

[7] T. Höppe, A. Mehrjou, S. Bauer, D. Nielsen, and A. Dittadi, *Diffusion models for video prediction and infilling*, 2022. arXiv: 2206.07696 [cs.CV].

[8] J. Coiffier, *Fundamentals of Numerical Weather Prediction*. Cambridge University Press, 2011.

[9] "Part iii: Dynamics and numerical procedures," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221949988.

[10] S. A. Ackerman and J. Martin, *What is the best weather forecast model?* Accessed: 2024-03-24, Mar. 2019. [Online]. Available: https://wxguys.ssec.wisc.edu/2019/03/04/models/.

[11] F. Bouttier and P. Courtier, "Data assimilation concepts and methods," Mar. 1999. [Online]. Available: https://www.ecmwf.int/sites/default/files/elibrary/2002/16928-data-assimilation-concepts-and-methods.pdf.

[12] M. K. Y. Madalina Surcel Isztar Zawadzki, "A study on the scale dependence of the predictability of precipitation patterns," *Journal of the Atmospheric Sciences*, vol. 72, pp. 216–235, Jan. 2015.

[13] R. Buizza, "Chaos and weather prediction," Jan. 2000. [Online]. Available: https://www.ecmwf.int/sites/default/files/elibrary/2002/16927-chaos-and-weather-prediction.pdf.

[14] T. Palmer, "Predicting uncertainty in forecasts of weather and climate," Nov. 1999. [Online]. Available: https://www.ecmwf.int/sites/default/files/elibrary/2003/16965-predicting-uncertainty-forecasts-weather-and-climate.pdf.

[15] M. Hasenbalg, P. Kuhn, S. Wilbert, B. Nouri, and A. Kazantzidis, "Benchmarking of six cloud segmentation algorithms for ground-based all-sky imagers," *Solar Energy*, vol. 201, pp. 596–614, 2020, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2020.02.042. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X2030147X.

[16] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, J. Bigun and T. Gustavsson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370, ISBN: 978-3-540-45103-7.

[17] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81, Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[18] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[19] L. Martín, L. F. Zarzalejo, J. Polo, A. Navarro, R. Marchante, and M. Cony, “Prediction of global solar irradiance based on time series analysis: Application to solar thermal power plants energy production planning,” *Solar Energy*, vol. 84, no. 10, pp. 1772–1781, 2010, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2010.07.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X10002379.

[20] W. Liu, Y. Liu, X. Zhou, *et al.*, “Use of physics to improve solar forecast: Physics-informed persistence models for simultaneously forecasting ghi, dni, and dhi,” *Solar Energy*, vol. 215, pp. 252–265, 2021, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2020.12.045. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X20313049.

[21] W. Ji and K. C. Chee, “Prediction of hourly solar radiation using a novel hybrid model of arma and tdnn,” *Solar Energy*, vol. 85, no. 5, pp. 808–817, 2011, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2011.01.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X11000259.

[22] E. Stellwagen and L. Tashman, “Arima: The models of box and jenkins,” *Foresight: Int. J. Appl. Forecast.*, pp. 28–33, Jan. 2013.

[23] S. J. D. Prince, *Understanding Deep Learning*. MIT Press, 2023. [Online]. Available: http://udlbook.com.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[25] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Education, 2021, ISBN: 9781292401171. [Online]. Available: https://books.google.ch/books?id=cb0qEAAAQBAJ.

[26] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2018. arXiv: 1603.07285 [stat.ML].

[27] E. J. Kirkland, “Bilinear interpolation,” in *Advanced Computing in Electron Microscopy*. Boston, MA: Springer US, 2010, pp. 261–263, ISBN: 978-1-4419-6533-2. DOI: 10.1007/978-1-4419-6533-2_12. [Online]. Available: https://doi.org/10.1007/978-1-4419-6533-2_12.

[28] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Jan. 2010.

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[30] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG].

[31] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473.

[32] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” Jun. 2017, pp. 6000–6010, ISBN: 9781510860964. [Online]. Available: http://arxiv.org/abs/1706.03762.

[33] Z. Niu, G. Zhong, and H. Yu, “A review on the attention mechanism of deep learning,” *Neurocomputing*, vol. 452, pp. 48–62, 2021, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.03.091. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122100477X.

[34] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” Oct. 2021. [Online]. Available: http://arxiv.org/abs/2010.11929.

[35] F. Hill, *Deep learning for language understanding*, UCL x DeepMind Lecture Series, 2020. [Online]. Available: https://storage.googleapis.com/deepmind-media/UCLxDeepMind_2020/L7%20-%20UCLxDeepMind%20DL2020.pdf.

[36] R. E. Turner, C.-D. Diaconu, S. Markou, A. Shysheya, A. Y. K. Foong, and B. Mlodozeniec, "Denoising diffusion probabilistic models in six simple steps," Feb. 2024. [Online]. Available: http://arxiv.org/abs/2402.04384.

[37] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, 2015. arXiv: 1503.03585 [cs.LG].

[38] L. Weng, "What are diffusion models?" *lilianweng.github.io*, Jul. 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[39] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," Jun. 2020, ISBN: 9781713829546. [Online]. Available: http://arxiv.org/abs/2006.11239.

[40] R. O'Connor, "Introduction to diffusion models for machine learning," May 2022. [Online]. Available: https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/.

[41] E. Hoogeboom, J. Heek, and T. Salimans, "Simple diffusion: End-to-end diffusion for high resolution images," Jan. 2023. [Online]. Available: http://arxiv.org/abs/2301.11093.

[42] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," Feb. 2021, pp. 8162–8171. [Online]. Available: http://arxiv.org/abs/2102.09672%20https://proceedings.mlr.press/v139/nichol21a.html.

[43] J. Song, C. Meng, and S. Ermon, *Denoising diffusion implicit models*, 2022. arXiv: 2010.02502 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2010.02502.

[44] A. Nichol, P. Dhariwal, A. Ramesh, *et al.*, *Glide: Towards photorealistic image generation and editing with text-guided diffusion models*, 2022. arXiv: 2112.10741 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2112.10741.

[45] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, 2022. arXiv: 2204.06125 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2204.06125.

[46] C. Saharia, W. Chan, S. Saxena, *et al.*, *Photorealistic text-to-image diffusion models with deep language understanding*, 2022. arXiv: 2205.11487 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2205.11487.

[47] W. Peebles and S. Xie, *Scalable diffusion models with transformers*, 2023. arXiv: 2212.09748 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2212.09748.

[48] A. Hatamizadeh, J. Song, G. Liu, J. Kautz, and A. Vahdat, *Diffit: Diffusion vision transformers for image generation*, 2024. arXiv: 2312.02139 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2312.02139.

[49] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," Dec. 2021. [Online]. Available: http://arxiv.org/abs/2112.10752.

[50] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, 2021. arXiv: 2003.05991 [cs.LG].

[51] D. Birla, *Basics of autoencoders*, Accessed: 2024-03-22, Mar. 2019. [Online]. Available: https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f.

[52] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," Dec. 2014. [Online]. Available: http://arxiv.org/abs/1312.6114.

[53] S. Patel, "All you need to know about variational autoencoder," *BayesLabs Blog*, Jun. 2019. [Online]. Available: https://blog.bayeslabs.co/2019/06/04/All-you-need-to-know-about-Vae.html.

[54] S. Paul, *Autoencoderkl*, Accessed: 2024-03-22, 2024. [Online]. Available: https://github.com/huggingface/diffusers/blob/main/docs/source/en/api/models/autoencoderkl.md.

[55] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," May 2015. [Online]. Available: http://arxiv.org/abs/1505.04597.

[56] A. Smets, K. Jäger, O. Isabella, R. Van Swaaij, and M. Zeman, *Solar Energy - The physics and engineering of photovoltaic conversion, technologies and systems*. Feb. 2016, ISBN: 9781906860325.

[57] J. A. Duffie, W. A. Beckman, and N. H. Blair, *Solar Radiation*. John Wiley & Sons, Ltd, 2013, ch. 1, pp. 3–42, ISBN: 9781118671603. DOI: `https://doi.org/10.1002/9781118671603.ch1`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118671603.ch1`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118671603.ch1`.

[58] L. Toreti Scarabelot, G. Arns Rampinelli, and C. R. Rambo, "Overirradiance effect on the electrical performance of photovoltaic systems of different inverter sizing factors," *Solar Energy*, vol. 225, pp. 561–568, 2021, ISSN: 0038-092X. DOI: `https://doi.org/10.1016/j.solener.2021.07.055`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0038092X21006253`.

[59] P. Ineichen and R. Perez, "A new airmass independent formulation for the linke turbidity coefficient," *Solar Energy*, vol. 73, no. 3, pp. 151–157, 2002, ISSN: 0038-092X. DOI: `https://doi.org/10.1016/S0038-092X(02)00045-2`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0038092X02000452`.

[60] K. Anderson, C. Hansen, W. Holmgren, A. Jensen, M. Mikofski, and A. Driesse, "Pvlib python: 2023 project update," *Journal of Open Source Software*, vol. 8, no. 92, p. 5994, 2023. DOI: `10.21105/joss.05994`.

[61] Z. B. Bouallègue, M. C. A. Clare, L. Magnusson, *et al.*, "The rise of data-driven weather forecasting: A first statistical assessment of machine learning–based weather forecasts in an operational-like context," *Bulletin of the American Meteorological Society*, vol. 105, no. 6, E864–E883, 2024. DOI: `10.1175/BAMS-D-23-0162.1`. [Online]. Available: `https://journals.ametsoc.org/view/journals/bams/105/6/BAMS-D-23-0162.1.xml`.

[62] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Proceedings of the 28th International Conference on Neural Information Processing Systems*, vol. 1, pp. 802–810, Jun. 2015. [Online]. Available: `http://arxiv.org/abs/1506.04214`.

[63] B. Klein, L. Wolf, and Y. Afek, "A dynamic convolutional layer for short rangeweather prediction," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4840–4848. DOI: `10.1109/CVPR.2015.7299117`.

[64] C. K. Sønderby, L. Espeholt, J. Heek, *et al.*, *Metnet: A neural weather model for precipitation forecasting*, 2020. arXiv: `2003.12140 [cs.LG]`.

[65] L. Espeholt, S. Agrawal, C. Sønderby, *et al.*, "Skillful twelve hour precipitation forecasts using large context neural networks," Nov. 2021. [Online]. Available: `http://arxiv.org/abs/2111.07470`.

[66] M. Andrychowicz, L. Espeholt, D. Li, *et al.*, "Deep learning for day forecasts from sparse observations," Jun. 2023. [Online]. Available: `http://arxiv.org/abs/2306.06079`.

[67] V. L. Guen and N. Thome, "Disentangling physical dynamics from unknown factors for unsupervised video prediction," Mar. 2020. [Online]. Available: `http://arxiv.org/abs/2003.01460`.

[68] V. Le Guen and N. Thome, "A deep physical model for solar irradiance forecasting with fisheye images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2020.

[69] S. Ravuri, K. Lenc, M. Willson, *et al.*, "Skilful precipitation nowcasting using deep generative models of radar," *Nature*, vol. 597, pp. 672–677, 7878 Sep. 2021, ISSN: 14764687. DOI: `10.1038/s41586-021-03854-z`.

[70] J. Pathak, S. Subramanian, P. Harrington, *et al.*, "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators," Feb. 2022. [Online]. Available: `http://arxiv.org/abs/2202.11214`.

[71] K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian, "Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast," Nov. 2022. [Online]. Available: `http://arxiv.org/abs/2211.02556`.

[72] K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian, "Accurate medium-range global weather forecasting with 3d neural networks," *Nature*, vol. 619, pp. 533–538, 7970 Jul. 2023, ISSN: 14764687. DOI: `10.1038/s41586-023-06185-3`.

[73] J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro, “Adaptive fourier neural operators: Efficient token mixers for transformers,” Nov. 2021. [Online]. Available: http://arxiv.org/abs/2111.13587.

[74] R. Keisler, *Forecasting global weather with graph neural networks*, 2022. arXiv: 2202.07575 [physics.ao-ph].

[75] R. Lam, A. Sanchez-Gonzalez, M. Willson, *et al.*, “Learning skillful medium-range global weather forecasting,” *Science*, vol. 382, no. 6677, pp. 1416–1421, 2023. DOI: 10.1126/science.adi2336. eprint: https://www.science.org/doi/pdf/10.1126/science.adi2336. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.adi2336.

[76] Y. Hatanaka, Y. Glaser, G. Galgon, G. Torri, and P. Sadowski, “Diffusion models for high-resolution solar forecasts,” Jan. 2023. [Online]. Available: http://arxiv.org/abs/2302.00170.

[77] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, “Cascaded diffusion models for high fidelity image generation,” May 2021. [Online]. Available: http://arxiv.org/abs/2106.15282.

[78] A. Asperti, F. Merizzi, A. Paparella, G. Pedrazzi, M. Angelinelli, and S. Colamonaco, “Precipitation nowcasting with generative diffusion models,” Aug. 2023. [Online]. Available: http://arxiv.org/abs/2308.06733.

[79] J. Leinonen, U. Hamann, D. Nerini, U. Germann, and G. Franch, “Latent diffusion models for generative precipitation nowcasting with accurate uncertainty quantification,” Apr. 2023. [Online]. Available: http://arxiv.org/abs/2304.12891.

[80] D. Yu, X. Li, Y. Ye, *et al.*, “Diffcast: A unified framework via residual diffusion for precipitation nowcasting,” Dec. 2023. [Online]. Available: http://arxiv.org/abs/2312.06734.

[81] Z. Gao, X. Shi, B. Han, *et al.*, “Prediff: Precipitation nowcasting with latent diffusion models,” Jul. 2023. [Online]. Available: http://arxiv.org/abs/2307.10422.

[82] C. Tan, Z. Gao, L. Wu, *et al.*, “Temporal attention unit: Towards efficient spatiotemporal predictive learning,” Jun. 2022. [Online]. Available: http://arxiv.org/abs/2206.12126.

[83] Z. Gao, X. Shi, H. Wang, *et al.*, “Earthformer: Exploring space-time transformers for earth system forecasting,” Jul. 2022. [Online]. Available: http://arxiv.org/abs/2207.05833.

[84] P. E. Debevec and J. Malik, “Recovering high dynamic range radiance maps from photographs,” in *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 1st ed. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3596711.3596779.

[85] Y. Sun, V. Venugopal, and A. R. Brandt, “Short-term solar power forecast with deep learning: Exploring optimal input and output configuration,” *Solar Energy*, vol. 188, pp. 730–741, Aug. 2019, ISSN: 0038092X. DOI: 10.1016/j.solener.2019.06.041.

[86] *Resize*, Torch Contributors. [Online]. Available: https://pytorch.org/vision/main/generated/torchvision.transforms.Resize.html.

[87] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: https://doi.org/10.1145/3065386.

[88] A. Paszke, S. Gross, F. Massa, *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, Version 1.8.0, 2019. DOI: 10.48550/arXiv.1912.01703. [Online]. Available: https://pytorch.org/.

[89] P. von Platen, S. Patil, A. Lozhkov, *et al.*, *Diffusers: State-of-the-art diffusion models*, Version 0.18.0, 2022. [Online]. Available: https://github.com/huggingface/diffusers.

[90] I. Reda and A. Andreas, “Solar position algorithm for solar radiation applications,” *Solar Energy*, vol. 76, no. 5, pp. 577–589, 2004, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2003.12.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X0300450X.

[91] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A flexible technique for accurate omnidirectional camera calibration and structure from motion,” in *Fourth IEEE International Conference on Computer Vision Systems (ICVS’06)*, 2006, pp. 45–45. DOI: 10.1109/ICVS.2006.3.

[92] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A toolbox for easily calibrating omnidirectional cameras,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5695–5701. DOI: 10.1109/IROS.2006.282372.

[93] W. Kabsch, “A solution for the best rotation to relate two sets of vectors,” *Acta Crystallographica Section A*, vol. 32, no. 5, pp. 922–923, Sep. 1976. DOI: 10.1107/S0567739476001873. [Online]. Available: https://doi.org/10.1107/S0567739476001873.

[94] P. Liashchynskyi and P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for nas*, 2019. arXiv: 1912.06059 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.06059.

[95] ndrplz, *Convlstm_pytorch*, https://github.com/ndrplz/ConvLSTM_pytorch, Accessed: July 8, 2024, 2018.

[96] J. Nilsson and T. Akenine-Möller, *Understanding ssim*, 2020. arXiv: 2006.13846 [eess.IV]. [Online]. Available: https://arxiv.org/abs/2006.13846.

[97] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, “SciPy 1.0: Fundamental algorithms for scientific computing in python,” *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: https://doi.org/10.1038/s41592-019-0686-2.