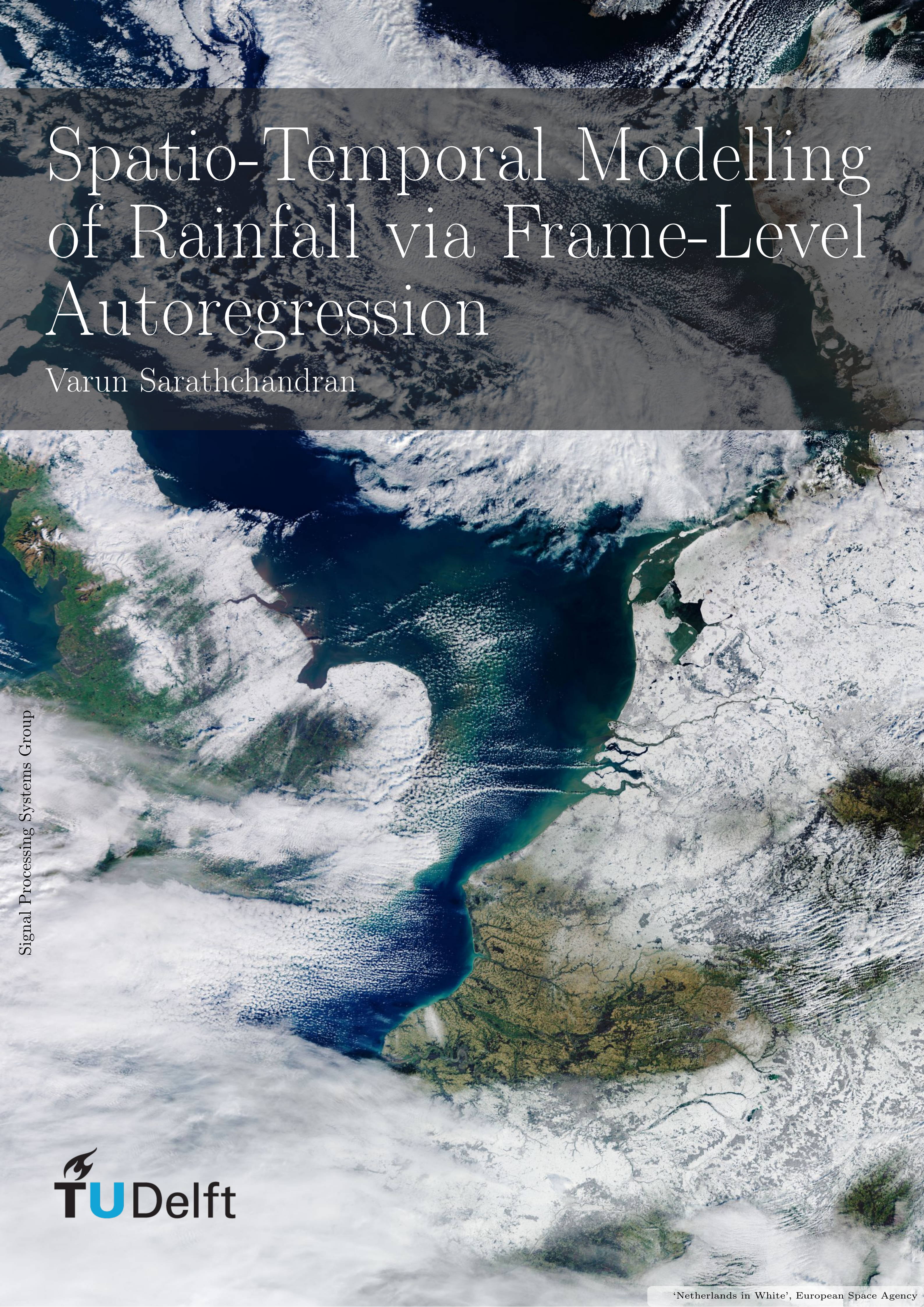


Spatio-Temporal Modelling of Rainfall via Frame-Level Autoregression

Varun Sarathchandran

Signal Processing Systems Group

Signal Processing Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2025-5925282

M.Sc. Thesis

Spatio-Temporal Modelling of Rainfall via Frame-Level Autoregression

Varun Sarathchandran

Abstract

Short-term precipitation forecasting, or nowcasting, plays a vital role in mitigating the impacts of extreme weather by supporting timely decisions in urban planning, flood management, and transportation systems. In this thesis, we cast precipitation nowcasting as a video prediction problem, where the goal is to generate future radar images given a sequence of past observations. To address this, we propose BlockGPT, a generative transformer model that predicts entire frames autoregressively, capturing spatial dependencies within each frame using bidirectional attention, while maintaining temporal causality across frames. BlockGPT is evaluated on two real-world radar datasets: KNMI (Netherlands) and SEVIR (United States), across two forecasting tasks involving different temporal resolutions. The model is benchmarked against existing state-of-the-art approaches-NowcastingGPT and Diffcast-which represent token-based and diffusion-based paradigms, respectively. Our results show that BlockGPT offers competitive performance and strong capabilities in detecting and localizing significant rainfall events, while also enabling faster inference. These properties make it a promising candidate for real-time, threshold-aware weather forecasting systems.

Spatio-Temporal Modelling of Rainfall via Frame-Level Autoregression

Thesis

submitted in partial fulfilment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Varun Sarathchandran
born in Bengaluru, India

This work was performed in:

Signal Processing Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Spatio-Temporal Modelling of Rainfall via Frame-Level Autoregression"** by **Varun Sarathchandran** in partial fulfilment of the requirements for the degree of **Master of Science**.

Dated: 9th July, 2025

Chairman:	prof.dr.ir. Justin Dauwels
Advisor:	ir. Cristian Meo
Committee Members:	dr. Jing Sun

Abstract

Short-term precipitation forecasting, or nowcasting, plays a vital role in mitigating the impacts of extreme weather by supporting timely decisions in urban planning, flood management, and transportation systems. In this thesis, we cast precipitation nowcasting as a video prediction problem, where the goal is to generate future radar images given a sequence of past observations. To address this, we propose BlockGPT, a generative transformer model that predicts entire frames autoregressively, capturing spatial dependencies within each frame using bidirectional attention, while maintaining temporal causality across frames. BlockGPT is evaluated on two real-world radar datasets: KNMI (Netherlands) and SEVIR (United States), across two forecasting tasks involving different temporal resolutions. The model is benchmarked against existing state-of-the-art approaches-NowcastingGPT and Diffcast-which represent token-based and diffusion-based paradigms, respectively. Our results show that BlockGPT offers competitive performance and strong capabilities in detecting and localizing significant rainfall events, while also enabling faster inference. These properties make it a promising candidate for real-time, threshold-aware weather forecasting systems.

Acknowledgments

My acknowledgements run the risk of becoming rather long- my apologies to those who choose to read them in full. I write this not just to thank those central to my thesis, but everyone who played a role throughout my entire MSc journey.

This thesis marked the first time I undertook a long-term individual project. And while, at times, it certainly felt like a solitary effort, most of the time it did not. I've felt like part of a large, interconnected graph, with friends, professors, colleagues, and family as nodes, each offering support, direction, or companionship in their own way. So I'd like to take the time to thank them all.

First and foremost, my deepest gratitude goes to the two people most central to my thesis: Justin and Cristian. Thank you for giving me a platform to explore and build exciting models, while also challenging me to think critically. As supervisors, I really appreciated the balance we struck- one that kept me comfortable, but always nudged me to push further. I must also thank you for the compute capabilities- not once have I had to worry about training time, which is a common roadblock in machine learning-based theses. Your advice, not just on the project but also on my future path, has meant a lot.

I'd also like to thank Jing, from the Computer Science faculty, for kindly agreeing to be on my thesis committee.

To Ankush (whose MSc thesis laid the foundation for mine) thank you for your helpful suggestions and explanations.

To my close friend (and perhaps the one with the most similar research interests), Thomas, thank you for letting me peek into your unconventional but refreshing ideas, and for the countless discussions we've had about our work. They were always a highlight.

I'm also grateful to my MSc friends in the SPS section-Clark, Frank, Nikos, Anjali, Giacomo, Dani, and Anja. Thank you for the laughs, the breaks, and the chats over the last two years. You made the day-to-day so much more enjoyable.

Though it's been almost a year since I last took a course, I want to thank the entire SPS faculty. If I had to sum up my time in the program in a single line: it was challenging, and I had tons of fun. Special thanks to Geert for helping carve out a potential path for my future academic adventures.

To my flatmates here in the Netherlands- thank you for always being there to listen to how my day went, and for sharing how yours did too. Finally, to my family back home, thousands of kilometers away, and some no longer with us, your love and support have transcended both time and space.

Varun Sarathchandran
Delft, The Netherlands
9th July, 2025

Contents

List of Figures

List of Tables

Introduction 1

1.1 What is Nowcasting?

Extreme weather events pose increasing threats to society, with consequences ranging from disruptions to transportation and infrastructure to significant risks to life and property. Flash floods, landslides, and other hazards caused by intense rainfall can result in economic losses and even loss of life. As climate change continues to alter weather patterns and intensify storm systems, the need for accurate, high-resolution short-term weather prediction (known as nowcasting) has become increasingly critical [1].

Nowcasting refers to the prediction of weather phenomena over short time horizons, typically ranging from a few minutes up to six hours. This contrasts with traditional Numerical Weather Prediction (NWP) systems, which rely on solving complex physical equations governing the atmosphere. While NWP models have improved significantly in both resolution and accuracy due to advances in computing power, their latency and coarser granularity still render them less effective for short-term, high-resolution forecasts required during extreme weather events [2].

In this setting, data-driven approaches have emerged as strong alternatives. Borrowing recent advances in computer vision and sequence modelling, these models learn to extrapolate future radar reflectivity or precipitation fields directly from past observations. By treating weather radar imagery as a video prediction task, deep learning-based nowcasting systems aim to capture both spatial and temporal patterns in precipitation dynamics without explicitly relying on physical models [3][4].

Recent state-of-the-art techniques incorporate models such as convolutional Long Short-Term Memory (LSTMs), transformers, and diffusion models, each aiming to overcome challenges such as blurry predictions, poor long-term consistency, or failure to represent extreme events. These models take in as input sequences of radar frames and output pixel-wise forecasts for the immediate future. Evaluating these predictions on both continuous metrics (e.g., MSE (Mean Square Error), MAE (Mean Absolute Error), PCC (Pearson Correlation Coefficient) and categorical metrics (e.g., CSI (Critical Success Index), FAR (False Alarm Rate)) helps assess their quality and usefulness [5].

This thesis focuses on developing and benchmarking deep learning-based now-

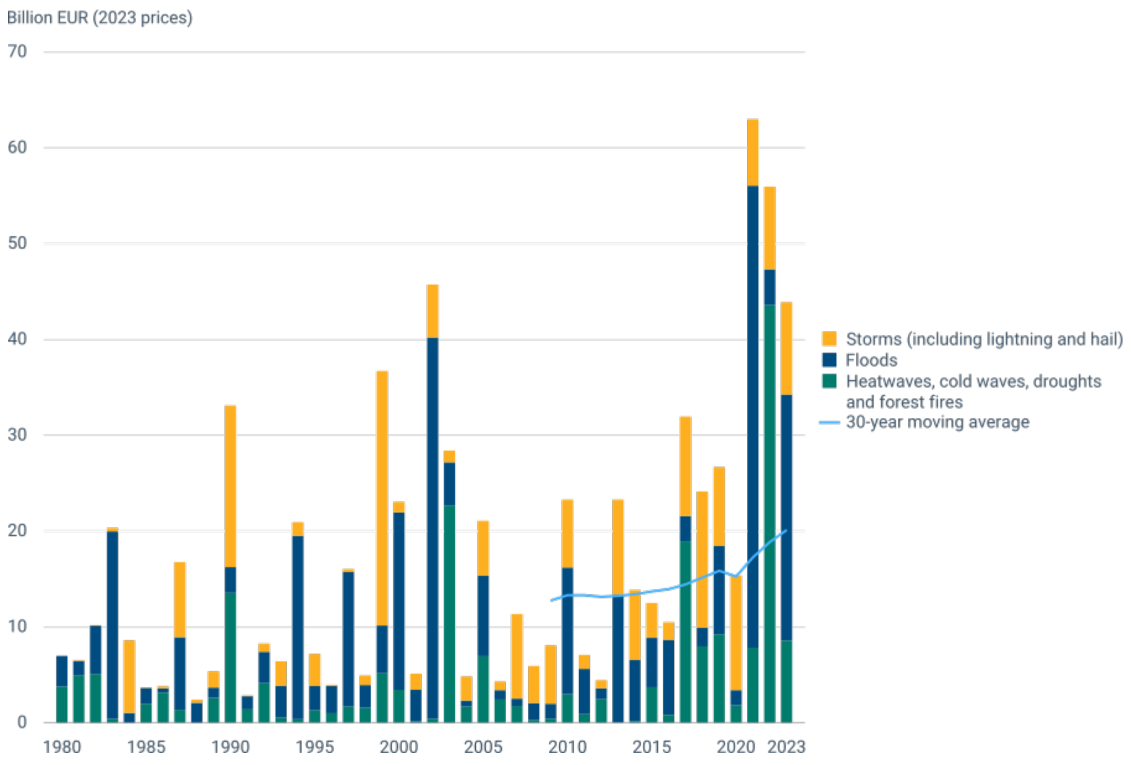


Figure 1.1: Rising trends in economic losses incurred by European nations. The 30-year average, shown in blue, is also on the rise [6]

casting models for short-term precipitation forecasting. Building upon prior work in autoregressive token modelling and generative architectures, we explore novel formulations that combine visual tokenisation and temporal transformers to enhance predictive quality. Particular emphasis is placed on comparing performance across datasets of varying resolution and on evaluating the model's ability to handle extreme precipitation events.

1.2 Why does Nowcasting Matter?

Climate-related disasters are becoming more frequent, intense, and costly. According to the European Environment Agency (EEA) [6], Europe has suffered tens of billions of euros in economic losses every year due to floods, storms, droughts, and wildfires, with statistics visualised in Figure 1.1. These are not isolated incidents; they are part of a growing global challenge driven by climate change and other factors.

Globally, the number of reported extreme weather events has also surged. As shown in Figure 1.2, data from Our World in Data[7] shows a consistent increase in the frequency of natural disasters linked to weather, particularly floods and storms, over the past several decades. This rise not only reflects improved reporting but

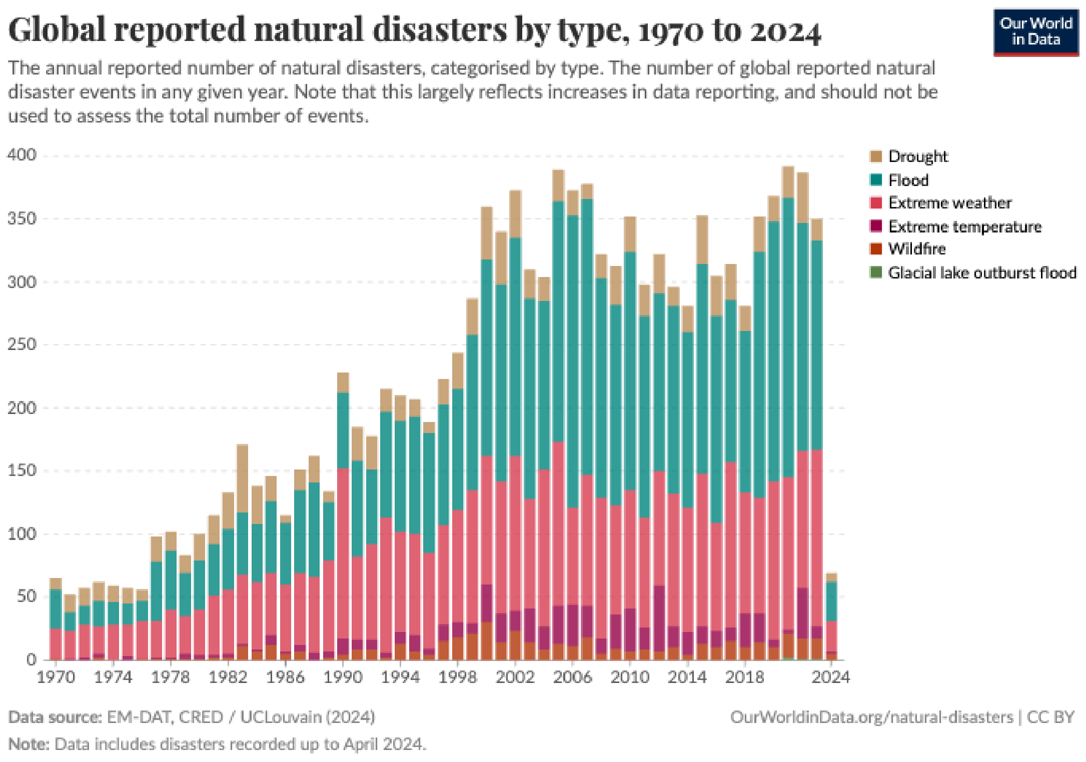


Figure 1.2: Natural disasters caused by extreme weather per year, reported globally [7].

also the increased incidence of climate-induced disasters, which now represent a significant share of all global emergencies.

The human cost of these events is immense. Flash floods, for instance, can overwhelm urban areas in under an hour, leading to destruction of property, loss of life, and long-term displacement of people. In low-lying or densely populated areas, even short bursts of intense rainfall can overwhelm infrastructure and drainage systems, posing severe risks to public safety. According to the Centre for Research on the Epidemiology of Disasters (CRED), over 90% of major disasters in the past 20 years have been caused by weather-related events [8].

In this context, timely and accurate short-term forecasts are no longer a luxury but a necessity. Predicting weather conditions in the immediate future (typically up to 6 hours) forms a critical component of early warning systems. When deployed effectively, nowcasting systems can support disaster response, mobilise emergency services, safeguard critical infrastructure, and ultimately save lives. This thesis contributes to this urgent challenge by developing and evaluating deep learning-based models for precipitation nowcasting. By improving the ability to anticipate high-impact rainfall events with greater accuracy and spatial precision, these systems can enhance preparedness and resilience in the face of climate extremes.

The goal is thus: to reduce the societal and economic burden of severe weather by enabling faster, smarter decisions in critical moments.

1.3 Thesis Outline

The outline of the thesis is as follows. In chapter 2, we elucidate our findings from an in-depth literature review, which briefly explains state-of-the-art machine learning models for weather prediction, as well as where our model is placed in this context. In Chapter 3, the methodology behind our proposed model is then motivated and detailed, both from a high-level as well as from a mathematical viewpoint. We dive deep into the underlying statistics of the datasets used in this study, since understanding the data distribution is fundamental to building good machine learning models, in Chapter 4. The different experiments conducted on the two datasets and their results are contained in Chapter 5. Finally, Chapter 6 presents overall conclusions from the experiments, before closing with ideas for future work in Chapter 7.

Literature Review 2

In this section, we review several recent state-of-the-art models for short-term weather prediction, each contributing distinct methodological advancements to the nowcasting problem. These models also mirror the broader evolution of the video prediction community, transitioning from RNN- and GAN-based approaches to more recent diffusion and transformer-based techniques. One of the earliest deep learning efforts in this domain was by Shi et al. [9], who introduced a translation-invariant variant of ConvRNNs tailored for spatiotemporal forecasting. We then discuss the seminal DGMR model from DeepMind [4], which leveraged a GAN architecture with recurrent units and multi-scale reasoning for precipitation nowcasting.

Following this, we examine PhyDNet [10], which built on a standard RNN structure but introduced PhyCell, a module designed to encode physical priors through differential operators, encouraging the model to respect the physics underlying atmospheric dynamics. We next turn to more recent diffusion-based methods. PreDiff [11] formulates precipitation forecasting as a video diffusion problem in latent space using a VAE and a cuboid-attention U-Net. However, like many diffusion-based models, it suffers from long training times. To address this, DiffCast [12] introduced a residual-diffusion approach that models only the correction (residual) between a base model's prediction and the ground truth, dramatically reducing diffusion model size and training cost.

We then explore transformer-based nowcasting methods. MAU [13] introduces a novel motion-aware unit that aggregates temporal information across time steps and fuses it with the current frame via a gating mechanism. Earthformer [14] proposes a space-time attention transformer that captures long-range spatial and temporal dependencies without relying on recurrence. Finally, NowcastingGPT [15] reframes the problem as a visual language modelling task: it compresses radar frames into discrete tokens using a VQ-VAE and autoregressively predicts these tokens using a transformer decoder.

After introducing these key contributions in the nowcasting literature, we position our own model within this landscape, explain its design philosophy, and motivate the selection of benchmarks used in our evaluation.

2.1 DGMR: Deep Generative Models of Radar

In [4], DeepMind introduced DGMR (Deep Generative Model for Radar), a pioneering GAN-based approach for precipitation nowcasting. The model, proposed in 2021, draws inspiration from video prediction architectures and was among the first to demonstrate that generative methods could produce highly realistic and skilful weather forecasts. The generator in DGMR is structured as a multi-scale stack of ConvGRU modules, operating over different spatial resolutions of the radar input. At each resolution level, the ConvGRU receives two inputs: (1) the hidden state from the previous time step at the same level, and (2) the output from the ConvGRU at the lower resolution but at the current time step. This hierarchical structure allows the model to capture fine-to-coarse spatial dependencies while maintaining temporal coherence across scales. At the highest level (i.e., the lowest spatial resolution), the GRUs are fed a noise vector, as is common in GANs. Each ConvGRU module then produces two outputs: an updated hidden state at the current level, and an upsampled output to be passed as input to the next level above. At the first time step, the ConvGRUs at different resolutions are fed in contextual information from the previous frames, at that resolution.

To guide the training of the generator, the authors introduce two discriminators in a standard GAN framework. A spatial discriminator, a 2D convolutional network, evaluates individual predicted frames for realism, ensuring sharpness and spatial consistency. A temporal discriminator, a 3D convolutional network, evaluates short sequences of frames to enforce temporal coherence across time steps. This dual-discriminator setup helps the model generate predictions that are both spatially plausible and temporally stable. DGMR demonstrated state-of-the-art performance at the time, and its design laid the groundwork for many follow-up works in generative nowcasting. Notably, it also showed that adversarial training could improve categorical performance metrics such as Critical Success Index (CSI) without sacrificing realism. Although GAN-based video prediction was an important early branch of video generation, they have since been replaced by diffusion-based and transformer-based techniques, which show better temporal consistency and realism. Motivated by the general move of the video prediction domain, we also choose the transformer-based design principle.

2.2 TrajGRU for Precipitation Nowcasting

The authors of [9], a landmark and highly cited paper in precipitation nowcasting, introduce TrajGRU—a variant of Convolutional GRUs (ConvGRUs) specifically designed for spatiotemporal prediction tasks. To briefly summarise: a GRU is a type of recurrent neural network (RNN) designed to mitigate the vanishing gradient problem, and a ConvGRU replaces the linear transformations in a GRU with convolutional operations, making it suitable for spatial data such as video or radar frames. However, ConvGRUs are inherently translation-invariant, which is problematic for weather data- weather dynamics are not translation-invariant, as they are strongly influenced by local topography and directional flow patterns like wind. To address this, TrajGRU replaces the fixed local neighbourhood in a convolution with a learned, dynamic trajectory of connections. The model learns where to gather information from at each location, enabling different spatial regions to focus on different contextual areas across time.

The core unit is the TrajGRU cell (illustrated in Figure 2.2), and in practice, the authors stack multiple such layers with downsampling and upsampling operations (shown in Figure 2.1). This hierarchical structure allows the network to capture information at multiple spatial and temporal scales, with higher layers exerting influence over the dynamics of lower layers. Despite its ingenuity, the community has since shifted toward transformer-based and diffusion-based models, which offer key advantages. Transformer architectures with causal attention can model the entire temporal context up to the current time step, making them more expressive and better suited for long-term forecasting. Moreover, attention mechanisms implicitly learn spatiotemporal structure, enabling the model to focus on the most relevant regions of the map dynamically. Our transformer-based latent space architecture provides full causal access to past information and learns spatial structure adaptively via attention, offering a more flexible and powerful alternative to TrajGRU.

2.3 Disentangling Physics in the Latent Space: PhyDNet

PhyDNet [10] is a physically informed video prediction model that aims to disentangle physical dynamics from residual unknown factors in unsupervised video prediction tasks. Recognising that physical laws often do not apply directly in pixel space, the authors propose learning a latent space where physics-inspired constraints can operate meaningfully.

The architecture consists of two recurrent branches: a physical branch that uses a novel cell called PhyCell, inspired by data assimilation methods (e.g., Kalman filtering), to encode partial differential equation (PDE)-based dynamics in latent space. A residual branch, implemented with a standard ConvLSTM, captures visual information not accounted for by the physics model (e.g., texture, appearance). Together, these components form a sequence-to-sequence architecture that learns to predict future video frames. The PhyCell itself incorporates a prediction-correction mechanism. First, it performs a physically constrained latent state update based on learned spatial differential operators. Then, it adjusts this prediction using actual observations through a learned gating mechanism (interpretable as a Kalman gain), which helps manage noisy or missing inputs.

The model is trained using both a reconstruction loss and a physics-based

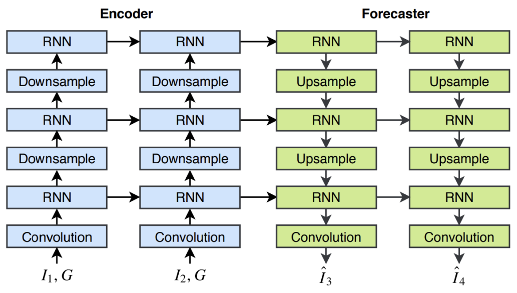
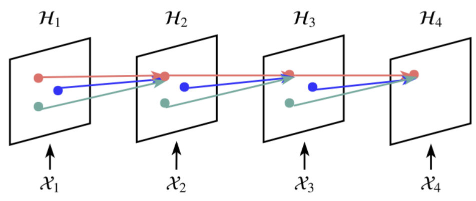
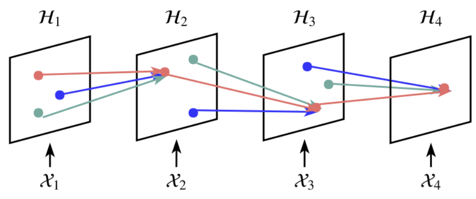


Figure 2.1: Full model structure of [9]- showing an RNN on multiple levels with hidden states capturing different levels of detail. The model consists of layered RNNs interspersed with convolutional layers.



(a) For convolutional RNN, the recurrent connections are fixed over time.



(b) For trajectory RNN, the recurrent connections are dynamically determined.

Figure 2.2: ConvGRUs vs TrajGRU. In ConvGRUs, since the convolution is translation invariant, the same neighbourhood affects a point of interest in the following map. This structure is, however, learnt, and can be dynamic in TrajGRU

regularisation term that ensures the learned filters approximate PDE operators. This regularisation, along with the disentangled architecture, significantly boosts performance across datasets.

Figure 2.3 illustrates the core architecture: it shows how input frames are encoded into a shared latent space, processed by the PhyCell and ConvLSTM branches in parallel, and recombined to decode predicted future frames. This figure highlights the disentanglement of physical and residual dynamics, which is the central innovation of PhyDNet.

2.4 PreDiff: a Latent Diffusion Model

PreDiff [11], introduced in 2023, presents a novel approach to precipitation nowcasting using diffusion models in a continuous latent space. The method begins by encoding the input event into a latent space using a frame-wise variational autoencoder (VAE). Prediction is then performed within this latent space via a diffusion process, where each denoising step conditions on the noise-free context frames and the noisy target frames from the previous step. The diffusion backbone is a U-Net architecture enhanced with cuboidal attention mechanisms inspired by Earthformer. To incorporate physical realism, PreDiff introduces a knowledge alignment module

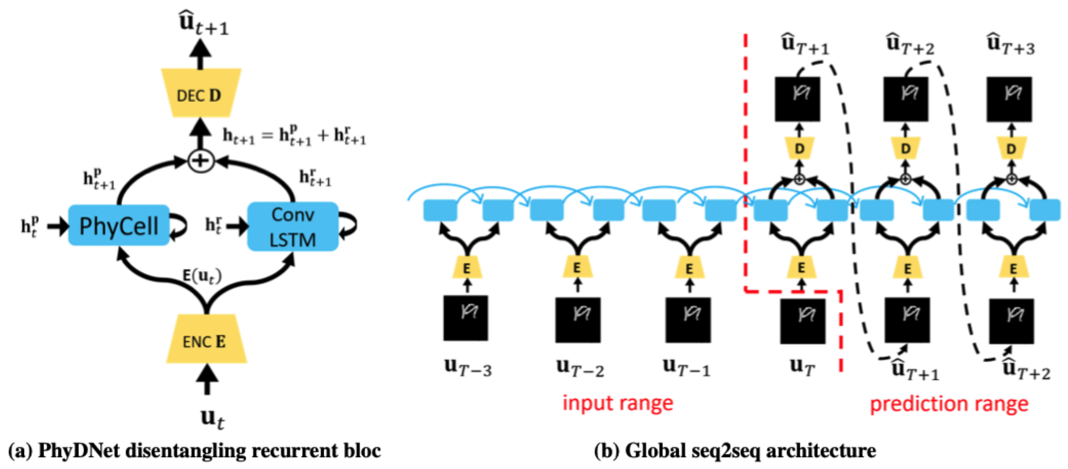


Figure 2.3: The core architecture pipeline of PhyDnet, which disentangles physics and orthogonal features in a latent space, and imposes physics constraints via PDEs in a PhyCell [10]

.

that modifies the reverse diffusion transition distribution. Specifically, it predicts the expected average intensity of future frames based on the context and penalises deviations from this expectation during denoising. This mechanism helps enforce physically meaningful constraints directly within the generative process.

While the method is conceptually elegant and introduces a compelling way to integrate physical priors into diffusion modelling, it suffers from very high computational costs. Training the full pipeline (including the VAE, diffusion model, and knowledge alignment network) takes 45 days on 4 A100 GPUs. Such substantial training time, characteristic of many diffusion-based models, makes these approaches less practical for fast experimentation. For this reason, we adopt fast and efficient transformer-based models in our work.

2.5 Modelling Residual Predictions with Diffusion: Diffcast

DiffCast [12] introduces a flexible framework for precipitation nowcasting that explicitly models the dual nature of weather systems: global deterministic motion and local stochastic variation. Recognising the limitations of both deterministic models (which often produce blurry outputs and miss high-intensity regions) and probabilistic models (which offer realism but sacrifice spatial accuracy), the authors propose a residual diffusion mechanism. In this setup, a deterministic backbone (any standard spatiotemporal predictor (e.g. PhyDNet)) forecasts a coarse global trend, while a temporal residual diffusion model refines this prediction by learning to model the stochastic residuals. Central to the diffusion module is their 'Global Temporal UNet' (GTUNet), which captures temporal dynamics and ensures consistency.

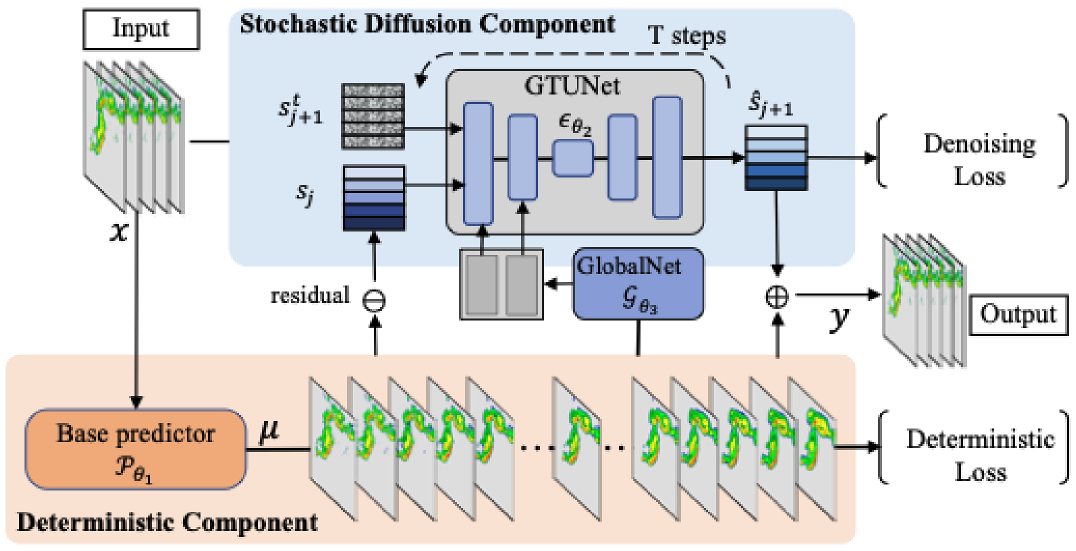


Figure 2.4: The Diffcast pipeline [12]. Diffcast operates on a deterministic backbone, which is an existing weather prediction model. Diffcast aims to model the residual between the backbone's predictions and the ground truth via a diffusion U-Net. The U-Net used is called a GTUNet (Global Temporal UNet), which essentially compresses frames into blocks and then uses diffusion to predict the residual in this compressed space.

Both components are trained end-to-end, and visualised in Figure 2.4. The work also achieved the most recent state-of-the-art results on four datasets- SEVIR [16] (which we also use), Shanghai [17], MeteoNet [18] (from France), and the CIKM competition dataset [19].

2.6 MAU: A Motion-Aware Unit for Video Prediction

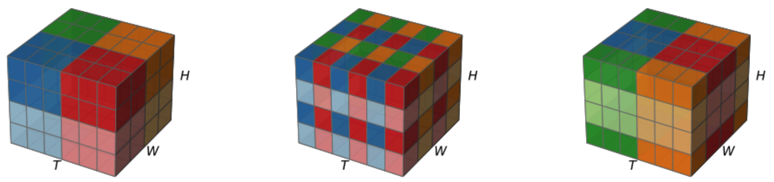
Another work that specifically addresses the vanishing gradient problem in RNNs for video prediction is presented in [13]. The authors propose the Motion-Aware Unit (MAU), which operates on a set of spatial and temporal states from a fixed number of previous time steps. The MAU first captures motion information by dynamically aggregating past temporal states using attention-like weights that reflect the correlation between the current and previous spatial states. Once motion is inferred, a gating mechanism integrates this motion information with the current spatial state to produce the next spatial output.

However, a key limitation of this design is that its attention weights are based on the correlation between spatial states—an inductive bias that may be ill-suited for domains such as weather prediction. For example, if a rain cloud moves across the map over time, the spatial patterns at each time step may not be strongly correlated due to the cloud's displacement. Consequently, a purely correlation-based attention mechanism might fail to identify temporal dependencies in such dynamic settings. This motivates a more flexible, fully-attentional design, as we adopt in our work, where any location in the current frame can learn to attend to any spatiotemporal location in the past, regardless of fixed spatial alignment. Further, the authors are forced to constrain the model to attend to only a fixed number of time steps. They do so to reduce computational burden. Having to do this despite functioning in the latent space also perhaps speaks to the complexity of their architecture, with various sub-modules having to perform sub-tasks like first capturing motion, and then integrating temporal information with the current spatial state. In our transformer-based design, which also functions in the latent space, the simplicity of our architecture allows a full temporal receptive field.

2.7 Earthformer: Cuboidal Attention for Nowcasting

Earthformer [14], a landmark work in weather prediction, introduced a Transformer-based architecture tailored for spatiotemporal forecasting, effectively framing the problem as video prediction in pixel space. A key challenge with applying self-attention directly in this domain is the quadratic computational cost associated with pixel-wise attention. To address this, the authors propose Cuboidal Attention, which partitions the spatial-temporal volume into blocks (or cuboids), allowing self-attention to operate within each block independently. To compensate for the lack of communication between blocks, the model incorporates global tokens, which aggregate and redistribute information across blocks via cross-attention layers. The authors explore two cuboidal grouping strategies: a local scheme, where neighbouring pixels are grouped together, and a dilated scheme, where pixels are assigned to blocks in a strided, alternating pattern. These mechanisms are illustrated in Figure 2.5.

While Earthformer is Transformer-based like our proposed model, it differs in several key aspects. It follows an encoder-decoder transformer architecture, where the encoder compresses context from past frames into global representations, and the decoder generates future frames directly from position embeddings conditioned on this context. Due to the cuboidal attention mechanism, causality is not enforced—pixels at a given time step may depend on future pixels, making the model inherently non-autoregressive. Consequently, Earthformer is constrained to fixed input-output lengths and cannot be rolled out autoregressively over time, unlike our frame-level autoregressive model, which supports flexible temporal horizons and explicit causal generation.



(a) strategy="`local`" shift=(0, 0, 0) (b) strategy="`dilated`" shift=(0, 0, 0) (c) strategy="`local`" shift = (0, 1, 1)

Figure 2.5: The cuboidal self-attention blocks used in Earthformer. Cells of the same colour belong to the same block and attend to each other. Their 'local' mechanism clusters neighbouring cells in the same block. An option to shift these blocks in either of the dimensions is also available, as shown in the third figure, where local blocks are shifted in the height and width dimensions by 1. The dilated scheme allows cells to be alternatively sampled and assigned to a block [14].

2.8 An LLM approach to Weather Prediction: NowcastingGPT-EVL

Previous thesis work in the research group proposed NowcastingGPT-EVL [15], a Transformer-based generative model tailored for extreme precipitation nowcasting. Building on the VideoGPT [20] framework, the model integrates a VQ-VAE [21] encoder and an autoregressive Transformer [22] to forecast radar-based precipitation maps. A key contribution is their novel formulation of the Extreme Value Loss (EVL), which dynamically adjusts weights for rare events without assuming fixed latent representations—an inductive bias shown to limit prior models. The study targets high-impact events using radar sequences from the KNMI dataset and benchmarks performance across standard metrics (MSE, MAE, PCC, CSI, FAR, and FSS). The authors demonstrated that NowcastingGPT-EVL consistently outperforms baselines in both visual fidelity and extreme event detection. Their pipeline achieves the highest AUC-ROC analysis for extreme event classification, outperforming TECO [23], PySTEPS [24], and an unregularized NowcastingGPT variant. This work reinforces the importance of explicitly modelling extremes in nowcasting and showcases the benefits of task-specific loss functions for rare event forecasting.

More importantly, in the context of this thesis, we investigate in detail and build upon the inductive biases and framework leveraged in NowcastingGPT. In contrast to the focus of the paper, instead of focusing on extreme values, we focus on tackling the core computer vision pipeline and improving its performance fundamentally. We leave extending this backbone with extreme value loss to future studies.

In light of recent advancements in video prediction, particularly the rise of diffusion and transformer-based architectures, we elect to pursue a transformer-centric design for our work. While diffusion models have demonstrated strong generative capabilities in spatiotemporal domains, their practical adoption is often hindered by prohibitively high training costs. For instance, as demonstrated by PreDiff [11], training even a single diffusion-based nowcasting model can require over a month on multiple A100 GPUs, rendering such approaches infeasible for many academic or resource-constrained environments like ours.

Transformer-based models, on the other hand, offer a compelling balance between scalability, efficiency, and modeling power. Their ability to capture long-range spatial and temporal dependencies, without the iterative overhead of diffusion sampling, makes them particularly attractive for rapid experimentation and deployment. Moreover, our decision to adopt a transformer-based architecture is further motivated by a desire to directly build upon and improve the prior work from our research group, NowcastingGPT [15]. While NowcastingGPT pioneered a novel visual language modeling framework for precipitation forecasting using VQ-encoded radar frames, it also introduced certain limitations in temporal modeling and prediction granularity. Our proposed model critically addresses these limitations, offering a refined transformer-based solution that advances both performance and efficiency within the nowcasting domain.

2.9 Selection of Benchmarks Used

The following are the benchmarks used to compare our models:

1. NowcastingGPT
2. Diffcast (with PhyDnet as its backbone)

NowcastingGPT attained state-of-the-art performance on the KNMI dataset, and is fundamentally the core idea which is innovated on in this thesis; thus, one of our benchmarks was selected to be NowcastingGPT. Note that for fair comparison, we do not use the EVL variant, which regularises extreme values, but the core pipeline instead. Our second benchmark was chosen to be Diffcast, along with PhyDnet as its backbone. This was chosen since a combination of Diffcast and PhyDnet was shown to achieve superior performance compared to Earthformer, MAU, TrajGRU, PreDiff and PhyDnet in [12] on the SEVIR dataset. Not only are NowcastingGPT and Diffcast state-of-the-art, but they also represent the two most recent philosophies from the video prediction domain- transformer-based discrete prediction models paired with an encoder, and diffusion-based models.

3 Methodology

This chapter presents the methodology used for weather prediction in this thesis. We begin by formulating the problem and providing a high-level overview of the proposed pipeline. Each component of the pipeline is then described in detail and formalised mathematically.

3.1 Problem Formulation

It is first important to define the language and mathematical notation in which our design will be introduced and motivated. Since our pipelines are designed in the context of the computer vision domain, we will use terms from the same. Each precipitation map will be termed a frame. A collection of such frames forms an event. Mathematically, each event can be represented by a sequence of frames,

$$\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T\},^{1} \qquad \mathbf{X}_t \in \mathbb{R}^{H\times W}, \tag{3.1}$$

where H and W denote the spatial dimensions of each frame, and T is the sequence length, or total number of frames in each event.

The objective of this thesis is to predict a sequence of future weather frames conditioned on a context of past frames. Mathematically, given a sequence of T_c context frames

$$\mathcal{X}_{\text{context}} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{T_c}\}, \quad \mathbf{X}_t \in \mathbb{R}^{H\times W}, \tag{3.2}$$

the task is to generate the future frames

$$\mathcal{X}_{\text{target}} = \{\mathbf{X}_{T_c+1}, \ldots, \mathbf{X}_T\}, \tag{3.3}$$

such that the full sequence is $\mathcal{X} = \{\mathcal{X}_{\text{context}}, \mathcal{X}_{\text{target}}\}$. The prediction model learns a mapping

$$f : \mathcal{X}_{\text{context}} \mapsto \mathcal{X}_{\text{target}}. \tag{3.4}$$

3.2 Pipeline Overview

In our pipeline, we decompose the task of mapping $\mathcal{X}_{\text{context}} \mapsto \mathcal{X}_{\text{target}}$ into two subtasks. The first involves compressing each frame into a set of discrete latent

[1] Braces are used for convenience to represent a sequence. They do not represent a set.

tokens. Mathematically, we learn a function

$$f_{compress} : \mathcal{X} \mapsto \mathcal{T}, \tag{3.5}$$

where

$$\mathcal{T} = \{\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_T\}, \quad \mathbf{T}_t \in \{1, \ldots, K\}^{H' \times W'},$$

and H' and W' denote the spatial resolution of the compressed event. Each element in $\mathbf{T}_t$ is a discrete token that indexes into a finite latent vocabulary of size K.

This compression is motivated by the fact that modelling the full-resolution frames in latent space (e.g.,128 $\times$ 128) is computationally prohibitive owing to the quadratic complexity of self-attention. We employ a Vector Quantised Variational Autoencoder (VQ-VAE) [21] to learn $f_{compress}$, which first downsamples the frame using convolutional layers and then quantises the result into discrete tokens. The second subtask is to model the dynamics between the latent tokens. Specifically, we learn a function

$$f_{dynamics} : \mathcal{T}_{context} \mapsto \mathcal{T}_{target}, \tag{3.6}$$

where $\mathcal{T}_{context} = \{\mathbf{T}_1, \ldots \mathbf{T}_{T_c}\}$ and $\mathcal{T}_{target} = \{\mathbf{T}_{T_c+1}, \ldots \mathbf{T}_T\}$. One of our benchmarks, NowcastingGPT, draws from foundational work in image and video generation with quantised latents [20], [25], treating the discrete latents as a flat sequence and applying an autoregressive transformer to model token-by-token dependencies. Mathematically, they flatten each frame's token grid $\mathbf{T}_t$ into a one-dimensional sequence:

$$\mathbf{z}_t = flatten(\mathbf{T}_t) = \left(z_t^{(1)}, z_t^{(2)}, \ldots, z_t^{(H'W')}\right), \quad z_t^{(i)} \in \{1, \ldots, K\} \tag{3.7}$$

The full event sequence is then obtained by concatenating these per-frame sequences in temporal order:

$$\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T) \tag{3.8}$$

or equivalently,

$$\mathbf{z} = \left(z_1^{(1)}, \ldots, z_1^{(H'W')},\ z_2^{(1)}, \ldots, z_T^{(H'W')}\right).$$

This flattened token sequence $\mathbf{z}$ forms the input to the temporal dynamics model. They model the dynamics of the sequence $\mathbf{z}$ using an autoregressive transformer, which factorises the joint distribution over tokens as a product of conditional distributions:

$$p(\mathbf{z}) = \prod_{i=1}^{T \cdot H'W'} p\left(z^{(i)} \mid z^{(1)}, z^{(2)}, \ldots, z^{(i-1)}\right). \tag{3.9}$$

Here, $z^{(i)}$ denotes the i^{th} token in the flattened sequence $\mathbf{z}$.

Our approach, which we call BlockGPT, introduces an approach inspired by recent advancements in video generation [26] and models the dynamics of latents frame-by-frame. Specifically, we claim that the factorisation introduced in NowcastingGPT, in Equation 3.9 is suboptimal, and instead factorise the joint distribution over the latent sequence $\mathcal{T} = \{\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_T\}$ as:

$$p(\mathcal{T}) = \prod_{t=1}^{T} p\left(\mathbf{T}_t \mid \mathbf{T}_1, \dots, \mathbf{T}_{t-1}\right). \tag{3.10}$$

This factorisation allows each token grid $\mathbf{T}_t$ to retain its two-dimensional structure, and enables bidirectional attention within a frame. Although the tokens are discrete and share similarities with words in a sentence, they represent visual content and are inherently spatially structured. As a result, tokens from a frame are highly correlated. Factorising the joint distribution of tokens as per Equation 3.9 is suboptimal, since it treats tokens z_t from the same frame t like a causal sequence, when in reality they are mutually correlated. Treating them as a sequence forces the dynamics module to treat visual tokens from a frame as a one-directional entity, and hence imposes a suboptimal inductive bias. Our frame-by-frame approach enables the joint distribution to be factorised more naturally as per Equation 3.10, which retains this 2D structure by allowing the tokens in a frame to attend bidirectionally with each other. This design more naturally aligns with the underlying structure of precipitation data: spatial patterns within a radar map are best interpreted holistically, while future frames should depend on past context. The approach is also $N\times$ faster during inference, where N is the number of tokens used to represent a single frame. Further details of each part of the pipeline will be elaborated on in the following sections. The two stages in our pipeline are visualised in Figure 3.1.

3.3 Learning $f_{compress}$ with VQGANs

Our first subtask is to learn a function $f_{compress} : \mathcal{X} \mapsto \mathcal{T}$. For each frame $\mathbf{X}_t$ in the sequence, we apply the compression function f_{compress} to obtain a discrete token map:

$$f_{\text{compress}}(\mathbf{X}_t) = \mathbf{T}_t \in \{1, \dots, K\}^{H' \times W'}. \tag{3.11}$$

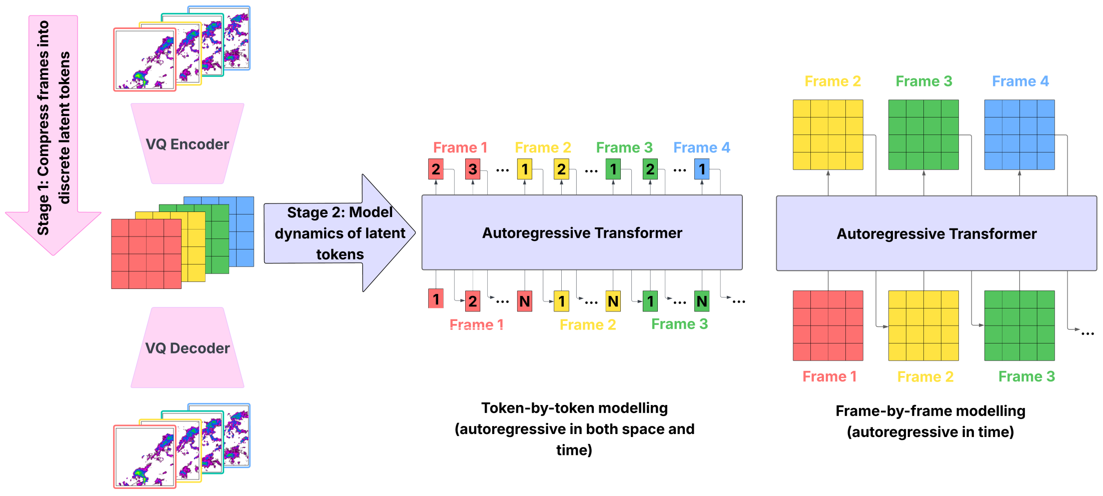
This function is learnt via a Vector-Quantised Generative Adversarial Network (VQGAN) [21], which includes an encoder, codebook, and decoder. The encoder produces an intermediate embedding tensor:

$$\mathcal{E}_t = \text{Encoder}(\mathbf{X}_t) \in \mathbb{R}^{H' \times W' \times D}, \tag{3.12}$$

where D is the embedding dimension. The embedding vector at spatial location (i, j) is denoted by $\mathcal{E}_t[i, j, :] \in \mathbb{R}^D$.

Each entry at the index (i, j) in $\mathcal{E}_t$ is quantised to the nearest codebook entry:

$$\mathbf{T}_t[i, j] = \arg\min_k \|\mathcal{E}_t[i, j, :] - \mathbf{c}_k\|_2^2, \quad \mathbf{c}_k \in \mathbb{R}^D, \ k \in \{1, \dots, K\}. \tag{3.13}$$



Stage 1: Compress frames into discrete latent tokens
VQ Encoder
VQ Decoder
Stage 2: Model dynamics of latent tokens
Autoregressive Transformer
Frame 1
Frame 2
Frame 3
Frame 4
Token-by-token modelling (autoregressive in both space and time)
Frame-by-frame modelling (autoregressive in time)

Figure 3.1: An overview of our two-stage pipeline for weather prediction. In stage 1, event frames are compressed into a set of discrete latent tokens. A key point is that the function $f_{compress}$ operates at the frame level, and does not temporally compress an event. Stage 2 involves modelling the dynamics of latent tokens to enable prediction. In NowcastingGPT (middle), an autoregressive transformer learns token-by-token dynamics by treating the full set of tokens as a single sequence. The transformer is then trained to predict a token, given a past set of tokens. Our approach, BlockGPT (right), works on the frame level. Here, the autoregressive transformer is trained to predict an entire frame, given past frames.

Essentially, the vector $\mathcal{E}[i,j,:]$ output from the encoder is mapped to the nearest embedding $\mathbf{c}_k$ from a codebook or dictionary of size K. $\mathbf{T}_t[i,j]$ can then be simply represented by the index of the codebook embedding, instead of the embedding itself. The codebook, $\mathbf{C} \in \mathbb{R}^{D\times K}$ is essentially a learnable matrix with K columns and D rows. In simpler terms, all we are doing is replacing each $\mathcal{E}[i,j,:]$ with the column number of the nearest (in L2 norm) column of the codebook matrix. This completes our mapping from $\mathbf{X}_t$ to $\mathbf{T}_t$. This combination of encoder and codebook is what comprises our function $f_{compress}$. However, the encoder and codebook are trainable, and they need to be learnt such that we can recover the frame in the pixel space $\mathbf{X}_t$ from $\mathbf{T}_t$. We thus need to complete the pipeline back to the pixel space. Storing the vectors $\mathbf{c}_k$ at every index $[i,j]$, $\mathbf{T}_t$ can equivalently be represented by a tensor $\mathcal{C}_t \in \mathbb{R}^{H'\times W'\times D}$, where,

$$\mathcal{C}_t[i,j,:] = \mathbf{c}_{\mathbf{T}_t[i,j]} \qquad 1 < i < H', 1 < j < W'. \tag{3.14}$$

The decoder takes the tensor $\mathcal{C}_t$, and then reconstructs the frame from the quantised embeddings:

$$\hat{\mathbf{X}}_t = \text{Decoder}\left(\mathcal{C}_t\right). \tag{3.15}$$

The encoder-quantizer-decoder pipeline is visualised in Figure 3.2.

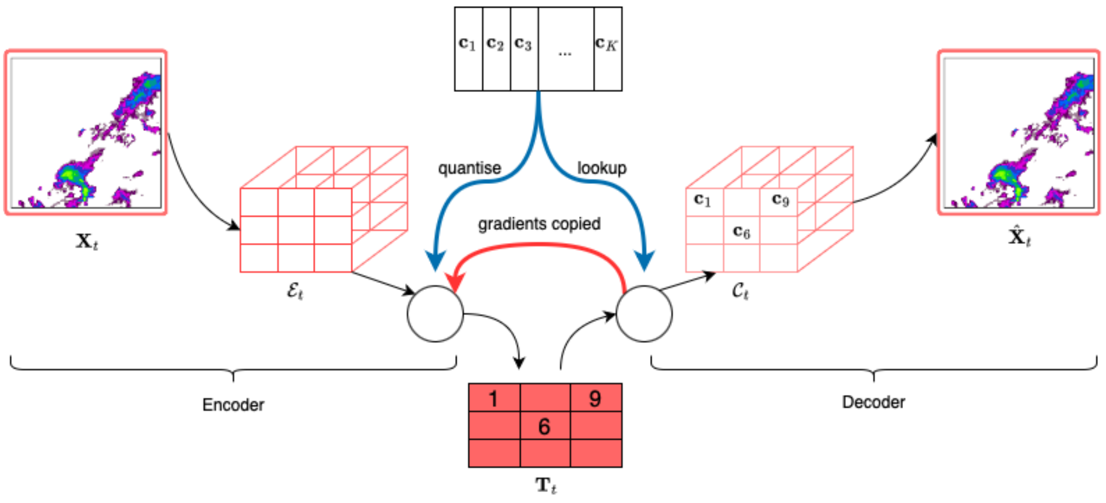


Figure 3.2: Encoder-quantizer-decoder pipeline in a VQGAN(discriminator not visualised). The encoder (block of convolutional layers) downsamples the frame $\mathbf{X}_t$ while multiplying the channel dimension. The resulting tensor $\mathcal{E}_t$ is quantised at every index (i, j) by the nearest vector $\mathbf{c}$ in the codebook. $\mathcal{E}_t$ can then be represented simply by the indices of the sampled codebook embeddings. The resulting matrix $\mathbf{T}_t$ is the compressed representation used in the dynamics module. To reconstruct the frame, the indices in $\mathbf{T}_t$ are replaced by the codebook embeddings at the respective indices to get $\mathcal{C}_t$. A decoder (block of convolutional layers) then upsamples $\mathcal{C}_t$ to get the reconstructed frame $\hat{\mathbf{X}}_t$. Figure adapted from [21].

Training Objective

The quantisation operation used makes the pipeline non-differentiable. The authors of [21] employ a simple straight-through estimator [27], which copies gradients from the decoder input $\mathcal{C}_t$ to the output of the encoder $\mathcal{E}_t$. The intuition given in [21] is that since both $\mathcal{E}_t$ and $\mathcal{C}_t$ lie in the same D-dimensional space, the decoder gradients provide a useful learning signal for the encoder, despite the quantisation bottleneck.

The loss function used to train the VQGAN comprises four components. The first is the reconstruction loss, which is standard in any encoder–decoder pipeline. This term encourages the decoder to accurately reconstruct the input frame:

$$\mathcal{L}_{\text{recon}} = \|\mathbf{X}_t - \hat{\mathbf{X}}_t\|_F^2, \tag{3.16}$$

where $\hat{\mathbf{X}}_t = \text{Decoder}(\mathcal{C}_t)$ is the reconstructed frame from quantised embeddings.

However, since gradients are not directly propagated through the codebook, additional loss terms are necessary. VQGANs adopt a simple dictionary learning mechanism based on vector quantisation. Specifically, the codebook loss is defined as:

$$\mathcal{L}_{\text{codebook}} = \sum_{i,j} \left\| \text{sg}[\mathcal{E}_t[i,j,:]] - \mathbf{c}_{\mathbf{T}_t[i,j]} \right\|^2, \tag{3.17}$$

which pulls the selected codebook entry $\mathbf{c}_{\mathbf{T}_t[i,j]}$ closer to the corresponding encoder output $\mathcal{E}_t[i,j,:]$. Here, sg$[\cdot]$ denotes the stop-gradient operation, which blocks gradients from being propagated into the argument.

A third term, the commitment loss, prevents the encoder output from drifting across the continuous embedding space and encourages it to "commit" to the nearest codebook vector:

$$\mathcal{L}_{\text{commit}} = \sum_{i,j} \left\| \mathcal{E}_t[i,j,:] - \text{sg}[\mathbf{c}_{\mathbf{T}_t[i,j]}] \right\|^2. \tag{3.18}$$

Finally, a last term introduced in VQGANs is the adversarial loss $\mathcal{L}_{\text{GAN}}$, which ensures that the reconstructed samples are perceptually realistic when training with a discriminator.

Putting all terms together, the total VQGAN loss becomes:

$$\mathcal{L}_{\text{VQGAN}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{codebook}} + \beta \cdot \mathcal{L}_{\text{commit}} + \lambda \cdot \mathcal{L}_{\text{GAN}}, \tag{3.19}$$

where β and λ are hyperparameters that balance the influence of the commitment and adversarial losses, respectively.

3.4 Learning $f_{dynamics}$ with Autoregressive Transformers

Our second subtask is to learn $f_{dynamics} : \mathcal{T}_{context} \mapsto \mathcal{T}_{target}$. We use an autoregressive transformer [22] to model the dynamics of the tokens. As explained in the previous section, our model, BlockGPT, models the dynamics frame-by-frame and in the process, imposes a more optimal inductive bias on the model. In order to motivate our design, we parallelly explain the pipelines of NowcastingGPT, which predicts token-by-token and BlockGPT. This lets us highlight the fundamental differences more concretely.

Transformers are essentially mixing or communication modules- they cleverly learn to allow elements of a sequence to attend to other elements. At its core, a transformer simply learns a weighting matrix which mixes a set of vectors depending on their relative importance to each other. To concretise this abstract description, a mathematical description follows, which can broadly be divided into three parts- converting scalar tokens to vector embeddings, mixing these embeddings (or communication) and finally prediction.

Our transformer module deals with the compressed representation given by $f_{compress}$, the set of tokens $\mathcal{T} = \{\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_T\}$. First, each frame's token grid $\mathbf{T}_t$ is flattened into a one-dimensional sequence[2],

$$\mathbf{z}_t = flatten(\mathbf{T}_t) = \left(z_t^{(1)}, z_t^{(2)}, \ldots, z_t^{(H'W')}\right), \quad z_t^{(i)} \in \{1, \ldots, K\}.$$

As explained previously, the full event sequence can be formed by concatenating per-frame sequences in temporal order,

$$\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T)\,. \tag{3.20}$$

As a first step, the scalar tokens are vectorised by sampling from a learnt embedding table. Each token index $z^{(i)}$[3] is mapped to an embedding vector:

$$\mathbf{e}^{(i)\top} = \mathbf{W}_E[z^{(i)}], \quad \text{for } i = 1, \ldots, L, \tag{3.21}$$

where $\mathbf{W}_E \in \mathbb{R}^{K \times D'}$ is the learnable token embedding matrix used by the transformer (distinct from the VQGAN codebook), $\mathbf{e}^{(i)\top} \in \mathbb{R}^{1 \times D'}$, $\mathbf{W}_E[i]$ represents sampling the i^{th} row from $\mathbf{W}_E$ and $L = H' \times W' \times T$. To retain information about the position of each token, we add a corresponding positional embedding $\mathbf{p}_e^{(i)} \in \mathbb{R}^{D'}$ to each token embedding:

$$\mathbf{h}^{(i)} = \mathbf{e}^{(i)} + \mathbf{p}_e^{(i)} \tag{3.22}$$

The final embedding matrix, which is used as input to the transformer, is constructed by stacking these vectors column-wise:

$$\mathbf{E} = \begin{bmatrix} \mathbf{h}^{(1)} & \mathbf{h}^{(2)} & \cdots & \mathbf{h}^{(L)} \end{bmatrix} \in \mathbb{R}^{D' \times L} \tag{3.23}$$

where D' is the embedding dimension used in the transformer.[4] This matrix $\mathbf{E}$ is a representation of the tokens in sequence $\mathbf{z}$ (*and their respective positions in the sequence*) in a higher-dimensional space, and is what is processed at the core of the transformer. The transformer architecture fundamentally revolves around the idea of mixing information across tokens. Given a sequence of token embeddings, the model learns to weight and combine these vectors based on their relevance to one another. This mechanism is called self-attention. The goal of self-attention is, for each token in the sequence, to gather relevant information from all other tokens, weighted by how important they are to this particular token.

To do this, the transformer first computes a matrix of attention weights, which we denote by $\mathbf{W}_A \in \mathbb{R}^{L \times L}$, where L is the length of the token sequence. Each entry $\mathbf{W}_A[i, j]$ measures how much the i^{th} token should attend to the j^{th} token. $\mathbf{W}_A$ is

[2]This is the processing pipeline in a general transformer. Our frame-by-frame method retains the 2D structure of tokens from a frame.

[3]We drop the time subscript for notational convenience.

[4]It is important to note that the embeddings in the transformer are distinct from the codebook vectors $\mathbf{c}_k$.

computed as follows. First, the token embeddings $\mathbf{E}$ are projected by two learnt matrices $\mathbf{W}_Q$ and $\mathbf{W}_K$ into two spaces called the query and key spaces. A query vector of a token represents the information that it wants to know from other tokens, and its key vector represents the information that it contains. Mathematically,

$$\mathbf{Q} = \mathbf{W}_Q\mathbf{E}, \tag{3.24}$$

$$\mathbf{K} = \mathbf{W}_K\mathbf{E}, \tag{3.25}$$

where $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{d\times L}$ (usually $d < D'$) and $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d\times D'}$.

The attention matrix $\mathbf{W}_A$ is then computed by taking the dot products between all queries and keys:

$$\mathbf{W}_A = \text{softmax}\left(\frac{\mathbf{Q}^\top\mathbf{K}}{\sqrt{d}}\right), \quad \mathbf{W}_A \in \mathbb{R}^{L\times L}. \tag{3.26}$$

The softmax function is used to make sure each row sums to 1, and scaling by $\frac{1}{\sqrt{d}}$ is applied for stability [22]. The final output of a transformer block is then,

$$\mathbf{E}_{\text{out}} = \mathbf{V}\mathbf{W}_A, \tag{3.27}$$

where $\mathbf{V} = \mathbf{W}_v\mathbf{E}$ lies in another learnt space, called the value space. Here, $\mathbf{W}_v \in \mathbb{R}^{d\times D'}$ and $\mathbf{V} \in \mathbb{R}^{d\times L}$. In practice, several blocks of these transformer layers are used in conjunction with MLPs.

The formulation in Equation 3.26, however, has one major issue. It lets all tokens communicate with all other tokens. In reality, weather is inherently causal, and tokens at a time step cannot communicate with tokens in the future. The attention score thus has to be masked to prevent this. And the mask used fundamentally differentiates our method from NowcastingGPT. Recall that the token-by-token modelling used in NowcastingGPT factorises the joint distribution of tokens as

$$p(\mathbf{z}) = \prod_{i=1}^{T\cdot H'W'} p\left(z^{(i)} \mid z^{(1)}, z^{(2)}, \ldots, z^{(i-1)}\right).$$

This factorisation is made possible by the assumption that token $z^{(i)}$ is independent of tokens at the $(i+1)^{th}$ position onward. The inductive bias imposed is causality in space and time. This assumption is injected into the attention calculation in Equation 3.26 via a mask depicted in Figure 3.3. This mask prevents tokens at the i^{th} location from communicating with tokens at position $i+1$ onward.

BlockGPT however, is a frame-by-frame approach, and factorises the joint distribution of tokens as

$$p(\mathcal{T}) = \prod_{t=1}^{T} p\left(\mathbf{T}_t \mid \mathbf{T}_1, \ldots, \mathbf{T}_{t-1}\right),$$

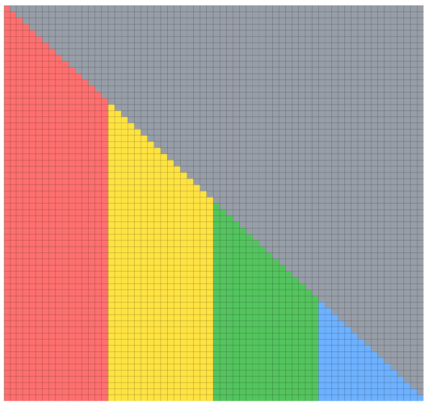


Figure 3.3: Attention mask used in the token-by-token framework. Every token is allowed to communicate with tokens that come before it.

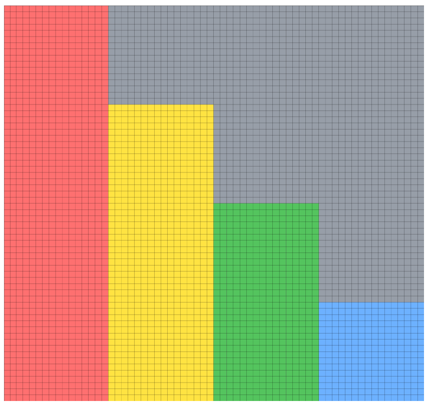


Figure 3.4: Attention mask used in the frame-by-frame framework. Every token in a frame is allowed to attend to each other. However, frames can only look at other frames in the past.

which in terms of $\mathbf{z}_t$ can be written as,

$$p(\mathcal{T}) = \prod_{t=1}^{T} p((z_t^{(1)}, \ldots, z_t^{(H'W')})|(z_1^{(1)}, \ldots, z_1^{(H'W')}), \ldots, (z_{t-1}^{(1)}, \ldots, z_{t-1}^{(H'W')})).$$

This formulation means that tokens in a single frame must be able to communicate with other tokens in the same frame, since they are not independent. Tokens from the frame t can communicate with tokens in the frame 1 to $t-1$, but not $t+1$ onward. This is a more natural inductive bias- tokens are only causal in time, and not in space. To allow this, a block attention mask is used, depicted in Figure 3.4.

The last piece of the puzzle is to use the mixed embeddings $\mathbf{E}_{out}$ for prediction. Again, our method differs from NowcastingGPT. In the token-by-token prediction, the task is to predict the next token. At each step i, the model outputs a probability distribution over the vocabulary (which is of size K) [5] using the attention-mixed embedding vector $\mathbf{e}^i_{out}$:

$$\mathbf{p}^{(i+1)} = \text{softmax}(\mathbf{W}_{\text{out}}\, \mathbf{e}^{(i)}_{out}) \in \mathbb{R}^K, \tag{3.28}$$

where $\mathbf{e}^{(i)} \in \mathbb{R}^{D'}$ is the attention-mixed embedding vector at position i (column i of $\mathbf{E}_{out}$) [6] and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{K\times D'}$ is the learned output projection matrix. The final

[5]Recall that the vocabulary was the size of the codebook $\mathbf{C}_t$- its number of columns. We output a distribution over the codebook so that we may predict the next token.

[6]Note that $\mathbf{E}_{out}$ had dimensions $d \times L$ at the end of Equation 3.27, but now has dimension $D' \times L$. This is because in practice, multiple attention heads are used, each of dimension $d \times L$, and finally concatenated together to give a matrix of dimension $D' \times L$

predicted token is then selected as:

$$\hat{z}^{(i+1)} = \arg\max_k \ \mathbf{p}_k^{(i+1)}$$

We have now predicted the token at position $i+1$, given the tokens at positions 1 to i.

In the frame-by-frame approach, the task is to predict all the tokens at the next frame. Mathematically, we first reshape the matrix $\mathbf{E}_{out} \in \mathbb{R}^{T.H'.W' \times D'}$ to a 4-D tensor $\mathcal{E}_{out} \in \mathbb{R}^{T \times H' \times W' \times D'}$ to recover the original spatial and temporal dimensions for clarity. Each mixed embedding vector at position (i, j) at time t is used to predict the token at position (i, j) at time $t+1$. For each time step $t \in \{1, \dots, T\}$ and spatial position $(i, j) \in \{1, \dots, H'\} \times \{1, \dots, W'\}$, the model outputs a probability distribution over the codebook:

$$\mathbf{p}_{t+1}[i, j] = \text{softmax}(\mathbf{W}_{\text{out}} \, \hat{\mathcal{E}}_{out}[t, i, j, :]) \in \mathbb{R}^K$$

The predicted token at that location is then:

$$\hat{\mathbf{T}}_{t+1}[i, j] = \arg\max_k \ \mathbf{p}_{t+1}[i, j][k]$$

After repeating for every (i, j), we have now predicted the entire frame at time $t+1$, given the frames up to t.

Training Objective

The training objective of the autoregressive transformer is to simply maximise the log-likelihood of the expected token (or minimise the negative log-likelihood). In the token-by-token method, the loss is defined as:

$$\mathcal{L}_{\text{token}} = -\sum_{i=1}^{T \cdot H'W'} \log \left(\mathbf{p}_{z^{(i)}}^{(i)} \right) \tag{3.29}$$

where:

- $\mathbf{p}^{(i)} \in \mathbb{R}^K$ is the predicted probability distribution over the vocabulary at position i,
- $z^{(i)} \in \{1, \dots, K\}$ is the ground-truth token at position i,
- $\mathbf{p}_{z^{(i)}}^{(i)}$ denotes the predicted probability assigned to the correct token.

The same loss holds for the frame-by-frame method. Formally, the loss is defined as the sum of cross-entropy losses across all spatial positions and time steps:

$$\mathcal{L}_{\text{frame}} = -\sum_{t=1}^{T-1} \sum_{i=1}^{H'} \sum_{j=1}^{W'} \log \left(\mathbf{p}_{t+1}[i, j]_{\mathbf{T}_{t+1}[i,j]} \right) \tag{3.30}$$

where:

- $\mathbf{p}_{t+1}[i,j] \in \mathbb{R}^K$ is the predicted distribution at spatial position (i,j) in frame $t+1$,
- $\mathbf{T}_{t+1}[i,j] \in \{1,\ldots,K\}$ is the ground-truth token index at that position,
- $\mathbf{p}_{t+1}[i,j]_{\mathbf{T}_{t+1}[i,j]}$ denotes the predicted probability of the correct token.

This completes the formulation of our method for modelling the dynamics of discrete latent tokens. During training, we assume access to the full token sequence from the event:

$$\mathcal{T} = \{\mathbf{T}_1,\ldots,\mathbf{T}_T\}$$

This allows the model, whether token-by-token or frame-by-frame, to learn to predict future tokens given the complete ground truth. For the token-by-token approach, the model learns to predict each token at position $i+1$ given all previous tokens $z_{(1)},\ldots,z_{(i)}$. Similarly, in the frame-by-frame approach, the model learns to predict the entire token grid $\mathbf{T}_{t+1}$ conditioned on all previous frames $\mathbf{T}_1,\ldots,\mathbf{T}_t$. During inference, however, we have access to only the context:

$$\mathcal{T}_{context} = \{\mathbf{T}_1,\ldots,\mathbf{T}_c\}.$$

To generate the remaining target frames, we perform an autoregressive rollout. That is, at each step:

1. The model uses the currently available tokens to predict the next token grid (in the frame-by-frame case).
2. The prediction is then appended to the sequence and fed back into the model.
3. This process continues until the desired number of future frames has been generated.

This iterative decoding strategy allows the model to simulate the evolution of the latent token space over time, producing plausible future sequences conditioned on the context.

3.5 Evaluation Metrics

To assess the quality of predicted precipitation sequences, we employ a combination of continuous and categorical evaluation metrics. Let an event be defined as a sequence of T precipitation frames:

$$\mathcal{X} = \{\mathbf{X}_1,\ldots,\mathbf{X}_{T_c},\mathbf{X}_{T_c+1},\ldots,\mathbf{X}_T\} = \{\mathcal{X}_{\text{context}},\mathcal{X}_{\text{target}}\},$$

where each frame $\mathbf{X}_t \in \mathbb{R}^{H\times W}$ represents a radar precipitation map at time t, with height H and width W. Given predicted target frames $\hat{\mathcal{X}}_{\text{target}} = \{\hat{\mathbf{X}}_{T_c+1},\ldots,\hat{\mathbf{X}}_T\}$, and corresponding ground truth frames $\mathcal{X}_{\text{target}} = \{\mathbf{X}_{T_c+1},\ldots,\mathbf{X}_T\}$, we define the following metrics.

Continuous Metrics

These metrics evaluate the accuracy of predicted rainfall values across the full spatiotemporal domain.

Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{(T - T_c)HW} \sum_{t=T_c+1}^{T} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(\hat{\mathbf{X}}_t[i,j] - \mathbf{X}_t[i,j] \right)^2 \tag{3.31}$$

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{(T - T_c)HW} \sum_{t=T_c+1}^{T} \sum_{i=1}^{H} \sum_{j=1}^{W} \left| \hat{\mathbf{X}}_t[i,j] - \mathbf{X}_t[i,j] \right| \tag{3.32}$$

Pearson Correlation Coefficient (PCC):

$$\text{PCC} = \frac{\sum_{t,i,j} \left(\hat{\mathbf{X}}_t[i,j] - \bar{\hat{\mathbf{X}}} \right) \left(\mathbf{X}_t[i,j] - \bar{\mathbf{X}} \right)}{\sqrt{\sum_{t,i,j} \left(\hat{\mathbf{X}}_t[i,j] - \bar{\hat{\mathbf{X}}} \right)^2} \sqrt{\sum_{t,i,j} \left(\mathbf{X}_t[i,j] - \bar{\mathbf{X}} \right)^2}} \tag{3.33}$$

where $\bar{\hat{\mathbf{X}}}$ and $\bar{\mathbf{X}}$ denote the mean predicted and ground truth values over all pixels and timesteps in the target sequence.

Categorical Metrics

To evaluate binary event detection (e.g., rainfall exceeding a threshold τ), we binarise both the predicted and target frames:

$$\hat{\mathbf{Y}}_t[i,j] = \mathbb{1} \left[\hat{\mathbf{X}}_t[i,j] \geq \tau \right], \quad \mathbf{Y}_t[i,j] = \mathbb{1} \left[\mathbf{X}_t[i,j] \geq \tau \right]$$

Critical Success Index (CSI):

$$\text{CSI} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{3.34}$$

CSI measures the fraction of correctly predicted rainfall exceedance events out of all observed or predicted exceedances.

False Alarm Rate (FAR):

$$\text{FAR} = \frac{\text{FP}}{\text{TP} + \text{FP}} \tag{3.35}$$

FAR reflects the proportion of predicted rainfall events that were not present in the ground truth.

These metrics jointly capture both the accuracy of predicted rainfall intensities and the reliability of the model in identifying threshold exceedance events, providing a comprehensive view of model performance in practical forecasting scenarios.

4 Datasets and Analysis

In this chapter, we describe the datasets used in our study, with a focus on their relevance to precipitation nowcasting. We utilize two key datasets: a radar-derived precipitation dataset from the Royal Netherlands Meteorological Institute (KNMI)[28], and the Storm EVent Imagery (SEVIR) dataset from the United States [16].

To align with common terminology in the computer vision (CV) literature, we refer to temporal sequences of precipitation fields as *event*, and individual time steps as *frames*. Understanding the statistical properties of the datasets is essential for designing effective models, especially for rare or extreme weather events. Accordingly, we conduct a detailed analysis of each dataset to characterise its distribution and identify relevant patterns.

The following sections provide a comprehensive overview of both datasets, including data formats, temporal and spatial resolutions, and the results of our exploratory analysis.

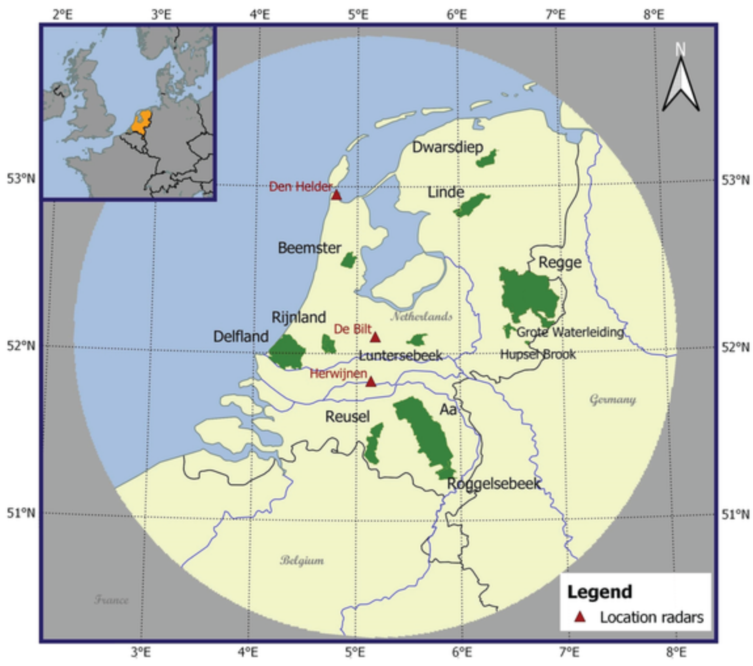
4.1 KNMI Dataset

4.1.1 Introduction to the dataset

The KNMI dataset contains radar-based precipitation estimates collected by two weather radars located in the Netherlands [28]. The data has a spatial resolution of 1 km^2 and a temporal resolution of 5 minutes, covering the entire land area of the Netherlands. It spans the period from 2008 to 2018, during which the radar infrastructure underwent a significant upgrade. Until September 2016, data were collected from radars located in De Bilt and Den Helder. Afterwards, these were replaced by newer dual-polarisation radars situated in Herwijnen and Den Helder.

The raw measurements recorded by these radars are in the form of *radar reflectivity*, which quantifies the amount of transmitted microwave energy reflected back after encountering precipitation particles. Reflectivity values, expressed in units of mm^6/m^3, span a wide dynamic range due to differences in particle size between light rain and intense thunderstorms. To manage this variation, reflectivity is typically expressed on a logarithmic scale [29]. These reflectivity values are then converted to precipitation rates using a standard Z–R relationship, given by:

$$Z_h = 200R^{1.6}$$



Number	Catchment name	Area (km^2)
1	Regge	957
2	Aa	836
3	Delfland	379
4	Reusel	176
5	Linde	150
6	Rijnland	89
7	Roggelsebeek	88
8	Dwarsdiep	83
9	Beemster	71
10	Luntersebeek	63
11	Grote Waterleiding	40
12	Hupsel Brook	6.5

Figure 4.1: (Left)Map of the Netherlands, with catchments (in green) and radars (red triangles) displayed. (Right) Names of the catchment zones and their area in km^2 [33].

where Z_h is the horizontal radar reflectivity factor and R is the precipitation rate in mm/hr [30].

It is important to note that the KNMI dataset used in this thesis already contains the processed precipitation estimates. Thus, we do not perform the reflectivity-to-precipitation conversion ourselves. However, additional preprocessing steps are applied to adapt the data to our modelling pipeline. The original data is provided on a 765×700 spatial grid, which includes surrounding regions outside the Netherlands as well as a circular radar mask that is not of interest to our study. We crop the grid to a region encompassing most of the Netherlands and downsample it to 128×128 using bilinear interpolation. The coordinates used for cropping were retained from prior thesis work in our research group [31], [32].

In addition to modelling precipitation across the entire Netherlands, we pay special attention to twelve catchment zones defined in [33]. These regions are of particular importance to Dutch water authorities due to their sensitivity to rainfall accumulation and their susceptibility to adverse outcomes during extreme weather events. Evaluation of our models will be performed particularly in these catchment areas, in addition to the whole of the Netherlands. The catchment areas, along with the three radar locations used for data collection, are illustrated in Figure 4.1.

4.1.2 Dataset Analysis

The primary objectives of the data analysis phase were twofold: (i) to understand the underlying distribution of precipitation events in the dataset, and (ii) to perform effective data curation. The raw dataset comprises radar frames recorded every five minutes over a span of ten years. As a result, a significant portion of the recorded events exhibit little to no precipitation activity. Including such events in the training set would bias the model toward predicting the absence of precipitation, which is undesirable for our goal of robust forecasting under varying conditions. To ensure the model is trained across a representative range of precipitation intensities, the dataset must be made as balanced as possible. This necessitates the removal of low-activity (or "empty") events and a careful examination of the statistical distribution of precipitation values.

For the analysis, we define an event as a 4-hour window sampled at a 30-minute temporal resolution. This setup is motivated by the specific problem formulation, which will be discussed in Chapter 5. Each event is characterised by its average precipitation, computed across both time and spatial dimensions. The distribution of these average values is then visualised using the following techniques:

- Violin plots, which represent data using density curves [34]. At each value of average precipitation on the y-axis, the width of the plot indicates the frequency of events with that particular value. To further interpret the distribution, quantile lines are overlaid at the 20th, 40th, 60th, and 80th percentiles.
- Quantile-Quantile(QQ) plots, which are scatter plots comparing the quantiles of the empirical distribution with those of a theoretical distribution [35]. In our case, we use the standard normal distribution as the theoretical reference. Q-Q plots help assess how closely the empirical data follows a Gaussian shape and reveal any skewness or heavy tails.

To remove no-precipitation events from the dataset, we truncate a fraction of the events based on their average precipitation. To guide this truncation threshold, we visualise the data distribution under different retention percentiles. This allows us to observe how the dataset's statistical characteristics evolve as more low-precipitation events are discarded.

The violin plots in Figure 4.2 reveal that the original distribution is heavily skewed, with a large concentration of events exhibiting very low precipitation. This confirms the initial hypothesis that the dataset is highly imbalanced. As increasingly larger fractions of low-percentile events are removed, the distribution becomes progressively more balanced, with reduced skewness.

The Q-Q plots in Figure 4.3 provide a complementary perspective. In the original dataset, the left tail (representing lower precipitation values) is horizontally

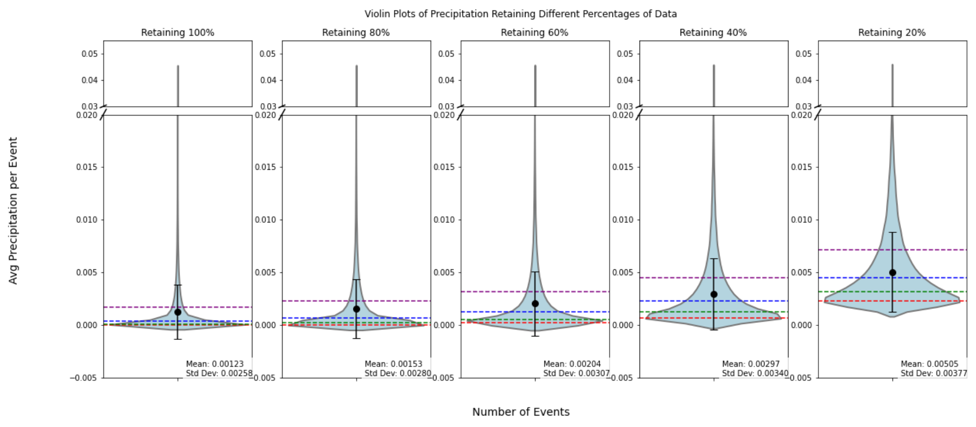


Figure 4.2: Violin plots of event average precipitation in the KNMI dataset, with truncation of increasing bottom-percentiles of data. The horizontal lines represent the 20^{th}, 40^{th}, 60^{th} and 80^{th} percentiles. The dataset is clearly highly imbalanced towards low/no precipitation events, and becomes less imbalanced as larger fractions of low precipitation events are removed.

clustered, indicating a saturation near zero since precipitation values are non-negative. The right tail, in contrast, deviates strongly above the diagonal reference line, suggesting a heavy-tail characteristic, i.e., more high-precipitation events than expected under a normal distribution. As more of the low-end events are truncated, this deviation reduces, and the right tail begins to align more closely with the normal distribution. This indicates a reduction in skewness and a move toward a more balanced dataset.

While truncating the majority of low-precipitation events, such as retaining only the top 20th percentile, would yield the most balanced dataset in terms of distribution, it comes at a significant cost: the loss of a substantial number of meaningful precipitation events. Therefore, it is necessary to strike a balance between statistical balance and dataset richness. To guide this trade-off, we visually inspected random samples of events across different percentile ranges: 0–20, 20–40, 40–60, 60–80, and 80–100. This qualitative assessment revealed that events with average precipitation values below the 50th percentile generally exhibited little to no meteorological significance and could be safely discarded without compromising the diversity or relevance of the dataset. Based on this observation, we choose to retain only those events with average precipitation above the 50th percentile for all subsequent experiments. This subset includes data spanning from 2008 to 2018 and provides a more informative, balanced foundation for model training and evaluation. Random samples from each percentile range are visualised in Figure 4.4.

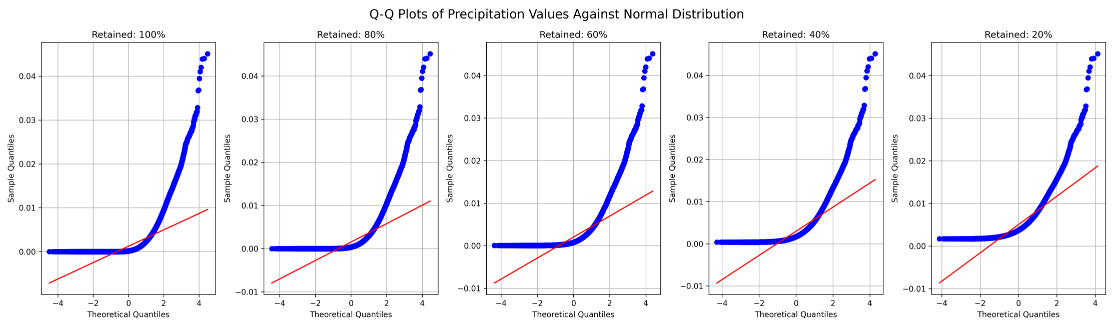


Figure 4.3: QQ plots of event average precipitation in the KNMI dataset, with truncation of increasing bottom-percentiles of data. The red line represents a perfect match with the theoretical distribution being considered- a standard normal. The left tail clearly shows a hard clamp at 0, since average precipitations are non-negative. The right tail shows that extreme high precipitation events are much fewer in number than low precipitation events, but are still far higher than expected in a standard normal. The plots clearly show significant deviation from standard normal behaviour, but more importantly, show that the data becomes less skewed (the right tail approaches the red line) as more data is truncated.

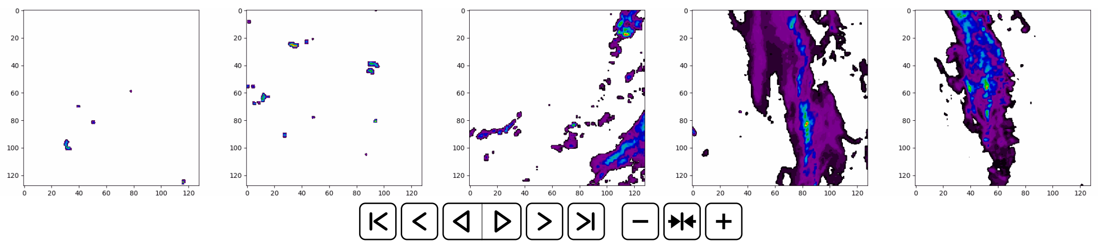


Figure 4.4: Events in different percentile blocks in KNMI. From left to right, 0-20 percentile,20-40,40-60,60-80 and 80-100. Clearly, the events with average precipitation under the 40th percentile have almost no activity. Events in the block 40-60 have significant activity, and a threshold of 50 percentile was decided upon.

4.2 SEVIR

4.2.1 Introduction to the Dataset

The SEVIR dataset, introduced in [16], is a machine-learning-ready resource that aggregates multiple remote sensing modalities, including satellite and radar data. It consists of 4-hour weather events covering 384 km $\times$ 384 km regions across the continental United States, sampled every 5 minutes. The spatial resolution is 1 km^2 for most modalities, including Vertically Integrated Liquid (VIL)[1].

SEVIR provides five sensing modalities: three channels from the GOES-16 (Geo-

[1]This is true for VIL. Spatial resolution varies slightly across modalities.

stationary Operational Environmental Satellite) system [36], VIL measurements, and data from the Geostationary Lightning Mapper (GLM) [37]. The GOES-16 channels include visible satellite imagery (available only during daylight hours) and infrared imagery at two wavelengths. VIL, which is derived from radar reflectivity, estimates the total liquid water content in a vertical column of the atmosphere and is commonly used to assess intense precipitation events such as thunderstorms and hail [38].

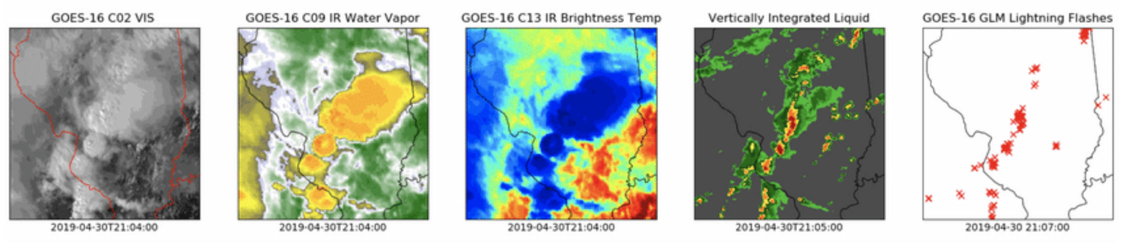
An example of a single frame across the available modalities is shown in Figure 4.5. For this study, we focus on the VIL modality, which is most directly comparable to the radar-derived precipitation estimates in the KNMI dataset and consistent with the modality used in our benchmarks. Multi-modal fusion and analysis are left for future work.

Events in SEVIR have been carefully curated through a systematic selection process, which marks a key distinction from the KNMI dataset. While KNMI consists of continuous radar frames captured every 5 minutes over 10 year, SEVIR contains discrete 4-hour events selected to balance data coverage and event diversity. The SEVIR dataset categorises events into two types: *random events* and *storm events.*

Random events are sampled by selecting random timestamps between 2017 and 2019. Event locations are determined by randomly sampled centres, with sampling probabilities biased toward regions of higher precipitation intensity. This approach is intended to avoid oversampling no-precipitation cases [16]. Storm events, on the other hand, are selected based on records from the National Centres for Environmental Information (NCEI) Storm Events Database[2]. This database provides detailed records of severe weather occurrences in the United States, including time, location, event type, and impact descriptions. SEVIR includes storm events recorded between 2017 and 2019 under the categories: Flood, Flash Flood, Hail, Heavy Rain, Lightning, Thunderstorm Wind, and Tornado. The authors of [16] clustered entries based on their time and spatial proximity (latitude and longitude), and selected the central point from each cluster to define an event. Figure 4.6 illustrates the spatial distribution of both storm and random events in SEVIR, along with their temporal coverage.

Since the dataset is already 'machine-learning ready', no particular preprocessing steps are necessary, which aligns with the pipelines of our benchmarks. Individual values of the data are stored between 0-254, and are retained as such without converting into raw VIL values. We do so for convenience, and also to suit the pipelines of our benchmarks. The original resolution of the data is 384×384, which we downsample to 128×128 for our pipeline.

[2] `https://www.ncdc.noaa.gov/stormevents/`



GOES-16 C02 VIS
2019-04-30T21:04:00
GOES-16 C09 IR Water Vapor
2019-04-30T21:04:00
GOES-16 C13 IR Brightness Temp
2019-04-30T21:04:00
Vertically Integrated Liquid
2019-04-30T21:05:00
GOES-16 GLM Lightning Flashes
2019-04-30 21:07:00

Figure 4.5: A frame across the five image modalities available in the SEVIR dataset [16].

4.2.2 Dataset Analysis

An in-depth analysis of the SEVIR dataset was conducted to better understand its underlying distribution and to assess whether additional data curation steps are necessary. It is important to highlight several key differences between SEVIR and the KNMI dataset, which have implications for data-driven weather prediction models.

First, SEVIR consists of curated events, each comprising a fixed number of frames, whereas the KNMI dataset contains continuous 5-minute interval recordings spanning a full decade. Second, as previously described, SEVIR events are carefully selected to include both significant storm events and randomly sampled cases, with a bias toward high-intensity precipitation. This curation aims to achieve a more balanced distribution of weather phenomena than that found in raw observational datasets. Third, while the KNMI dataset is geographically fixed to the Netherlands, SEVIR spans multiple regions across the continental United States.

This spatial diversity presents a challenge to data-driven weather forecasting methods. In particular, it violates the assumption that the underlying data distribution is stationary and spatially invariant. In practice, this assumption does not

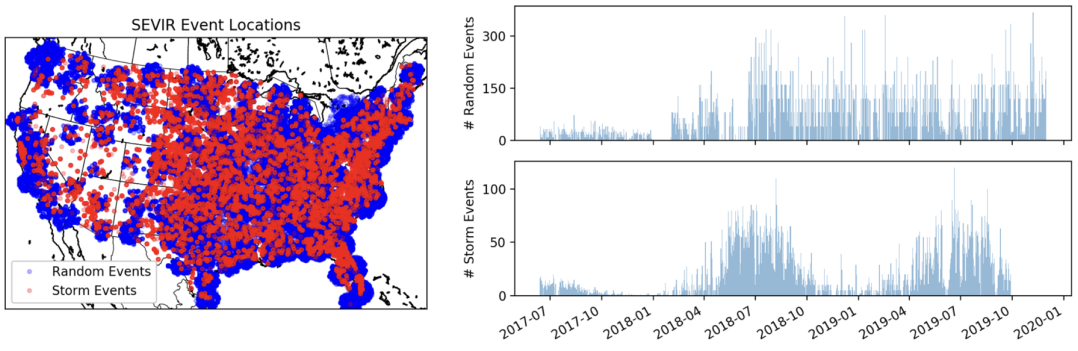


Figure 4.6: (Left) Spatial map of SEVIR events, both random and storm events. (Right) Distribution of SEVIR events over time [16].

hold: as shown in Figure 4.6, SEVIR includes events sampled from both inland regions and coastal areas, whose weather is governed by different meteorological factors. Additional geographical factors, such as topography and regional climate, further contribute to the variation in the dataset.

As with the KNMI dataset, we use violin plots and quantile–quantile (QQ) plots to analyse the underlying distribution of the SEVIR dataset. Each event is characterised by its average precipitation value across both space and time. These summary statistics are visualised using violin and QQ plots to understand the skewness and variance of the data. To enable a direct comparison with the KNMI dataset, we also examine how the distribution evolves when events with the lowest average values are progressively truncated. That is, we visualise the data distribution under various retention percentiles.

The violin plot in Figure 4.7 reveals that, while the SEVIR dataset remains skewed, the degree of skewness is substantially lower than in the KNMI dataset. This is expected, as SEVIR was designed with distributional balance in mind, as discussed in the event selection procedure. Truncating events with low average precipitation further improves the balance, which is reflected in the increased separation of quantile lines within the violin plot. In contrast, Figure 4.2 shows that the quantile lines for the KNMI dataset are tightly clustered, indicating only minor differences between the 20th, 40th, 60th, and 80th percentiles. In SEVIR, as shown in Figure 4.7, these lines are more widely spaced, suggesting a more balanced distribution.

Similar insights are confirmed by the QQ plot in Figure 4.8. While the SEVIR dataset still deviates from a standard normal distribution and retains a heavy-tailed shape, its deviation is less pronounced than that of the KNMI dataset. Both the violin and QQ plots demonstrate that excluding the lowest percentile events leads to a more symmetric and balanced distribution.

A qualitative inspection of random SEVIR events (Figure 4.9) supports this conclusion. Notably, events above the 20th percentile already exhibit significant precipitation activity. Therefore, although truncating the lowest 20% of events provides only marginal gains in distributional balance, we choose not to discard them. This decision ensures consistency with prior work, which achieved state-of-the-art performance on the SEVIR dataset and included these events.

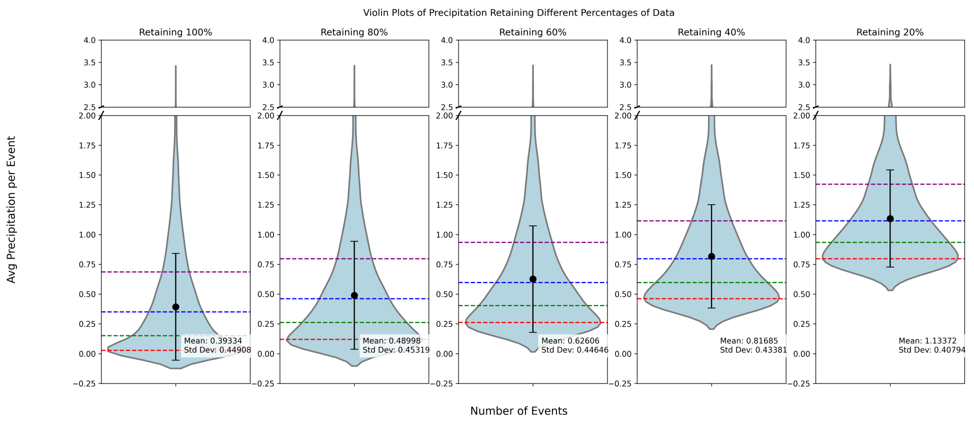


Figure 4.7: Violin plots of event average precipitation in the SEVIR dataset, with truncation of increasing bottom-percentiles of data. Horizontal lines represent the 20^{th}, 40^{th}, 60^{th} and 80^{th} percentiles. Compared to the violin plots of the KNMI dataset, the spacing between the lines indicates a better distribution balance. The dataset becomes more balanced when truncating larger portions of the low precipitation events. However, to remain consistent with prior work and the relatively better balance of the full dataset, we choose not to truncate any data.

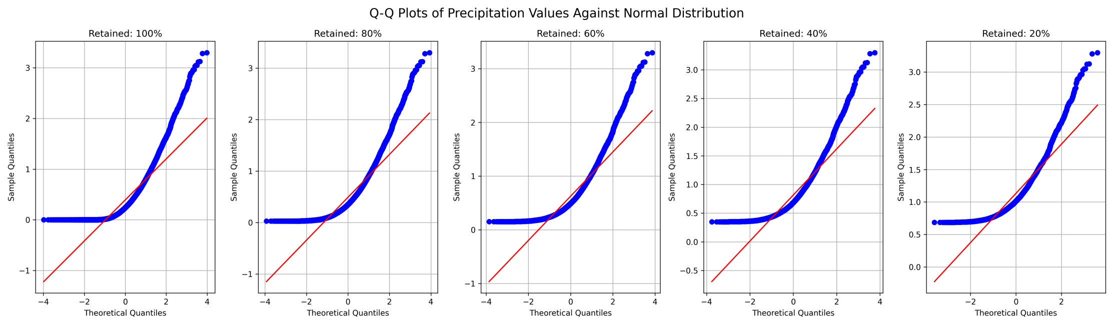


Figure 4.8: QQ plots of average event precipitation in SEVIR, with truncation of increasing bottom-percentiles of data. The event distribution is compared to a theoretical standard Gaussian distribution. Clustering of events at the left tail is once again observed due to precipitation values being clamped at 0. The distribution is much less skewed than the KNMI dataset since the right tail is much closer to the red line, although it still significantly deviates from standard normal behaviour.

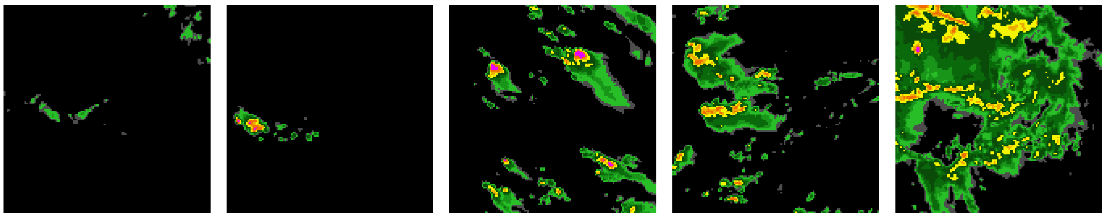


Figure 4.9: Events in different percentile blocks in SEVIR. From left to right, 0-20 percentile,20-40,40-60,60-80 and 80-100. Compared to the KNMI dataset, significant activity is observed in the $20 - 40^{th}$ percentile block.

Experiments and Results 5

This chapter presents the experimental evaluation of our models on two benchmark datasets: the KNMI and SEVIR datasets. We assess model performance using the evaluation metrics defined in the previous chapter. As outlined in Chapter 3, our objective is to learn a mapping from context frames to future targets, i.e., $\mathcal{X}_{context} \mapsto \mathcal{X}_{target}$, where $\mathcal{X}_{context} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{T_c}\}$, and $\mathcal{X}_{target} = \{\mathbf{X}_{T_c+1}, \ldots, \mathbf{X}_T\}$.

We design two experimental setups, varying the values of T_c, T and the temporal resolution between frames, to test our model's ability to generalise and learn temporal dynamics under diverse conditions.

In the first setting, we choose $T_c = 3$ and $T = 9$ with a temporal resolution of 30 minutes between frames. The task here is to predict future precipitation fields at 30-minute intervals for the next 3 hours, using 1 hour of past context. Specifically, the model is provided with observations at the current time, 30 minutes prior, and 60 minutes prior, and must predict the next six frames at +30, +60, +90, +120, +150, and +180 minutes. This constitutes a long-term forecasting task with relatively sparse context.

In the second setting, we use $T_c = 5$ and $T_c = 20$, with a 5-minute temporal resolution. In this case, the model is tasked with forecasting the next 100 minutes using 25 minutes of past data. While the total forecast horizon is shorter than in the first task, the number of steps to predict increases to 20, which stresses the model's ability to maintain accuracy over many autoregressive rollouts.

These two configurations were chosen to complement each other: the first makes the model predict farther in time, but predict fewer frames. The second, however, requires the model to learn shorter dynamics, but predict 20 frames. Since both of our proposed models operate in an autoregressive manner, these setups allow us to investigate the degree to which prediction quality degrades over time, a known limitation of autoregressive models due to error accumulation. Additionally, both configurations align with prior work, enabling fair comparisons across baselines.

Our baselines, as discussed in Chapter 2, are NowcastingGPT, a token-by-token autoregressive transformer model which achieved state-of-the-art prediction results on the KNMI dataset, and Diffcast-Phydnet, a diffusion-based residual prediction model which models the residual between a deterministic backbone

(PhyDnet) and the ground truth. The organization of this chapter is as follows: first, we present results on the two tasks in the first two sections. In section 3, we present extended evaluations on the KNMI dataset- such as evaluation in catchment zones as described earlier in the data analysis chapter, and testing how BlockGPT's performance varies in different seasons. Finally, in section 4, we investigate in-detail, the inductive bias of the frame-by-frame generation method.

5.1 Task 1: 3-Hour Prediction with 1-Hour of Context

5.1.1 Results on the KNMI Dataset

Overall Performance

We evaluate predictive performance using the continuous metrics (MSE, MAE, PCC) introduced previously. Figure 5.1 compares these metrics for our two proposed models against the baselines. Along with the values of the metrics as a function of time, we also report the percentage difference in model performance to make comparison easier and more concrete.

On average (across seeds), BlockGPT consistently outperforms both baselines on MSE, MAE and PCC. On MSE, BlockGPT starts about 15% better than NowcastingGPT and Diffcast, but is only 5% better at the 3-hour mark. This is expected since, over time, errors accumulate owing to its autoregressive nature. On MAE, BlockGPT starts about 8% better than NowcastingGPT, and is still just as good at the 3-hour mark, although there is an intermediate dip in the performance gap. Thus, compared to NowcastingGPT, MSE performance degrades more quickly, while MAE degrades at a similar rate. This could mean that in BlockGPT, the degradation over time causes outliers, resulting in high MSE but relatively low MAE. Compared to Diffcast, BlockGPT has a 15% performance gap to Diffcast, but this degrades to only about 5% after 3 hours. On MAE, its performance gap to Diffcast varies between 1 and 4% only. However, it must be conceded that these absolute comparisons on MSE and MAE are not statistically relevant, since the standard deviation across seeds is high in our models. Thus, we cannot say with statistical certainty that we outperform our benchmarks on MSE and MAE.

Analysing PCC, the advantage of BlockGPT is clear- on average, it starts about 20% better than the baselines, and the performance edge increases to about 60% at the end of 3 hours. Taking into account the standard deviation across seeds also paints a better picture than MSE and MAE. Despite having a large deviation, we are still better than Diffcast and NowcastingGPT. This clearly shows the advantage of generating one frame at a time- a larger correlation to the ground truth indicates that patterns are better captured via the more accurate inductive bias of letting tokens in a frame attend bidirectionally. NowcastingGPT (and Diffcast) both degrade at a much faster rate than BlockGPT, further attesting to this conclusion.

To summarise, although our proposed model achieves competitive performance compared to the benchmarks, on the continuous metrics, we can say with statistical certainty that performance is better on PCC. On MSE and MAE, although BlockGPT is better on average across seeds, it exhibits a large deviation. The strong performance of BlockGPT on PCC shows the clear benefit of generating frame-by-frame. Mainly, the advantage is better correlation with ground truth since frames can attend bidirectionally (and hence capture patterns better), and fewer roll-out steps, which accumulate smaller error. It is also clear, on PCC, that NowcastingGPT's performance degrades at a faster rate than BlockGPT. This faster degradation is also a common trend observed in later results.

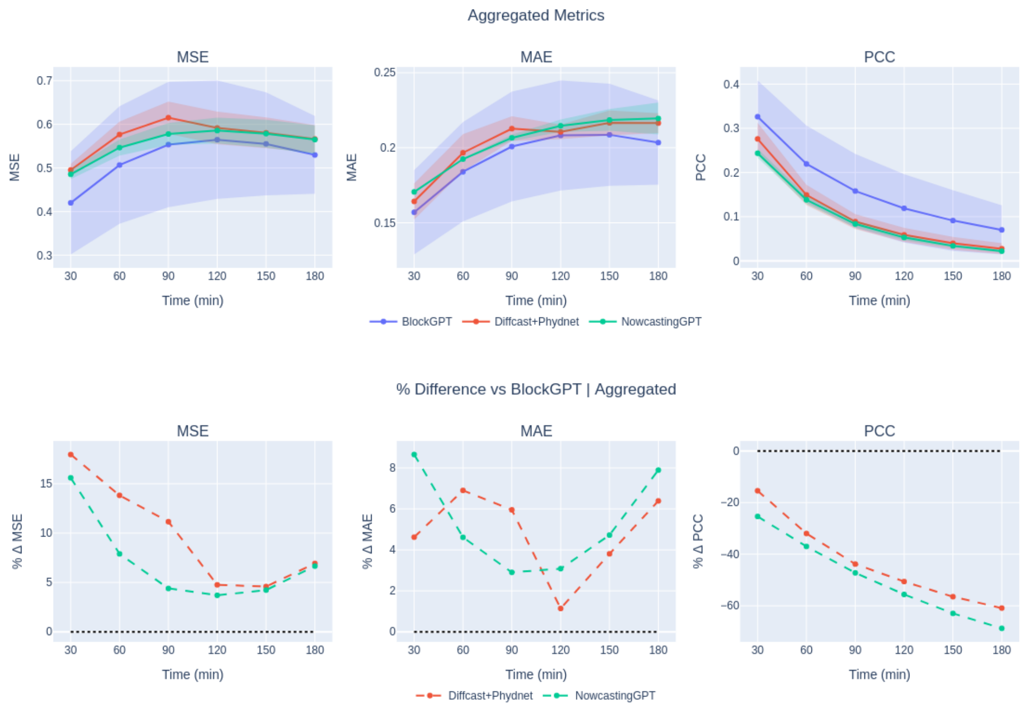


Figure 5.1: Continuous Metrics (MSE, MAE and PCC) on the KNMI dataset (task 1, predict 3 hours given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds. Top: Absolute values of the metrics of each model as a function of time. Bottom: metric values relative to BlockGPT, in percentage deviation.

The CSI metric, visualised in Figure 5.2 captures pixel-level performance at different thresholds at 1mm, 2mm and 8mm rainfall. The proposed model, BlockGPT, outperforms the benchmarks across all time steps and thresholds. A higher CSI indicates that the model is more skilful at correctly forecasting precipitation exceeding that intensity, showing that it captures more true events while minimising both misses and false alarms. On average results across seeds, BlockGPT outperforms NowcastingGPT and Diffcast by approximately 30% and 40%, respectively, on the 1mm threshold, at the 3-hour mark. Similar levels of improvement are observed on the 2mm threshold. On the 8mm threshold, BlockGPT shows a 40% to 80% improvement over NowcastingGPT, and up to a 40% improvement over Diffcast. Further, we can also claim that these results are statistically significant, since our performance is better than the benchmarks even after taking into account the standard deviation across seeds. Once again, the benefit of having to roll out fewer steps during inference is clear- the performance gap between NowcastingGPT and BlockGPT increases with time.

Similar trends are reflected in the FAR metrics, reported in Figure 5.3. On average, BlockGPT outperforms the benchmarks, with a 15% improvement on the 1mm threshold in the first time step, but this degrades to just about 5% at the 3-hour mark. Similar performance improvements are more or less similar to the other models. Taking the standard deviation of models across seeds, we also show that the performance gain is statistically significant. The trend of performance gaps increasing concerning NowcastingGPT is not true on FAR, across all thresholds. In fact, over time, their performance gap decreases. An important observation across all models is that the models degrade hugely with respect to CSI and FAR on their absolute values. CSI values approach almost zero at the three-hour mark, and FAR values are near 1. This clearly shows that these models degrade over time, with errors accumulating across time steps. The performance gap likely decreases because all models reach the extreme values of FAR, and struggle equally by overreporting extreme values.

On Task 1 of the KNMI dataset, BlockGPT, on average, outperforms both benchmark models, NowcastingGPT and Diffcast, across all evaluation metrics: MSE, MAE, PCC, CSI, and FAR. However, we can only say with statistical certainty that BlockGPT is better only on PCC, CSI and FAR. While the performance gap is relatively modest on MSE, a consistent and widening gap is observed across the other metrics, particularly in PCC and CSI, as the forecast horizon increases. This trend highlights a key architectural advantage of BlockGPT: its frame-level autoregressive structure, which requires fewer autoregressive rollouts compared to token-level models like NowcastingGPT.

Fewer rollouts reduce the accumulation of error over time, resulting in more stable and coherent long-term predictions. Additionally, the stronger performance on structure-aware metrics such as PCC and CSI suggests that BlockGPT captures the global spatiotemporal dynamics more effectively. These improvements are not merely the result of architectural differences but reflect a superior inductive bias-one that treats entire frames as single prediction units, preserving both consistency and semantic integrity across time. BlockGPT's better absolute scores and growing relative advantage over token-based baselines underscore its robustness and suitability for structured nowcasting tasks like those found in the KNMI dataset.

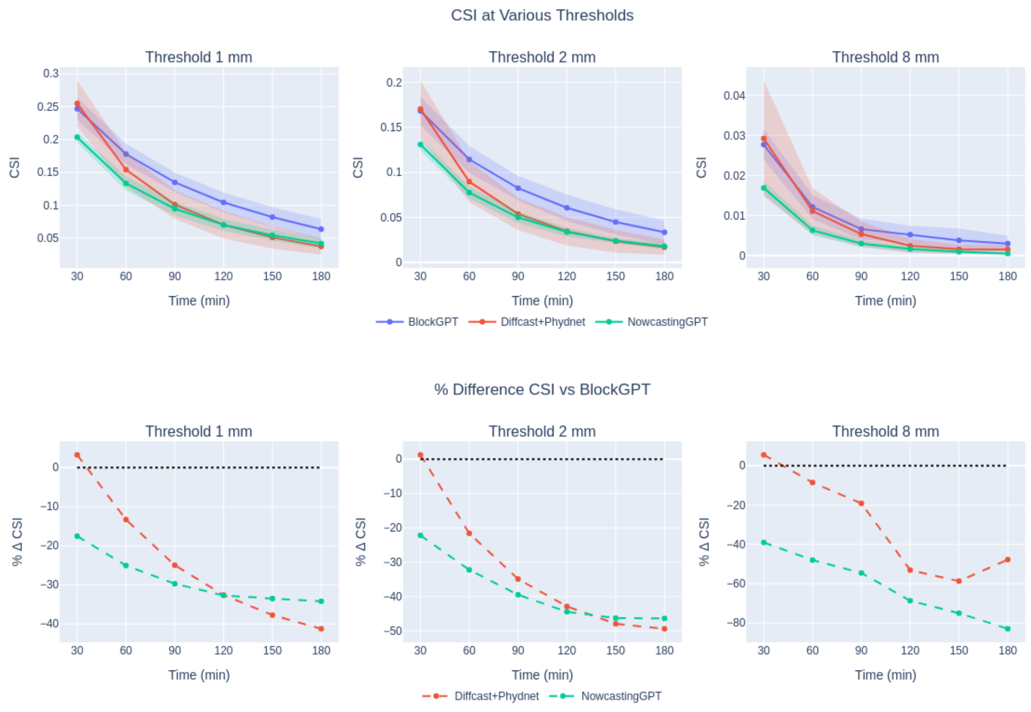


Figure 5.2: CSI on the KNMI dataset (task 1, predict 3 hours given 1 hour of context). Top: Absolute values of the metrics of each model as a function of time. Solid line and shading indicate average and standard deviation across seeds. Bottom: metric values relative to BlockGPT, in percentage deviation.

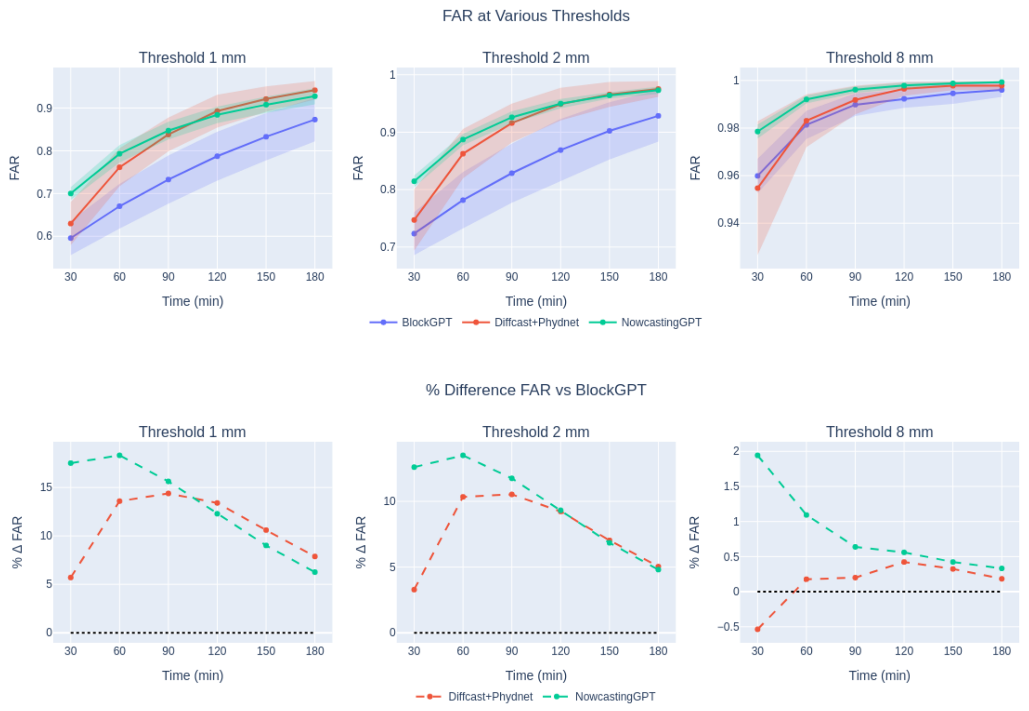


Figure 5.3: FAR on the KNMI dataset (task 1, predict 3 hours given 1 hour of context). Top: Absolute values of the metrics of each model as a function of time. Solid line and shading indicate average and standard deviation across seeds. Bottom: metric values relative to BlockGPT, in percentage deviation.

Performance on Higher Levels of Rainfall

In the data analysis section, we defined different precipitation blocks based on the average precipitation in each event. These blocks were then defined based on the average event percentiles. In this section, we especially analyse the performance on the last block, the $80 - 100^{th}$ block. [1] This analysis is important because extreme events cause larger damage, both financially and impact on lives. Thus, it warrants special attention. We split the block into two, from $80 - 95$ to analyse high precipitation, and from $95 - 100$ to analyse extremes. It is important to make a distinction between the analysis in extreme percentile blocks and categorical metrics on higher thresholds. The former defines whole events, which are extreme (since it defines extremes based on event-average precipitation), and the latter defines pixel-wise extremes.

Figures 5.4 and 5.5 show the MSE, MAE and PCC in the 80-95 and 95-100 percentile blocks, respectively. In both blocks, trends follow similarly to the overall results, with BlockGPT, on average, outperforming the benchmarks on all metrics.

[1]In the appendix, plots visualising the metrics in each percentile block will be displayed.

Interestingly, the performance gap is higher in the extreme (95-100) block, across all extremes, clearly visible in the figure. However, the performance gap is statistically relevant only on PCC. Note that we only report the continuous metrics in the extreme blocks of precipitation. CSI and FAR are already inclusive of a hierarchy-larger thresholds test performance pixel-wise on higher amounts of precipitation. Since CSI and FAR on higher thresholds indicate pixel-wise performance across the whole dataset already, we do not report it exclusively in these extreme blocks (which are defined on the entire event).

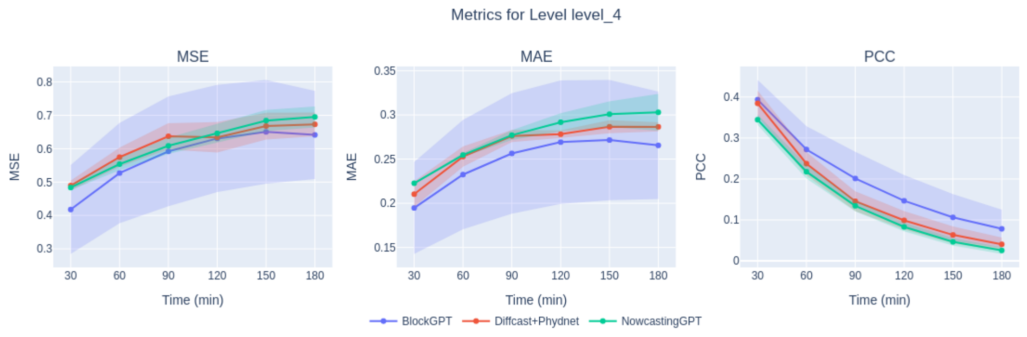


Figure 5.4: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing high rainfall. Solid line and shading indicate average and standard deviation across seeds.

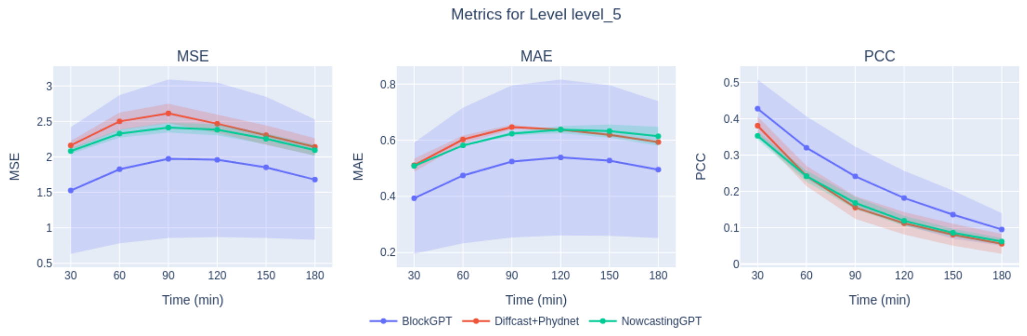


Figure 5.5: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing extreme rainfall. Solid line and shading indicate average and standard deviation across seeds.

5.1.2 Results on the SEVIR dataset

Overall Performance

An analysis of the continuous metrics in Figure 5.6 reveals that on the SEVIR dataset, it is Diffcast that has an edge over BlockGPT, on the continuous metrics. This is especially true on MSE and MAE, but less so on PCC, where there is only a small performance gap. Since PCC is scale-invariant, it allows for direct comparisons across datasets. One clear observation is that all models achieve substantially higher PCC scores on the SEVIR dataset compared to the KNMI dataset. This suggests strongly that the frame-to-frame difference in both datasets varies, even though they are of the same temporal resolution. Since a higher PCC indicates that the model can capture the trends of the frames better, we may hypothesise that frames in SEVIR have easier dynamics and have relatively lower variation from frame to frame.

BlockGPT has the worst MSE and MAE among all benchmarks. While it begins slightly better than NowcastingGPT, it accumulates more error over time and performs worse beyond the 90-minute mark. At the 3-hour point, on average, BlockGPT is nearly 40% worse than Diffcast on MSE and about 20% worse on MAE. However, it achieves better PCC than NowcastingGPT, starting off with a performance gap of about 10%, and increases to 30% after 3 hours. It is only slightly worse than Diffcast+PhyDnet, with a gap of just a few percentage points, which increases to about 10% at the end of 3 hours. These results are also statistically significant, taking into account the standard deviation of the models across seeds. On MSE and MAE, we see that BlockGPT's performance gap to NowcastingGPT worsens, unlike the KNMI dataset, where the trend worsened only on MSE. This means that on the SEVIR dataset, as time progresses, BlockGPT not only has more outliers but more errors in general (on average). On PCC, however, like on the KNMI dataset, NowcastingGPT's performance degrades faster over time.

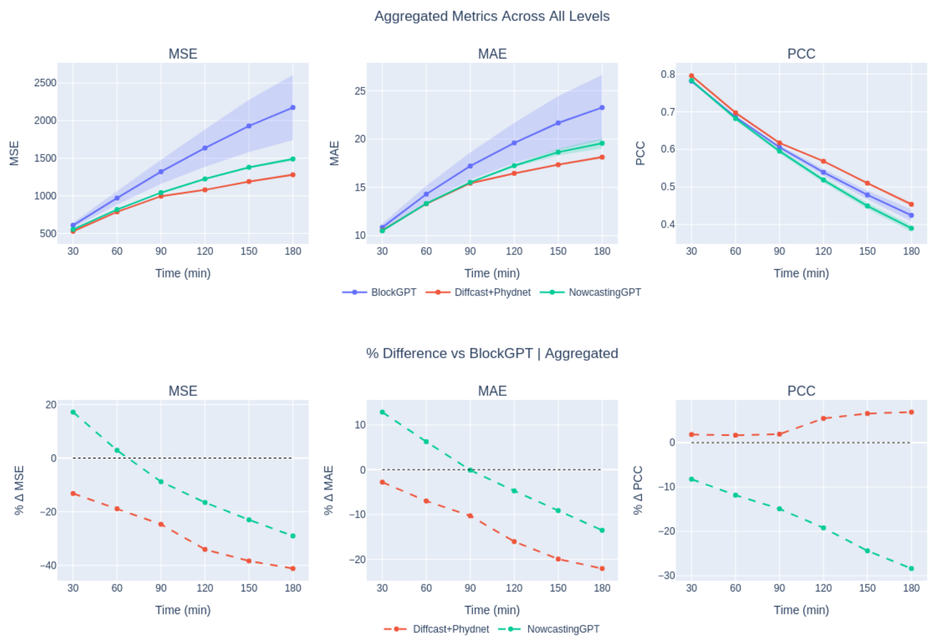


Figure 5.6: Continuous Metrics (MSE, MAE and PCC) on the SEVIR dataset (task 1, predict 3 hours given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds. Top: Absolute values of the metrics of each model as a function of time. Bottom: metric values relative to BlockGPT, in percentage deviation

The CSI plots across different thresholds reveal a more promising picture. Absolute values are presented in Figure 5.7, while relative differences of the benchmarks with respect to BlockGPT are shown in Figure 5.8.

BlockGPT delivers strong results. On average, it consistently outperforms NowcastingGPT across all thresholds, with the largest improvement exceeding 60% on threshold 181, sustained across all time steps. Over the three lower thresholds, performance gaps with respect to NowcastingGPT vary from over 10% to 50%. Again, as the time step increases, the BlockGPT's performance advantage increases. On the other hand, its advantage over Diffcast becomes more apparent at higher thresholds, where it shows at least a 20% improvement over time on the last three thresholds. We can claim that these results are statistically significant, since the performance gap remains despite taking into account the standard deviation across seeds. Once again, BlockGPT outperforms NowcastingGPT and Diffcast, at least on categorical metrics, further demonstrating the effectiveness of its design.

FAR results follow a similar pattern in favour of BlockGPT. Absolute values are shown in Figure 5.9, while relative differences with respect to BlockGPT are presented in Figure 5.10.

BlockGPT exhibits the most favourable FAR values starting from the second threshold onwards, consistently achieving the lowest FAR across all models. Its performance advantage is particularly pronounced over NowcastingGPT, with a 40% improvement sustained over time on thresholds 74 and 133. Although on average, the performance gaps are narrow at higher thresholds, BlockGPT continues to outperform all other models, reaffirming its effectiveness on categorical metrics. Taking into account the standard deviation across seeds, BlockGPT's performance gap on FAR is statistically significant from the second threshold onwards.

To summarise our model's performance, BlockGPT underperforms primarily on the categorical metrics, which measure per-pixel(averaged over the event) absolute and squared differences between predicted and ground truth fields. These metrics mean that, on average, BlockGPT is dominated by small errors across all pixels. However, BlockGPT demonstrates consistently strong performance on other evaluation metrics—namely, Pearson Correlation Coefficient (PCC) (only compared to NowcastingGPT), Critical Success Index (CSI), and False Alarm Rate (FAR).

High PCC scores indicate that BlockGPT accurately captures the spatiotemporal structure and variability of precipitation patterns, even if there are small deviations in pixel intensity. Stronger CSI values (which are per-pixel), particularly at higher precipitation thresholds, show that the model excels at detecting localised and high-impact weather events, which are typically more relevant in operational nowcasting. At the same time, lower FAR values suggest that BlockGPT maintains a conservative prediction strategy with fewer false positives, especially under extreme conditions.

In all, these results suggest that while BlockGPT may be slightly less precise at matching exact pixel intensities, it is more effective at capturing the overall spatial dynamics and critical decision-relevant aspects of precipitation systems. This reinforces the strength of BlockGPT's frame-level autoregressive architecture, which enables coherent and reliable forecasts without the noise accumulation often seen in token-level or recurrent models.

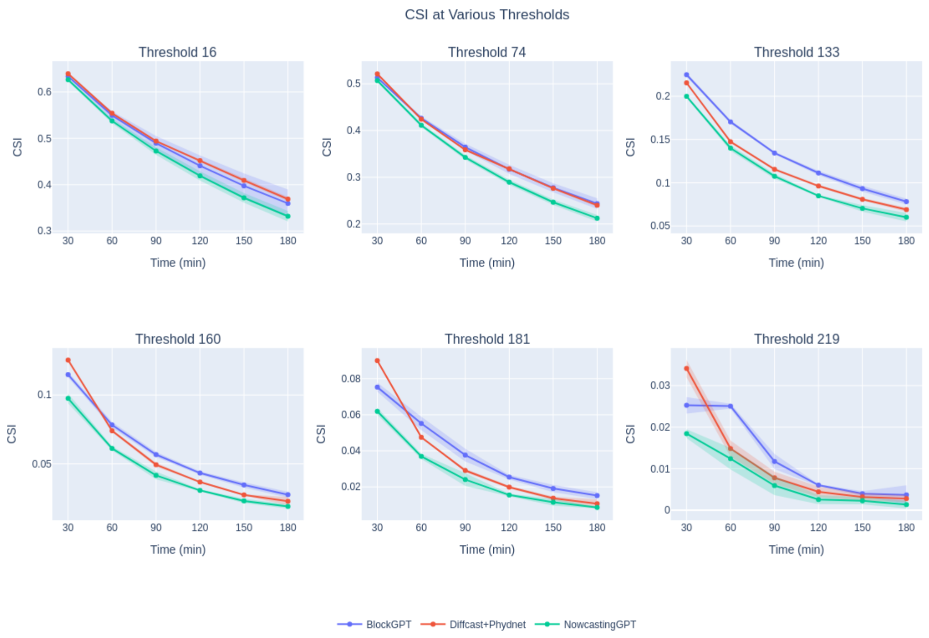


Figure 5.7: CSI values evaluated at different thresholds on the SEVIR dataset (task 1: predicting 3 hours, given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds.

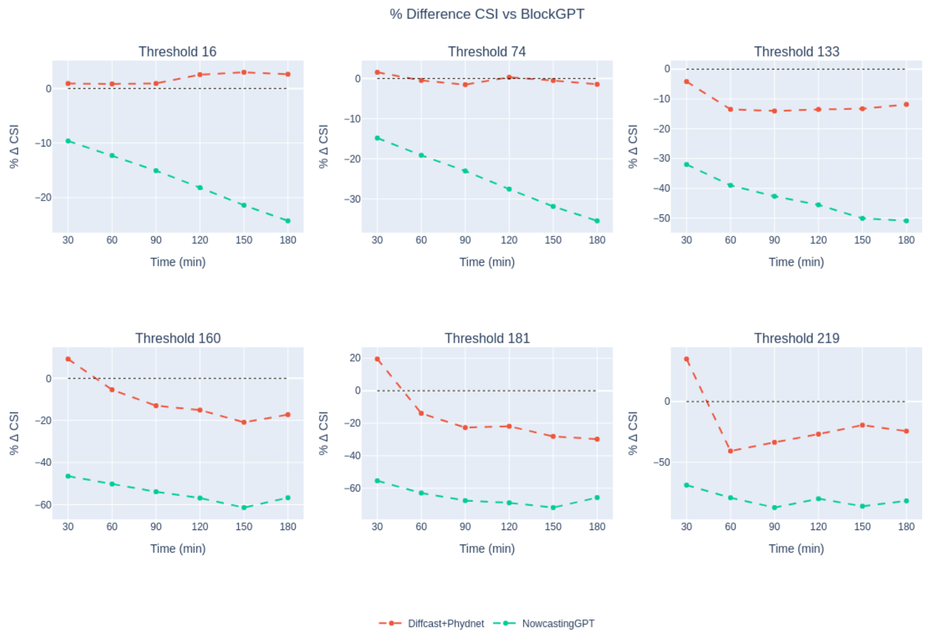


Figure 5.8: Relative difference between CSI values with respect to BlockGPT.

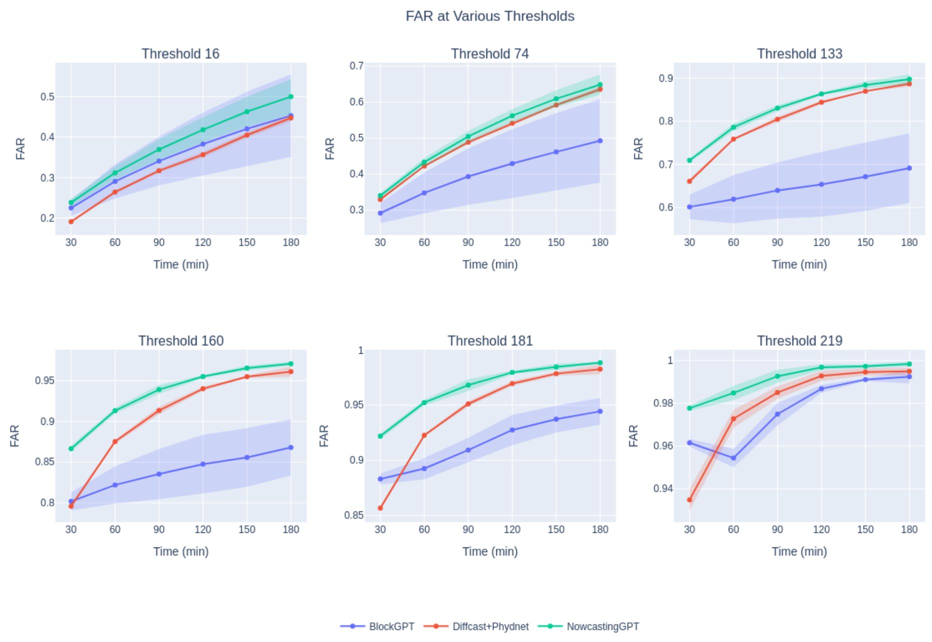


Figure 5.9: FAR values evaluated at different thresholds on the SEVIR dataset (task 1: predicting 3 hours, given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds.

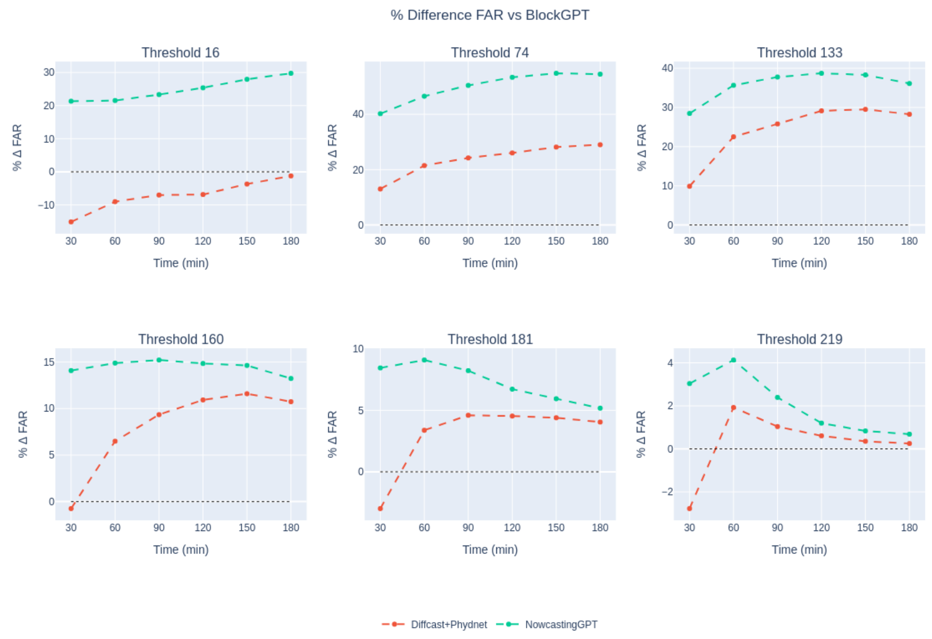


Figure 5.10: Relative difference between FAR values with respect to BlockGPT

Performance on Higher Levels

Results on the higher precipitation thresholds in the SEVIR dataset, shown in Figures 5.11 and 5.12, largely mirror the trends observed in the global performance metrics. On the MSE and MAE metrics, BlockGPT underperforms relative to both NowcastingGPT and DiffCast, indicating that it is less effective in minimising pixel-wise absolute and squared differences, even under high-intensity scenarios. However, on PCC, which evaluates structural correlation, BlockGPT outperforms NowcastingGPT and performs comparably to DiffCast, albeit with slightly lower values.

More notably, BlockGPT demonstrates superior performance on CSI and FAR (which analyse pixel-wise extremes) across these higher thresholds, suggesting that it is more effective at detecting localised and extreme precipitation events while maintaining a lower false alarm rate. These findings emphasise the strength of BlockGPT's inductive bias, which appears to generalise well even in the tails of the precipitation distribution. In particular, when compared to NowcastingGPT, BlockGPT shows a clear advantage in structure and threshold-sensitive metrics, reinforcing its robustness in forecasting extremes. Although it does not surpass DiffCast in PCC, its higher CSI and lower FAR scores indicate a stronger balance between detection and precision in high-impact forecasting scenarios.

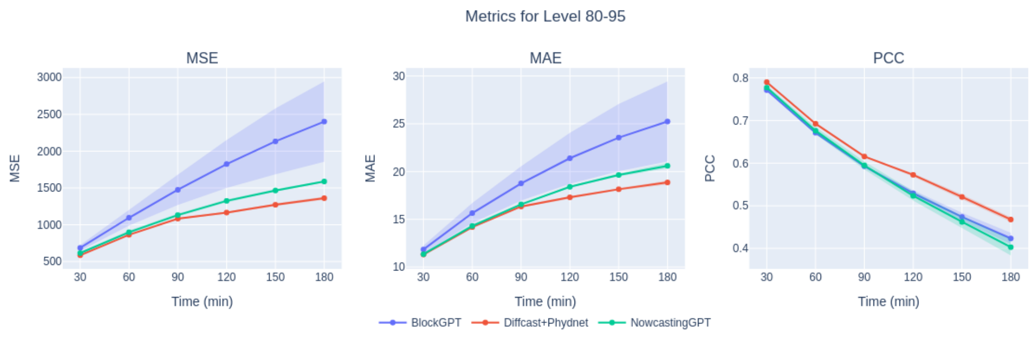


Figure 5.11: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing high value events, but not extremes. Solid line and shading indicate average and standard deviation across seeds.

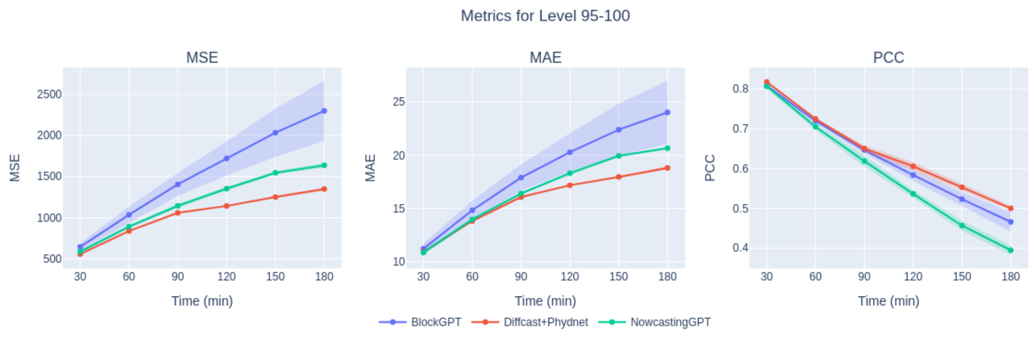


Figure 5.12: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing extreme events. Solid line and shading indicate average and standard deviation across seeds.

5.2 Task 2: 100-Minute Prediction with 25 Minutes of Context

This task involves predicting 100 minutes into the future using only 25 minutes of context, with frames sampled at a 5-minute temporal resolution. Compared to coarser settings, this setup introduces a distinct challenge. At such a fine resolution, consecutive frames exhibit only subtle changes, allowing models to more easily capture short-term dynamics. However, this advantage comes with a new difficulty: modelling fine-grained details. Transformer-based prediction models, especially when paired with visual tokenisers, are known to suffer from blurring effects in their outputs [12]. These models tend to excel at capturing the overall structure of precipitation patterns but often struggle with fine details. Thus, the prediction task can be conceptually divided into two stages: first, modelling the coarse structure, and second, refining the fine details.

In contrast to the previous task, which used a 30-minute resolution and primarily tested the model's ability to capture coarse shapes, this 5-minute resolution task emphasises the model's ability to preserve and predict fine-scale details. Like the previous section, we first introduce results on the KNMI dataset, and then the SEVIR dataset.

5.2.1 Results on the KNMI dataset

Overall Results

This section presents the overall results on the KNMI dataset for the 5-minute resolution prediction task, comparing BlockGPT against two baselines: NowcastingGPT and Diffcast. While BlockGPT underperforms on continuous metrics, it consistently outperforms both benchmarks on categorical metrics, indicating different strengths depending on the evaluation criterion.

The results for continuous metrics, MSE, MAE and PCC are visualised in Figure 5.13. On MSE, BlockGPT performs significantly worse than the baselines, with an average degradation of approximately 80% compared to both NowcastingGPT and Diffcast. Similarly, on MAE, BlockGPT is 20% worse than NowcastingGPT and 60% worse than Diffcast. As before, results are reported based on the average results across seeds. The standard deviation of the results suggests small variation, and results are thus statistically significant.

However, the results on PCC reveal a more nuanced picture. While BlockGPT consistently underperforms NowcastingGPT by over 50% across all time steps, it outperforms Diffcast on average by 50%, and this gap increases to over 100% in the final time steps. This suggests that BlockGPT, despite poor absolute accuracy, retains some ability to capture the correlation structure of rainfall patterns over time, especially when compared to Diffcast.

Overall, these results mirror trends observed earlier on the SEVIR dataset for the 30-minute resolution task: BlockGPT is consistently weaker when evaluated on precise rainfall amounts. This points to a limitation in its ability to regress exact values of precipitation. However, continuous metrics alone do not fully capture a model's practical utility, especially in flood-risk contexts where detecting events above certain thresholds is more critical than estimating precise intensities.

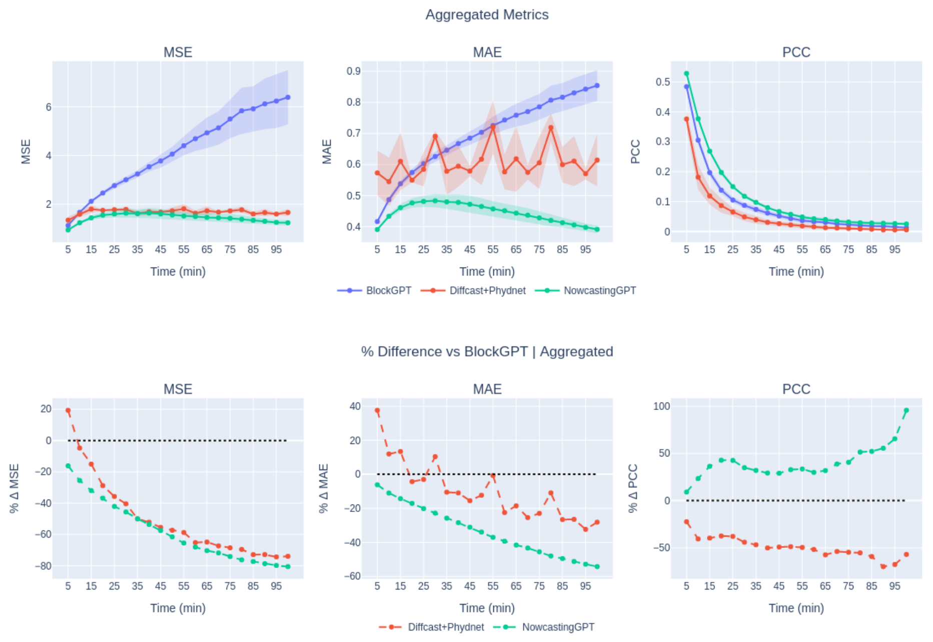


Figure 5.13: Continuous Metrics (MSE, MAE and PCC) on the KNMI dataset (task 2, predict 100 minutes given 25 minutes of context). Solid line and shading indicate average and standard deviation across seeds. Top: Absolute values of the metrics of each model as a function of time. Bottom: metric values relative to BlockGPT, in percentage deviation.

On categorical metrics, BlockGPT demonstrates strong performance, particularly on CSI across all evaluated thresholds (1mm, 2mm, and 8mm), as shown in Figure 5.14. At the 1mm threshold, on average, it is approximately 40% better than Diffcast in early forecast steps, with the advantage narrowing to 20% by the final timestep. Compared to NowcastingGPT, BlockGPT begins to outperform from the 30-minute mark onward, with the performance gap increasing to 60% by the final timestep. Similar trends are observed at the 2mm and 8mm thresholds, where improvements over both benchmarks grow steadily from 20% to 60% over time. Taking into account the standard deviation across seeds, the performance gaps BlockGPT exhibits are statistically significant. These results highlight BlockGPT's growing robustness over time for categorical event detection, especially under more extreme thresholds.

In terms of FAR, shown in Figure 5.15, BlockGPT again leads, though with smaller margins. At the 1mm threshold, on average, the performance gap over NowcastingGPT grows to about 10% by the final timestep, while the difference with Diffcast remains negligible. For the 2mm threshold, the margin over both benchmarks steadily increases to 10%. At the 8mm threshold, however, the advantage is modest; BlockGPT outperforms both baselines by only 2–4% on average. This suggests that while BlockGPT is effective in reducing false positives, its gains are more pronounced at lower thresholds. We can also say with statistical certainty that BlockGPT has better FAR by looking at the standard deviation across different seeds.

In summary, BlockGPT displays a clear trade-off in performance depending on the type of metric. On continuous metrics such as MSE and MAE, BlockGPT underperforms significantly, indicating difficulty in predicting precise rainfall intensities. On correlation-based metrics like PCC, its performance is mixed, surpassing Diffcast while lagging behind NowcastingGPT. On categorical metrics like CSI and FAR, BlockGPT consistently outperforms both benchmarks, particularly at higher thresholds and later time steps. These results reinforce the interpretation that BlockGPT is more effective as a classifier of extreme weather conditions than as a regressor of exact rainfall amounts. Its performance gains on categorical evaluations suggest it is especially suited for decision-making tasks (e.g., flood warnings) where detecting the occurrence of heavy precipitation is more critical than quantifying its exact value.

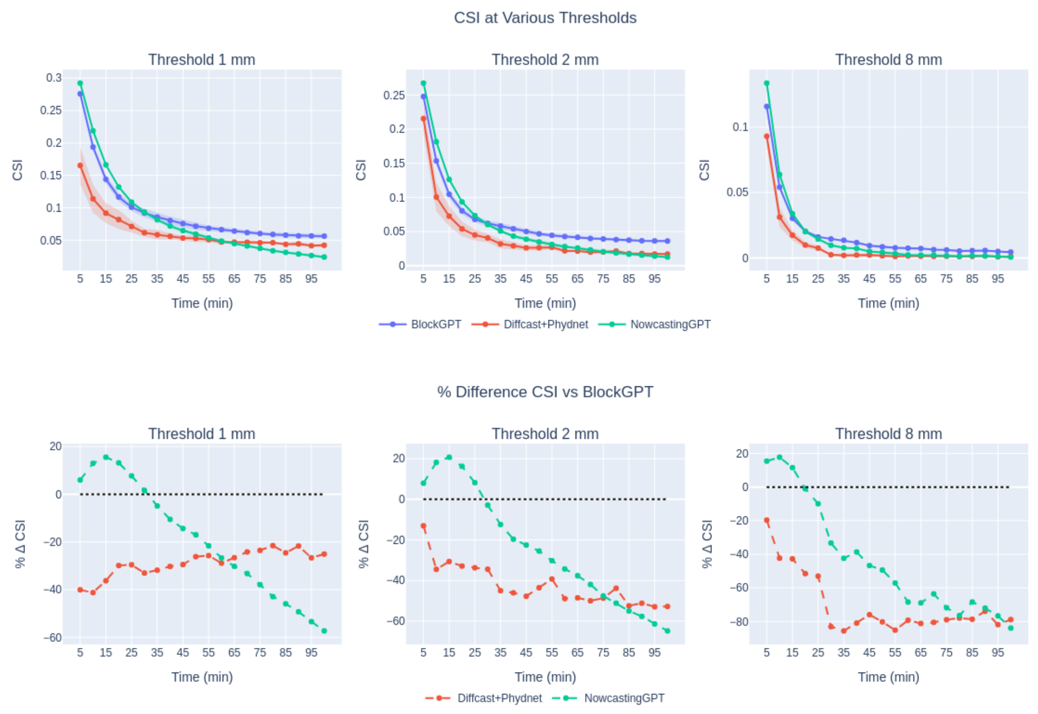


Figure 5.14: CSI on the KNMI dataset (task 2, predict 100 minutes given 25 minutes of context). Top: Absolute values of the metrics of each model as a function of time. Solid line and shading indicate average and standard deviation across seeds. Bottom: metric values relative to BlockGPT, in percentage deviation.

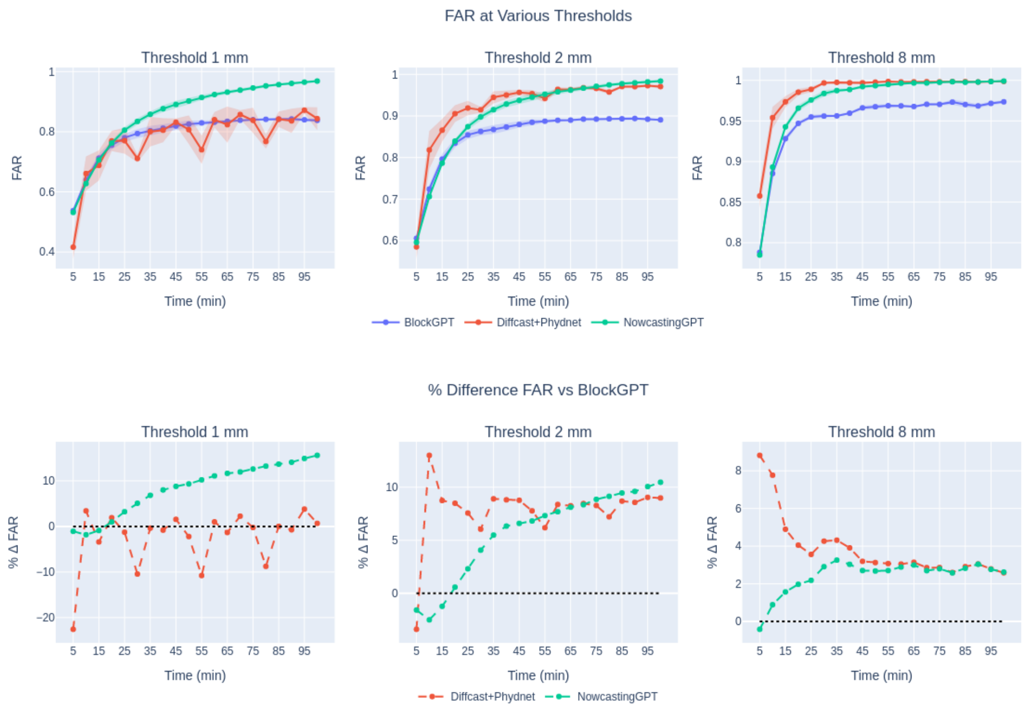


Figure 5.15: FAR on the KNMI dataset (task 2, predict 100 minutes given 25 minutes of context). Solid line and shading indicate average and standard deviation across seeds. Top: Absolute values of the metrics of each model as a function of time. Bottom: metric values relative to BlockGPT, in percentage deviation.

Performance on Higher Levels

The evaluation of model performance at higher precipitation levels on the KNMI dataset, focusing only on regions and time steps with heavier rainfall, is presented in Figures 5.16 and 5.17.

Across all blocks and continuous metrics, the trends remain consistent with the overall performance observed earlier. Specifically, BlockGPT continues to underperform relative to NowcastingGPT on Mean Squared Error (MSE) and Mean Absolute Error (MAE), indicating that it struggles to estimate precise rainfall magnitudes even in regions with substantial precipitation. However, a more favourable pattern emerges for Pearson Correlation Coefficient (PCC): BlockGPT consistently outperforms Diffcast on this metric. This suggests that while the absolute error may be high, the model retains some ability to capture the underlying spatial-temporal structure of high-intensity rainfall events better than Diffcast, though still behind NowcastingGPT.

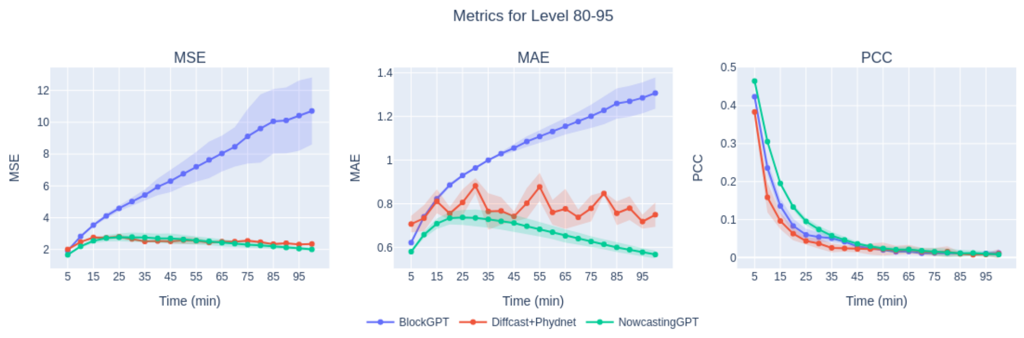


Figure 5.16: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing high value events, but not extremes. Solid line and shading indicate average and standard deviation across seeds.

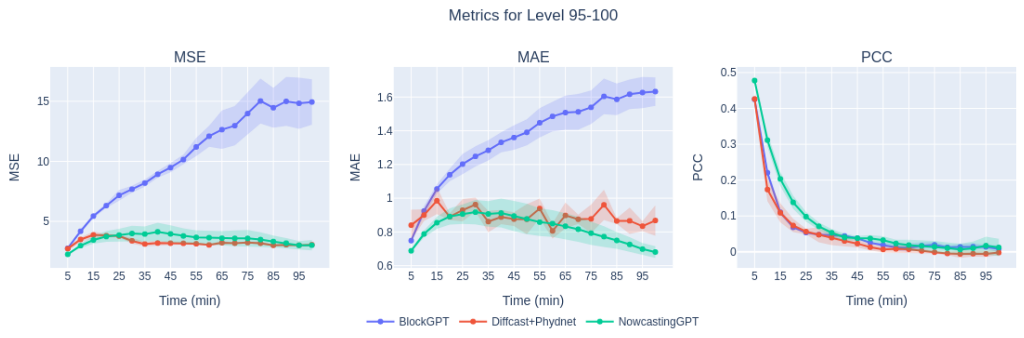


Figure 5.17: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing extreme events. Solid line and shading indicate average and standard deviation across seeds.

5.2.2 Results on the SEVIR dataset

Overall Results

This section presents overall results on the SEVIR dataset for the 5-minute resolution prediction task. On the continuous metrics, visualised in Figure 5.18, Diffcast, on average, consistently outperforms our model. However, we consistently outperform NowcastingGPT. In terms of MSE, Diffcast has a 20% advantage in the first time step. This gap, however, falls sharply and decreases to 0. We have a clear advantage over NowcastingGPT of about 20%. Despite the standard deviation of Diffcast across its seeds being high, it still outperforms BlockGPT.

Results on MAE are slightly in favour of BlockGPT. We outperform Diffcast by a margin of about 10%, which progressively falls to 0 over the last time steps.

Over NowcastingGPT, we start with a performance gap of 10%, which tapers to 0. Taking into account the standard deviation of the models, these conclusions are statistically significant. The standard deviation of Diffcast is in fact, higher across its seeds, and we can say that BlockGPT outperforms Diffcast, although with a slight margin

On PCC, Diffcast once again has a slight performance advantage over BlockGPT, with a maximum difference of about 6%. BlockGPT again outperforms NowcastingGPT, with a performance gap that increases to about 4% over time. Once again, although the standard deviation of seeds in Diffcast is higher, it consistently outperforms BlockGPT.

In summary, across all continuous metrics, Diffcast and BlockGPT have very comparable performance, although Diffcast has the edge on MSE and PCC. These results support our hypothesis that diffusion models are more effective at capturing fine details, an aspect that is more critical in the 5-minute resolution task than in the 30-minute one. Nevertheless, our frame-level autoregressive model, BlockGPT, outperforms the token-level autoregressive approach of NowcastingGPT on MSE, MAE and PCC. This indicates that the stronger inductive bias of modelling entire frames as atomic units in BlockGPT is more beneficial than the token-by-token nature of NowcastingGPT.

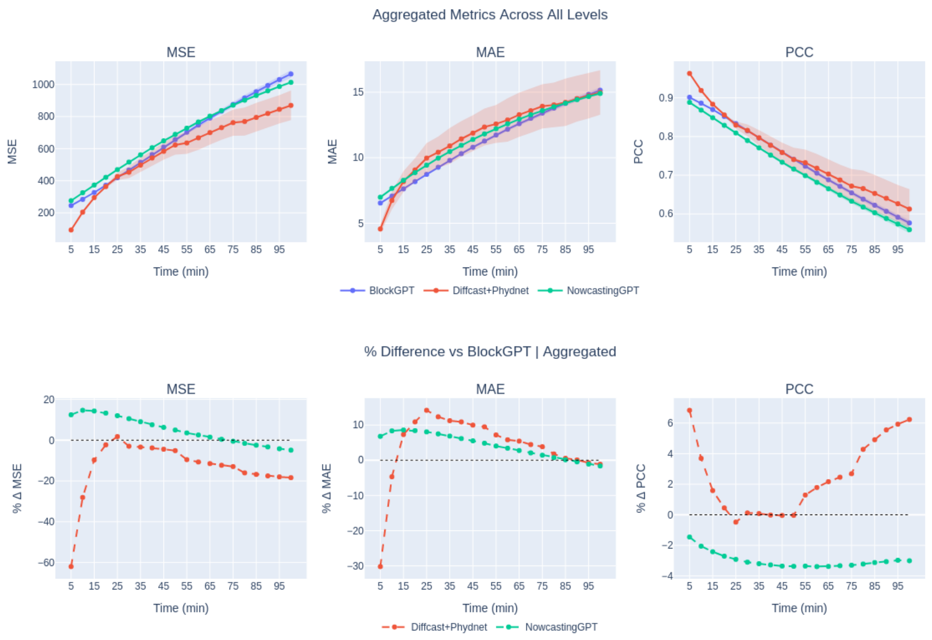


Figure 5.18: Continuous Metrics (MSE, MAE and PCC) on the SEVIR dataset (task 2, predict 100 minutes given 25 minutes of context). Solid line and shading indicate average and standard deviation across seeds. Top: Absolute values of the metrics of each model as a function of time. Bottom: metric values relative to BlockGPT, in percentage deviation.

Results on CSI show that BlockGPT performs well in the lower thresholds, but is slightly lacking compared to Diffcast and NowcastingGPT on the higher thresholds. Compared to the benchmarks, on the lower thresholds, BlockGPT, at its highest, has a performance advantage of about 20%. However, on the higher thresholds, compared to Diffcast, BlokcGPT starts off much poorer at the initial time steps, but this performance gap reduces to 0 towards the end. Compared to NowcastingGPT, however, it starts off a few per cent better, but NowcastingGPT outperforms BlockGPT towards the last few time steps.

Similar trends are observed for FAR as well. Like other experiments, a common pattern, especially at higher thresholds, is that the performance differences among our models diminish after the first few time steps. On the three higher thresholds, the FAR values for all models converge to nearly 1, indicating that the models struggle to accurately predict extreme pixels and tend to overestimate their values in later time steps. This points to an accumulation of error over time, which particularly affects the prediction of extreme precipitation values.

Within these narrow margins, across all thresholds except two, Diffcast consistently outperforms our model—but only during the initial time steps. In the lowest threshold, Diffcast has a 20% advantage consistently across all time steps. On the second and third threshold, BlockGPT is actually better, with a performance gap of about 10 to 20%. However, on the last three thresholds, except after the initial time steps, the models perform very similarly. BlockGPT outperforms NowcastingGPT on FAR, with similar trends, by 20% on average on the three lower thresholds, but with much lower margins in the higher thresholds.

Thus, across all metrics, DiffCast consistently demonstrates the best overall performance, particularly in the early time steps and at higher precipitation thresholds. BlockGPT, while slightly behind DiffCast in absolute terms, shows competitive results, especially on structure-aware metrics like PCC and FAR, and consistently outperforms NowcastingGPT across the board. Although we expect Diffcast to be better on the 5-minute task, it is surprising that BlockGPT achieves worse CSI compared to NowcastingGPT, on the last three thresholds, but this is true only on later time-steps.

These results confirm that DiffCast's diffusion-based architecture excels at capturing fine details and early lead-time accuracy, particularly in high-resolution settings. However, BlockGPT's frame-level autoregressive structure offers a strong inductive bias, leading to improved spatial consistency and a reliable detection of high-impact events. Its superior performance over NowcastingGPT across all metrics except CSI on the last three thresholds underscores the benefit of treating entire frames as atomic units, reducing error accumulation and enhancing temporal coherence. While diffusion models currently lead in precision, BlockGPT presents a strong alternative that balances performance and speed.

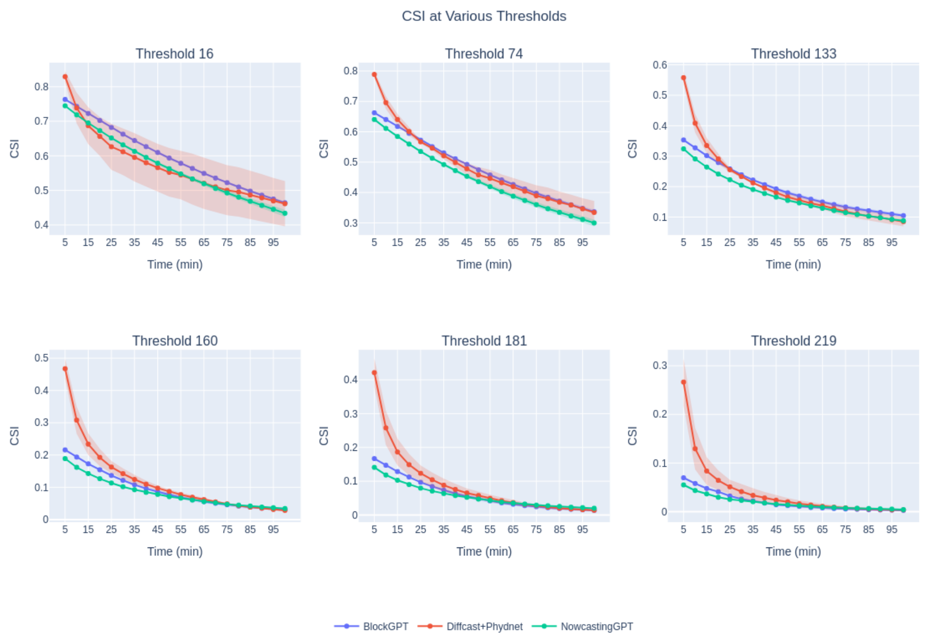


Figure 5.19: CSI values evaluated at different thresholds on the SEVIR dataset (task 2: predicting 100 minutes, given 25 minutes of context). Solid line and shading indicate average and standard deviation across seeds.

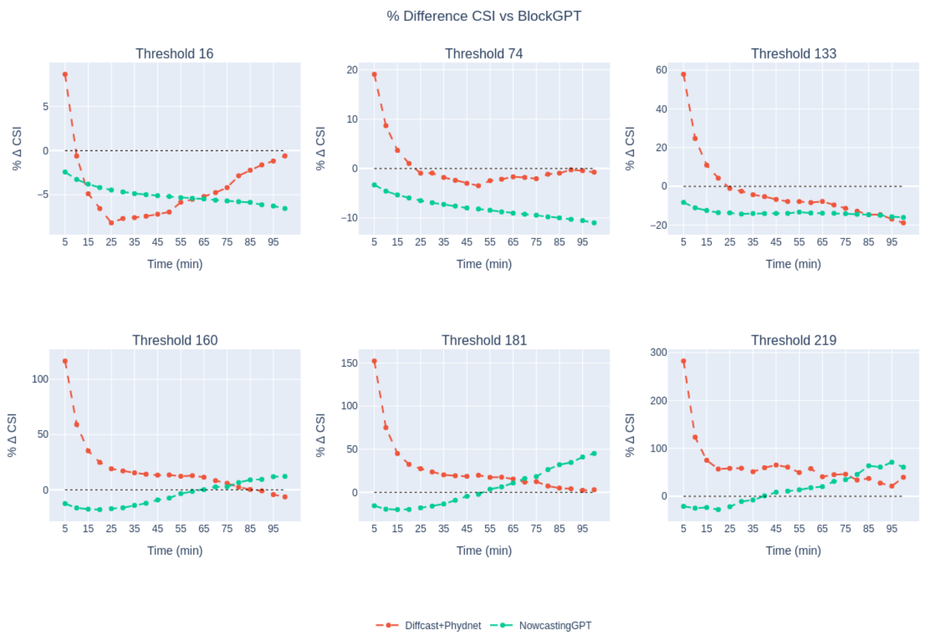


Figure 5.20: Relative difference between CSI values with respect to BlockGPT.

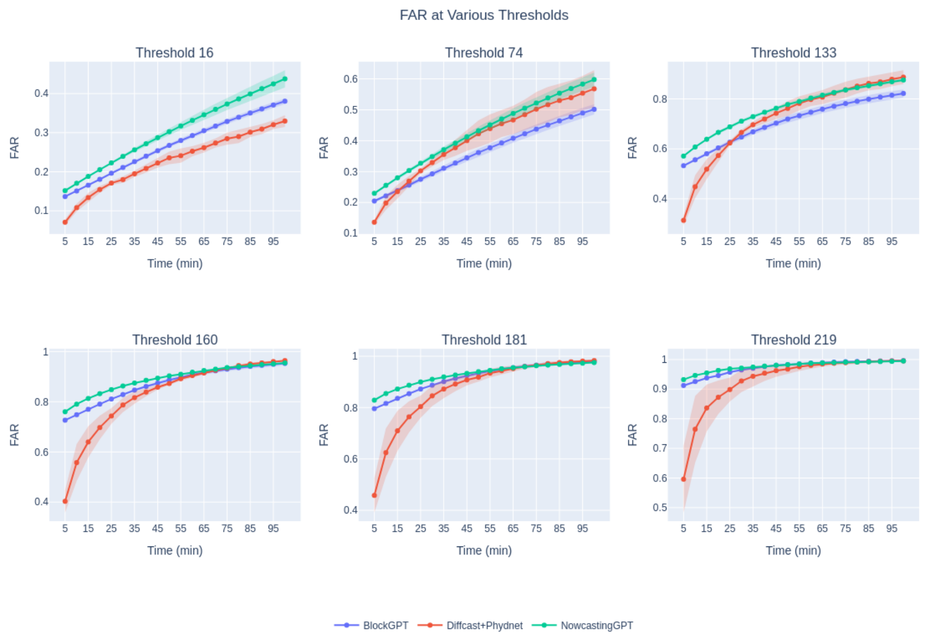


Figure 5.21: FAR values evaluated at different thresholds on the SEVIR dataset (task 2: predicting 100 minutes, given 25 minutes of context). Solid line and shading indicate average and standard deviation across seeds.

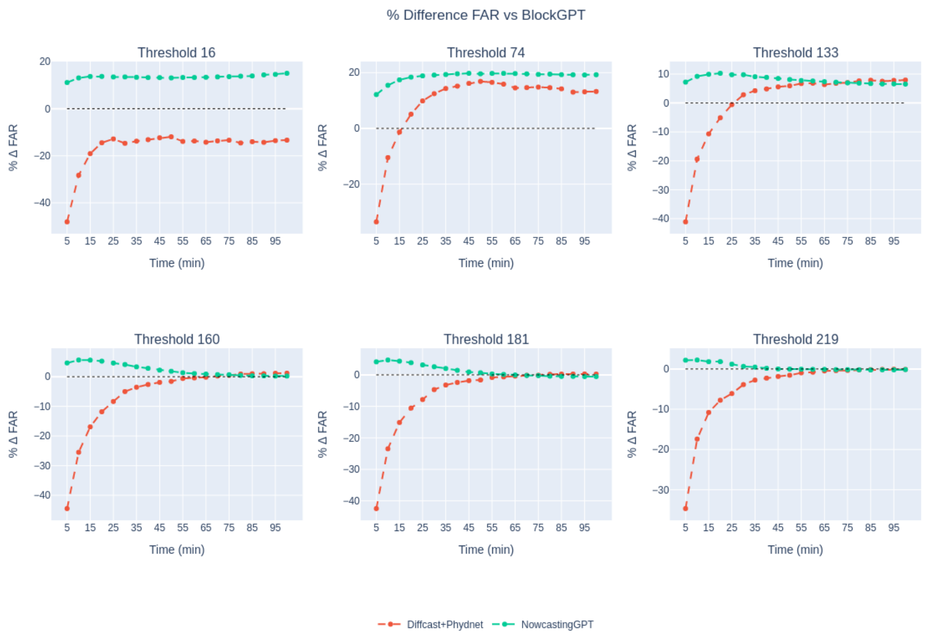


Figure 5.22: Relative difference between FAR values with respect to BlockGPT

Performance on Higher Levels

Results on the higher levels in the SEVIR dataset, shown in Figures 5.23 and 5.24, reveal that for each block, results are once again very similar to the global, overall metrics. BlockGPT consistently outperforms NowcastingGPT on MAE, MSE and PCC. However, compared to Diffcast, it has poorer MSE and PCC. On MAE, BlockGPT has slightly better values. To reiterate, better MAE values than MSE indicate the presence of outliers- a common theme that has been observed with BlockGPT. This carries over to the extreme events as well.

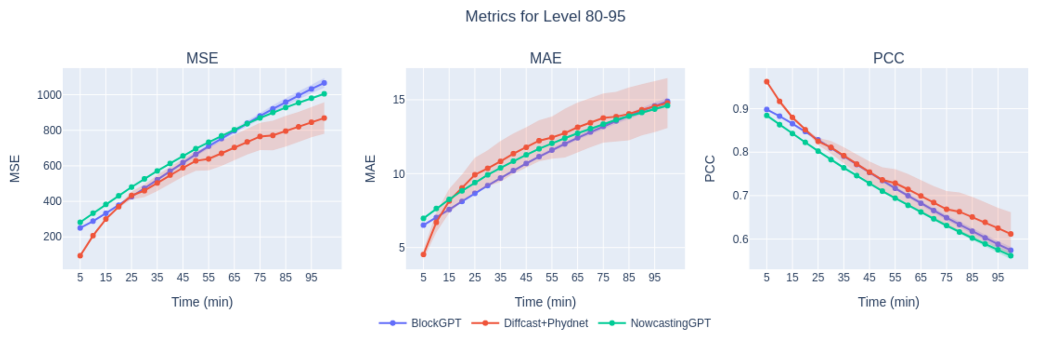


Figure 5.23: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing high value events, but not extremes. Solid line and shading indicate average and standard deviation across seeds.

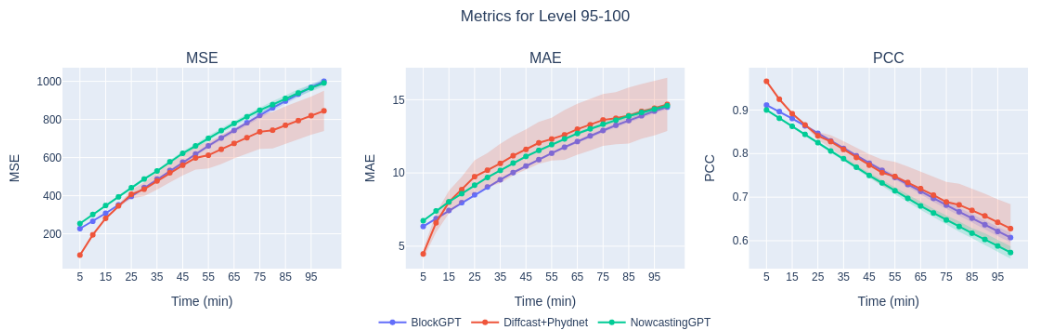


Figure 5.24: Continuous Metrics exclusively evaluated in the block 80-95 percentile of events, representing extreme events. Solid line and shading indicate average and standard deviation across seeds.

5.3 Extended Analysis on the KNMI Dataset

In this thesis, we place particular emphasis on the Dutch KNMI dataset and conduct a series of extended evaluations focused on it. These include catchment-based and seasonal analyses. As introduced in Section 3, catchment zones in the Netherlands are critically important for accurate weather prediction, as they are more susceptible to flooding and its associated risks. The first part of this section presents model evaluations specific to these catchment zones. In the latter part, we assess model performance across different seasons to examine how varying weather complexities affect predictive accuracy.

5.3.1 Catchment Analysis

To assess the performance of our models in regions of hydrological significance, we perform targeted evaluations over catchment zones in the Netherlands, as described in Section 3. These zones, which are prone to localised flooding, are of critical importance in practical nowcasting applications. Figure 4.1 illustrates the layout of these catchment areas. Since our models and the baseline benchmarks are not designed to accept cropped spatial inputs, we evaluate catchment performance by running the models on the full spatiotemporal event and subsequently extracting the relevant subregions from both the ground truth and predicted outputs for metric computation.

For the extracted catchment regions, we compute Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves. These curves provide complementary views of model performance under varying threshold criteria:

- The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) across different classification thresholds. Its Area Under the Curve (AUC) reflects the probability that the model ranks a randomly chosen positive event higher than a randomly chosen negative one. A higher ROC-AUC indicates better overall separability between rain and no-rain regions.
- The PR curve plots Precision (positive predictive value) against Recall (sensitivity) and is particularly useful when the data is imbalanced, as is often the case in precipitation prediction, where heavy rainfall is rare. The PR-AUC captures the trade-off between correctly identifying extreme rainfall and minimising false alarms.

We compute these metrics for multiple rainfall intensity thresholds (1mm, 2mm, and 8mm), in line with our previous categorical evaluations, to facilitate comparison across model types and rainfall levels. Furthermore, because model accuracy is known to deteriorate over time due to error accumulation in autoregressive generation, we compute time-resolved AUC values—that is, AUC-ROC and AUC-PR at each forecast timestep.

Figure 5.25 summarises the temporal evolution of these AUC values. Several observations emerge:

- BlockGPT consistently achieves higher AUC-ROC values than both Diffcast and NowcastingGPT, across all thresholds. This indicates robust discriminative power in separating rainfall from non-rainfall regions, regardless of rainfall intensity.
- For AUC-PR, BlockGPT also performs favorably across most thresholds. However, at the highest threshold (8mm), the performance gap narrows. Notably, NowcastingGPT occasionally outperforms BlockGPT at certain time steps. Results on both ROC and PR curves are statistically significant, taking into account the standard deviation of models across seeds.
- Across all models, we observe a clear degradation in AUC values over time, especially at the later time steps. This aligns with prior observations from the CSI and FAR metrics (Figures 5.2 and 5.3), which showed declining critical success index and rising false alarm rates over longer forecast horizons.

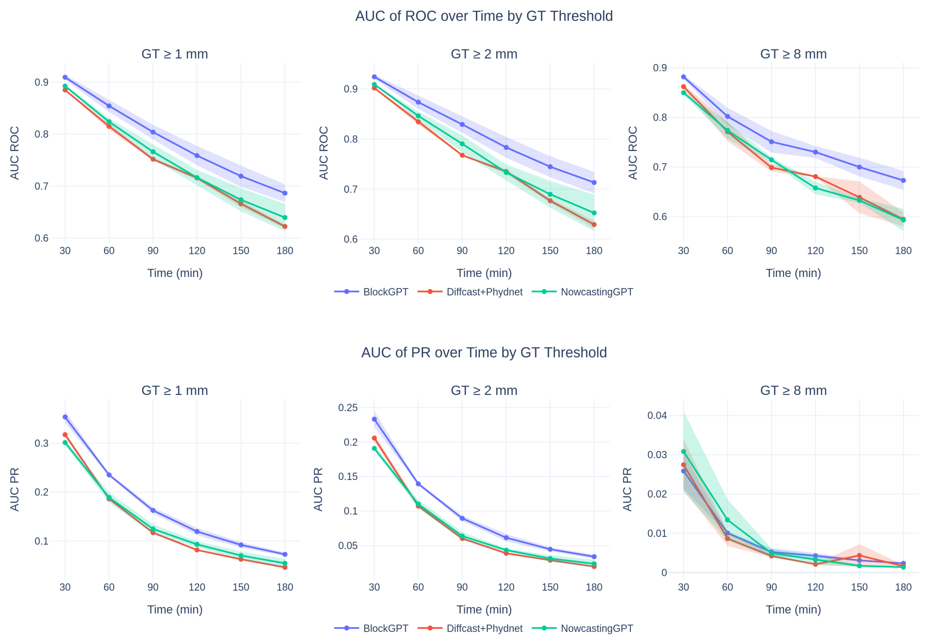


Figure 5.25: Area Under the Curve of ROC and PR over time, aggregated across all the catchment regions in the Netherlands. The models are evaluated at different thresholds to analyse performance at different rainfall levels. Solid lines indicate results averaged across seeds, and shaded parts indicate standard deviation.

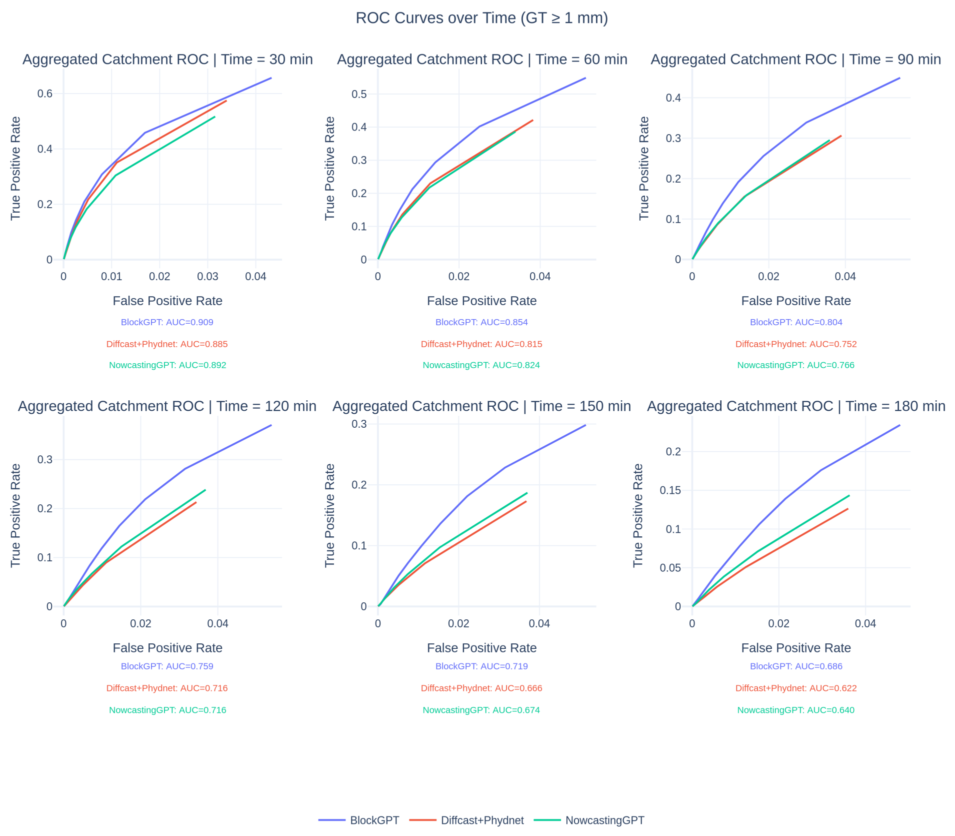
This temporal decline reflects two key phenomena: first, increased difficulty in predicting high-precipitation extremes as thresholds become stricter, owing to their inherently low frequency and higher spatial variability. Second, Autoregressive error accumulation, a known limitation in sequence modelling, where slight inaccuracies in early frames compound and propagate through time. These results underscore both the strengths and limitations of the evaluated models. While BlockGPT demonstrates superior overall performance, especially in moderate rainfall scenarios, all models struggle as they forecast further into the future and when tasked with detecting extremes.

Figures 5.26a and 5.26b show the ROC and PR curves at the 1mm threshold, respectively. BlockGPT starts off with an AUC of ROC of 0.909, which degrades to 0.686 at the end of 3 hours. Diffcast and NowcastingGPT start off with 0.885 and 0.892, respectively, and trail off to 0.622 and 0.640, respectively. The PR curves start off with considerably lower AUC, as indicated in 5.25. BlockGPT starts with a value of 0.354 and falls to a very low value of 0.073 after three hours. Diffcast and NowcastingGPT start with 0.317 and 0.301, respectively. Both fall to 0.046 and 0.055 at the end of three hours. Although BlockGPT considerably outperforms both NowcastingGPT and Diffcast, the PR curves of all models indicate that their skill diminishes rapidly as time progresses, indicating the difficulty of the task. This is expected since we have already seen that the CSI and FAR of models fall to near 0 and 1 in previous analyses.

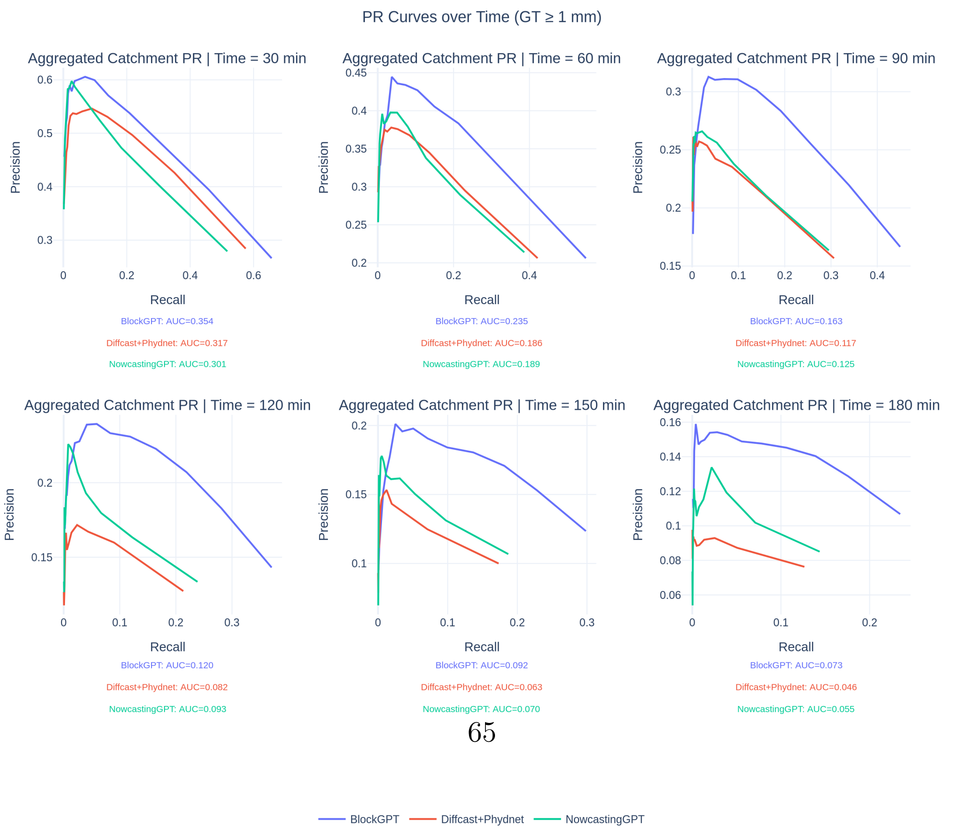
As expected, the narrative shows decreasing skill at larger thresholds. As seen in Figure 5.27a, at the 2mm threshold, BlockGPT starts with an AUC of ROC of 0.924 and falls to an AUC of 0.713. Diffcast and NowcastingGPT start with 0.902 and 0.909, respectively. These values fall to 0.629 and 0.653. On the PR curves, shown in Figure 5.27b BlockGPT starts with an AUC of 0.233, which falls to 0.034 at the 3-hour mark. NowcastingGPT and Diffcast start with 0.206 and 0.191, which fall to 0.020 and 0.024, respectively. Once again, BlockGPT outperforms the benchmarks, but the values in general are very low.

On the 8mm threshold, shown in Figures 5.28a and 5.28b, BlockGPT starts with an AUC of ROC of 0.882, which falls to 0.673. Diffcast and NowcastingGPT start with 0.862 and 0.850. These values fall to 0.595 and 0.593, respectively. On the PR curves, the values of all models are the lowest at the 8mm threshold. BlockGPT starts with 0.026 and ends with 0.002 at the 3-hour mark. Diffcast and NowcastingGPT start with 0.027 and 0.031, respectively. At the 3-hour mark, both have a PR AUC of 0.002 and 0.001.

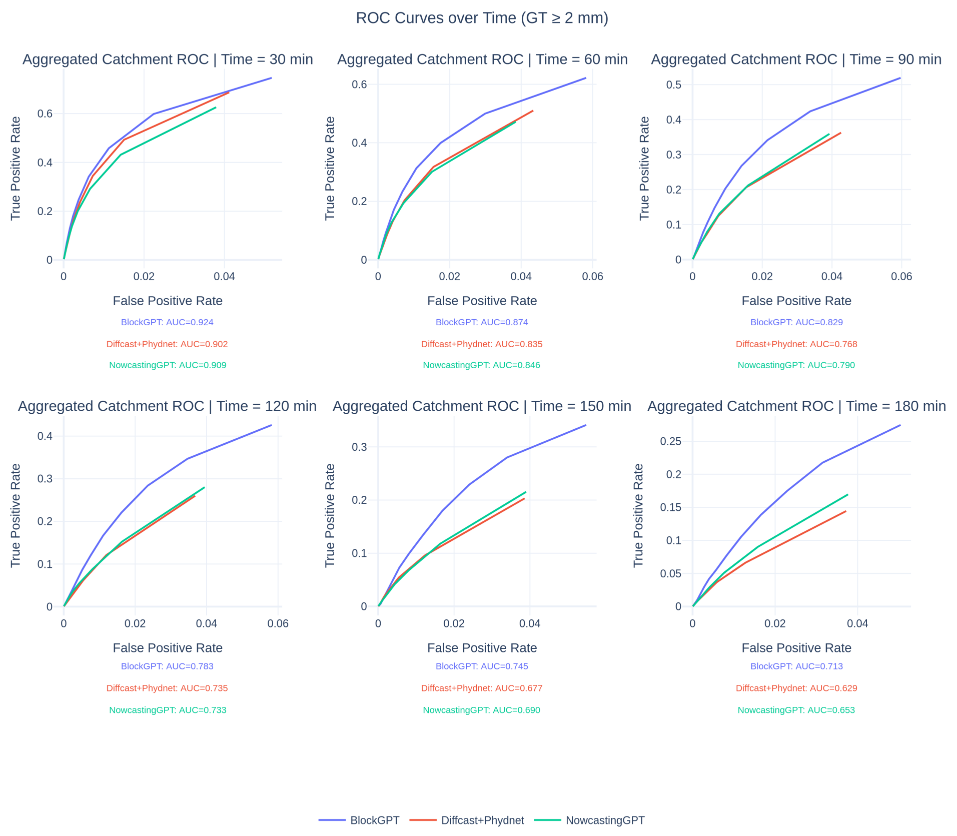
To summarise, BlockGPT consistently outperforms NowcastingGPT and Diffcast across all thresholds on both ROC and PR AUC metrics. However, performance degrades significantly over time, especially in PR AUC, reflecting the increasing difficulty of detecting rare rainfall events at longer lead times and higher thresholds. Despite this, BlockGPT maintains a relative advantage over time. The higher degradation in the Precision-Recall (PR) AUC than in the ROC AUC is expected in highly imbalanced settings (the dataset becomes more imbalanced as the threshold increases), where positive events become increasingly rare. ROC AUC, being relatively insensitive to class imbalance, degrades gradually. In contrast, PR AUC is highly sensitive to both false positives and missed detections, leading to a sharper collapse in performance. The rapid drop in PR AUC across all models highlights the growing difficulty in confidently identifying true rainfall exceedance events.



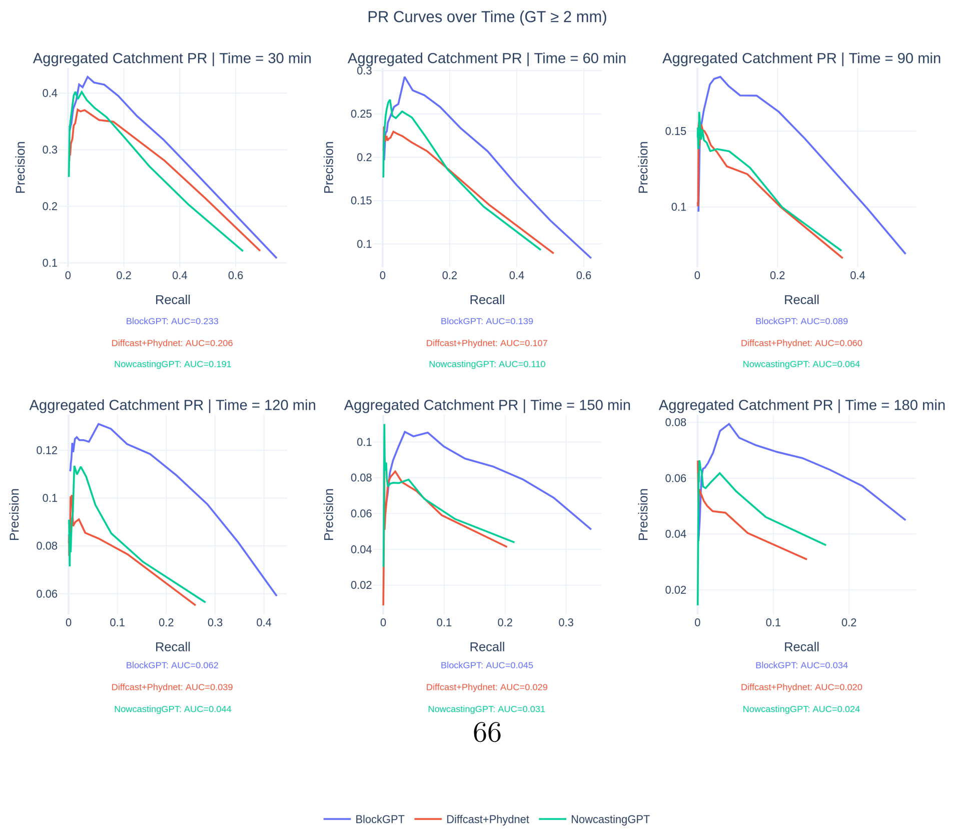
(a) The ROC curves aggregated over the catchment regions, with ground truth thresholded at 1mm of rainfall.



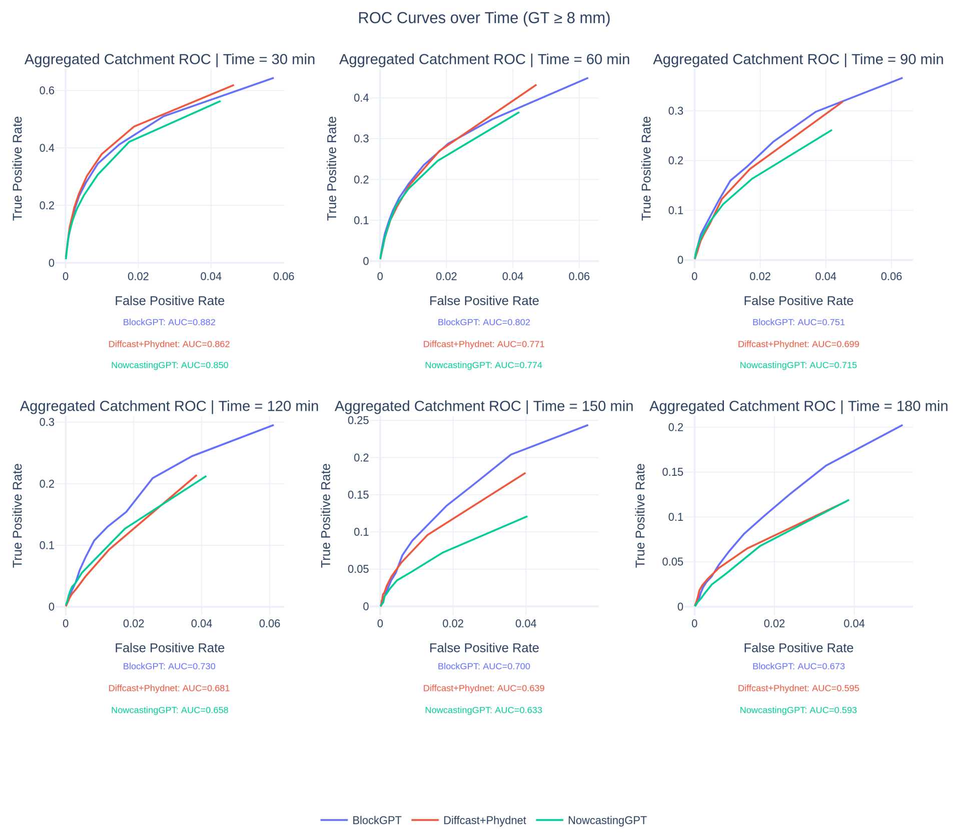
(b) The PR curves aggregated over the catchment regions, with ground truth thresholded at 1mm of rainfall.



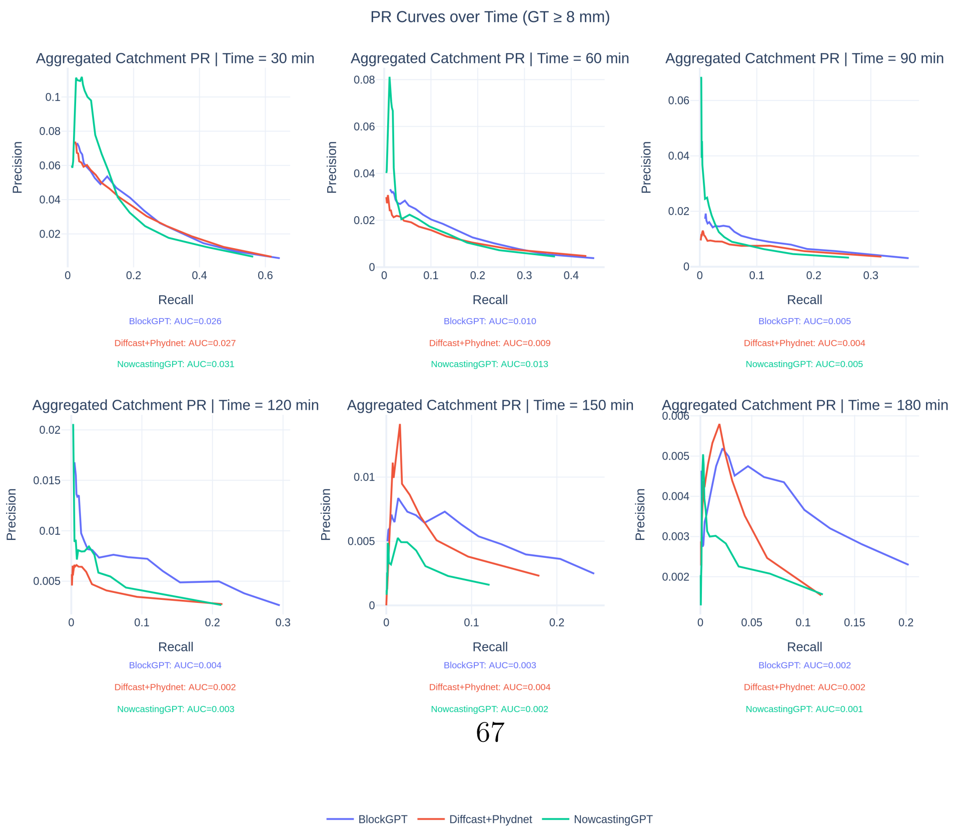
(a) The ROC curves aggregated over the catchment regions, with ground truth thresholded at 2mm of rainfall.



(b) The PR curves aggregated over the catchment regions, with ground truth thresholded at 2mm of rainfall.



(a) The ROC curves aggregated over the catchment regions, with ground truth thresholded at 8mm of rainfall.



(b) The PR curves aggregated over the catchment regions, with ground truth thresholded at 8mm of rainfall.

5.3.2 Seasonal Analysis

In this section, we analyse the performance of our model across different seasons of the year. It is important to note that seasonal information is **not** provided to the model as an explicit input feature. During training, events are sampled in randomly shuffled batches, and the model processes each event independently without knowledge of its occurrence time. Consequently, this seasonal analysis serves as an exploratory investigation to determine whether any consistent performance patterns emerge across seasons, despite the model's lack of temporal awareness.

To conduct this analysis, we curate a separate test set composed of events exclusively from each season: spring (March to May), summer (June to August), fall (September to October), and winter (November to February).

According to the continuous metrics reported in Figure 5.29, BlockGPT achieves its best performance in winter, and its worst in summer, with spring and fall falling in between. This trend may be explained by the inherent variability in weather patterns: summer months typically exhibit higher atmospheric energy and convective activity, leading to more chaotic and difficult-to-predict events. In contrast, winter weather patterns are often more stable and structured, potentially making them easier for the model to extrapolate.

Interestingly, the pattern reverses when considering categorical metrics, as shown in Figure 5.30. These metrics assess whether predictions and ground truth exceed fixed thresholds on a pixel-wise basis. Across nearly all thresholds in the Critical Success Index (CSI), BlockGPT performs best in summer and worst in winter. A similar trend is observed in False Alarm Rate (FAR) metrics, with the model achieving its lowest FAR in summer and the highest in winter. These findings are consistent across multiple random seeds, indicating statistical significance.

This difference between continuous and categorical performance is intriguing. On one hand, the model struggles with precise value prediction in summer (as seen in continuous metrics like MSE, MAE, and PCC), likely due to the more chaotic nature of precipitation. On the other hand, it is more successful at crossing the correct threshold boundaries, which is rewarded in categorical metrics. This suggests that while the model may not precisely estimate intensities during turbulent summer events, it still detects the presence of precipitation effectively.

Given that the model has no concept of seasonality, such patterns are not expected a priori. Yet the emergence of clear, and opposite, seasonal trends in continuous and categorical metrics highlights a fascinating interaction between event characteristics and model behaviour. However, it must be conceded that to understand better why these patterns emerge, it is important to study how the data curation affects the distribution of events in different seasons. Recall that we accumulate events over the entire duration of the dataset (2008-2018), form 4-hour events, and then proceed to truncate the bottom 50th percentile since they contain little to no rainfall. It is indeed possible that this truncation affects the true seasonal distribution. For instance, if there are relatively lower intensity rainfall events occurring in a certain season, they could be truncated, causing poor generalisation when evaluating exclusively in that period.

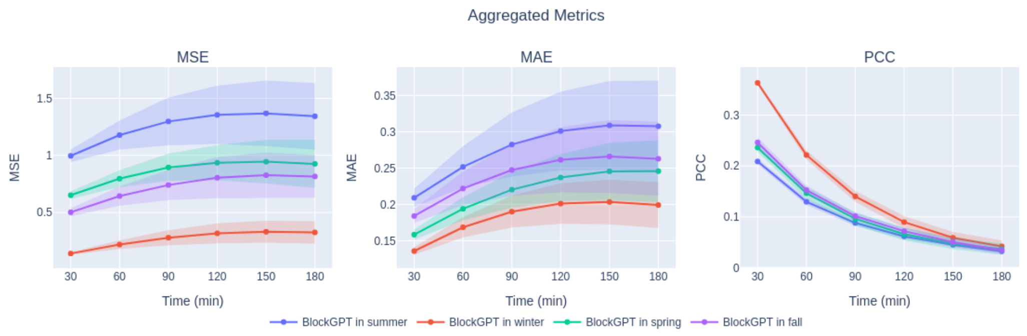


Figure 5.29: Continuous metrics of BlockGPT on the KNMI dataset, evaluated in different seasons (task 1, predict 3 hours, with 1 hour of context). Each line represents the performance of our model within a particular season. The solid line represents results averaged across seeds, and the shaded zone represents standard deviation across seeds.

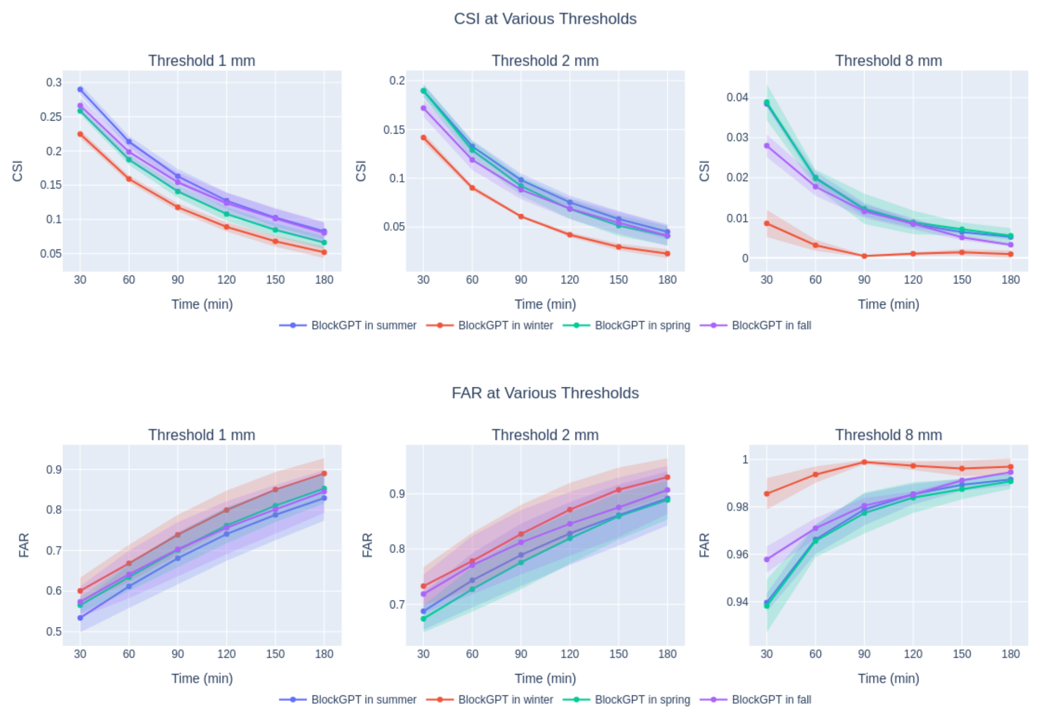


Figure 5.30: CSI and FAR of BlockGPT on the KNMI dataset, evaluated in different seasons (task 1, predict 3 hours with 1 hour of context). Each line represents the performance of our model within a particular season. The solid line represents results averaged across seeds, and the shaded zone represents standard deviation across seeds.

5.4 On the Superior Inductive Bias of Frame-by-Frame Generation

In this section, we investigate in greater detail the inductive bias introduced by generating predictions one frame at a time. As outlined in our methodology, BlockGPT is built on the idea of generating blocks of tokens, where each block corresponds to a full frame. Within each block, tokens can attend to one another bidirectionally, effectively allowing the model to treat an entire frame as a cohesive unit. This design encourages the model to learn spatial dependencies more effectively. More generally, however, the concept of a "block" can be adapted-if we set the block size to one token, we recover a token-by-token autoregressive model. If we define a block as a row, we get a row-by-row generation scheme. Inspired by experimental design choices in [26], we systematically vary the block size to evaluate three variants: token-by-token, row-by-row, and frame-by-frame generation. Our goal is to determine whether the frame-wise approach indeed offers the most effective inductive bias for video prediction in this domain.

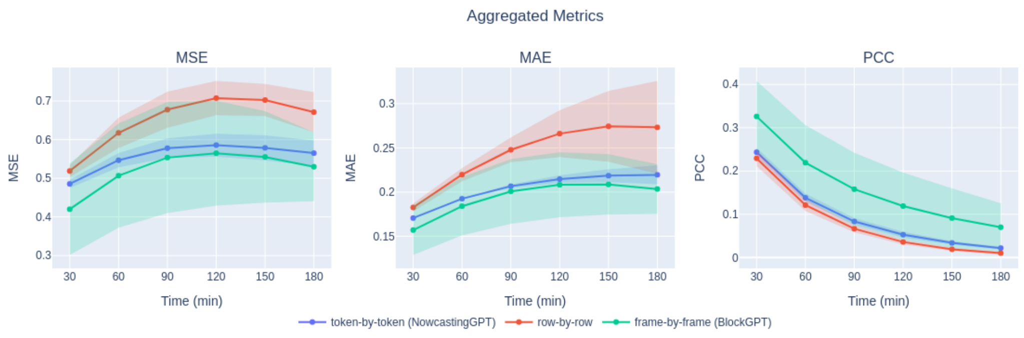


Figure 5.31: Continuous Metrics of the frame-by-frame, row-by-row and token-by-token methods on the KNMI dataset (task 1, predict 3 hours given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds

The results of this experiment on continuous metrics are visualized in Figure 5.31. On average, the frame-by-frame method (BlockGPT) clearly outperforms the other two, followed by the token-by-token and row-by-row methods, which exhibit similar but substantially lower performance. This pattern holds for the categorical metrics as well. As seen in Figure 5.32 (top), BlockGPT achieves the highest CSI scores across the board, while the token-by-token and row-by-row variants lag significantly behind and are nearly indistinguishable from each other. The same holds for the FAR scores in Figure 5.32 (bottom), where BlockGPT demonstrates notably better behavior.

This clear separation in performance indicates that the superior results of BlockGPT are not merely due to having fewer autoregressive rollout steps. If that were the dominant factor, we would expect a monotonic relationship where row-by-row (with fewer steps than token-by-token) performs better. However, the fact that row-by-row and token-by-token perform similarly suggests that the inductive bias introduced by the frame-level generation. treating an entire frame as a coherent unit and allowing bidirectional attention within it, is the critical advantage.

This aligns with the nature of the VQGAN used for tokenization. Since each token already encodes global spatial context due to the convolutional receptive field, the tokens within a frame are highly interdependent. Thus, allowing these tokens to communicate freely within a frame is crucial. By contrast, row-by-row generation arbitrarily enforces a sequential structure that does not match the underlying spatial dependencies in the data. The performance gap supports the conclusion that BlockGPT's frame-wise design is not just computationally efficient, but also a significantly better inductive bias for spatiotemporal video prediction.

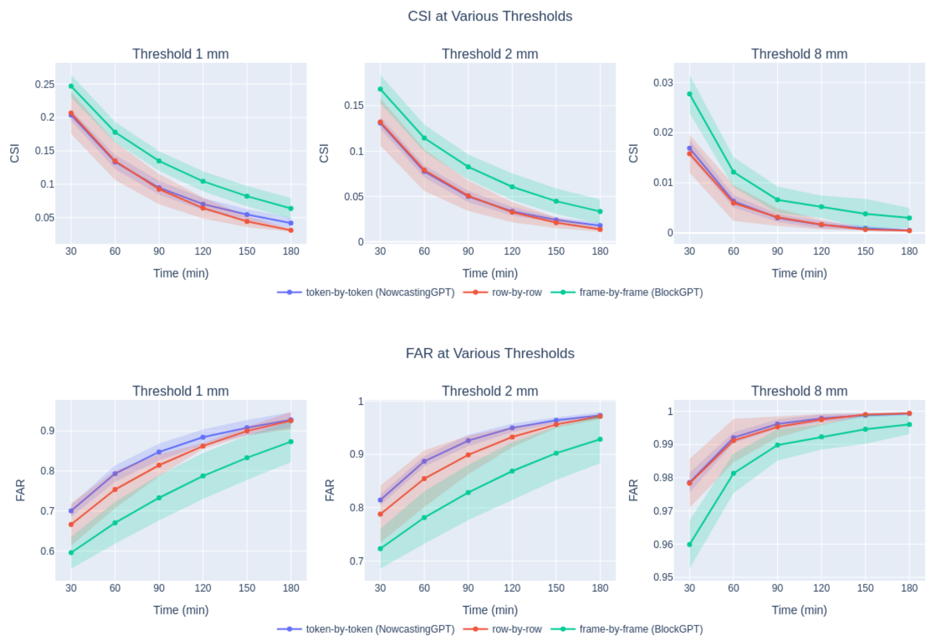


Figure 5.32: Categorical Metrics of the frame-by-frame, row-by-row and token-by-token methods on the KNMI dataset (task 1, predict 3 hours given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds.

6 Conclusion

This thesis introduced BlockGPT, a frame-level autoregressive model for precipitation nowcasting, and evaluated its performance across multiple datasets and forecasting tasks. Through an extensive set of experiments on the KNMI and SEVIR datasets, we benchmarked BlockGPT against leading generative models, including NowcastingGPT and DiffCast, and drew insights into its behaviour across both global and extreme weather conditions.

A summary of the results allows us to draw some overarching conclusions. On categorical metrics, BlockGPT outperforms NowcastingGPT and DiffCast on both the 5-minute and 30-minute tasks on the KNMI dataset. On the SEVIR dataset, we outperform both benchmarks on the 30-minute task. However, on the 5-minute task, we only outperform NowcastingGPT and narrowly lag behind DiffCast.

On continuous metrics, BlockGPT on average outperforms NowcastingGPT and DiffCast on the KNMI 30-minute task, although the results are not statistically significant for MSE and MAE. On the 5-minute task, BlockGPT has the highest MSE and MAE compared to both benchmarks, though it achieves better PCC than DiffCast. On the SEVIR 5-minute task, BlockGPT performs better than NowcastingGPT on all continuous metrics (MSE, MAE, and PCC), but only surpasses DiffCast in terms of MAE. On the 30-minute task, BlockGPT once again records the highest MSE and MAE but achieves a higher PCC than NowcastingGPT.

Thus, we can fairly summarise that, especially in terms of categorical metrics, BlockGPT consistently outperforms NowcastingGPT. In contrast, its performance on continuous metrics is less consistent, with improvements mainly observed on the KNMI 30-minute task and the SEVIR 5-minute task. One of BlockGPT's main limitations is its tendency to exhibit increasing MSE and MAE.

To analyse BlockGPT's performance relative to DiffCast, it is useful to look deeper into the temporal resolution of the tasks and the datasets themselves. On KNMI, BlockGPT outperforms DiffCast on all metrics for the 30-minute task. However, for the 5-minute task, it only surpasses DiffCast on categorical metrics and PCC. On SEVIR, BlockGPT outperforms DiffCast on categorical metrics for the 30-minute task but lags behind on all metrics for the 5-minute task. This reveals a clear pattern: BlockGPT performs comparatively worse on the 5-minute task than on the 30-minute task, particularly when compared to DiffCast.

Another pattern also emerges: BlockGPT's advantage reduces on SEVIR compared to KNMI, even at the same resolution. A closer look at the datasets reveals that frame-to-frame differences are typically larger in KNMI than in SEVIR, as shown in Figure 6.1. This suggests that the inherent dynamics in KNMI are more complex and harder to predict. Suppose we decompose the task of precipitation forecasting into two components: (1) predicting the general shape of the weather system and (2) refining these predictions to enhance granularity. When DiffCast was introduced, its main innovation was its ability to produce detailed, fine-grained predictions. The authors of [12] argue that transformer-based models suffer from blurring and propose that a diffusion-based residual predictor enables sharper outputs. This is also evident in the SEVIR visualisations in the Appendix.

BlockGPT, on the other hand, appears more effective at predicting the overall structure or shape of the precipitation field. On datasets like KNMI, or at coarser temporal resolutions, this quality is more valuable, since capturing the general dynamics becomes the primary challenge. This explains the consistent pattern we observe: BlockGPT performs better on KNMI than SEVIR, and better on the 30-minute task than the 5-minute task. We conclude that although BlockGPT is more prone to blurring, it is better suited for long-range or coarse-resolution forecasting, where its ability to model broad patterns gives it an advantage.

Beyond benchmarking, we conducted additional evaluations on the KNMI dataset. In the catchment-based evaluation, BlockGPT consistently outperforms both NowcastingGPT and DiffCast on categorical metrics such as AUC-ROC and AUC-PR, across all rainfall thresholds. Despite not being explicitly trained for spatial subregions, BlockGPT generalises well to hydrologically critical zones, accurately distinguishing high- and low-precipitation events. This performance, especially at higher thresholds, highlights the model's robustness in detecting extreme rainfall within localised regions, an essential capability for flood-prone areas.

Finally, we compare inference times across models. As shown in Table 6.1, BlockGPT is significantly faster than both benchmarks. On the 30-minute task, it is 27× faster than NowcastingGPT and 31× faster than DiffCast. On the 5-minute task, it is 31× faster than NowcastingGPT and 10× faster than DiffCast. These results indicate that frame-level autoregression not only improves performance but also greatly enhances computational efficiency. Compared to token-level prediction, BlockGPT provides superior speed and accuracy, consistently outperforming NowcastingGPT. Against DiffCast, it performs more reliably in coarse temporal resolutions, offering better accuracy farther into the future using the same number of frames.

To close our discussion on model performance, we must acknowledge an important limitation: although BlockGPT demonstrates competitive results relative to our benchmarks, the overall performance of all models remains modest—particularly on

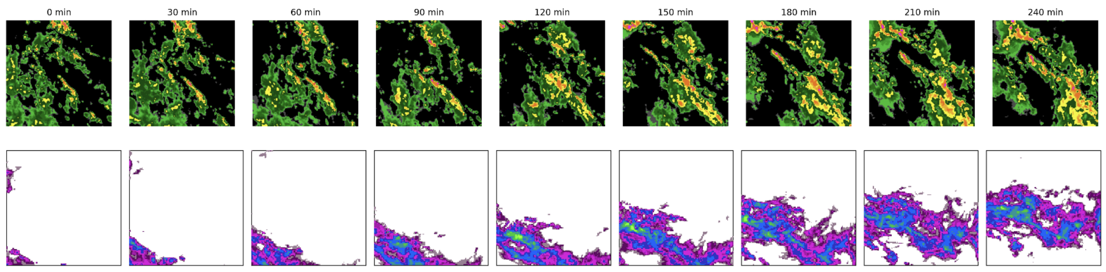


Figure 6.1: A comparison of the SEVIR (top) and KNMI (bottom) dataset at the same temporal resolution. The frames reveal an interesting property- in the SEVIR dataset, the weather pattern remains relatively at the same position, with fewer changes between the frames. However, the change is more prominent on the KNMI dataset. This suggests that the inherent dynamics in the KNMI dataset could be more difficult for models to learn, compared to SEVIR.

the KNMI dataset. This is evident from metrics such as the Critical Success Index (CSI) and False Alarm Rate (FAR), which degrade rapidly over time, with CSI approaching zero and FAR nearing one by the end of the 3-hour forecast horizon. Similarly, the Area Under the Precision-Recall Curve (AUC-PR) values remain low across all models, highlighting difficulty in maintaining high precision at longer lead times.

This raises questions about the practical utility of such models in real-world operational systems. One avenue we have not explored in this work is the use of post-processing to support actionable decisions based on model outputs. Our current evaluation uses pixel-wise categorical metrics, which may not align with the spatial scale at which decisions are typically made—especially in applications such as flood warning in catchment regions. Aggregating predictions over meaningful spatial units (e.g., catchments) and applying thresholds at this coarser level could improve decision relevance and mitigate the influence of local noise or outliers.

To conclude, while BlockGPT presents a promising modeling approach, bridging the gap between model output and operational utility will require further research into task-specific post-processing and spatially aggregated decision-making frameworks

Table 6.1: Inference time (in seconds) per batch for each model across two experiments.

Model	**Task 1 (generate 6 frames)**	**Task 2 (generate 20 frames)**
NowcastingGPT	7.09	47.37
DiffCast	8.17	15.51
BlockGPT	0.26	1.51

Future Work 7

In this thesis, we introduced BlockGPT, a core computer vision architecture to predict weather frames. In this chapter, ideas for future work will be pondered upon.

Throughout previous chapters, we have consistently highlighted the fact that Diffcast and BlockGPT have opposing strengths. We claim that the former is better at predicting fine detail, while the latter is better at predicting general weather shapes. Thus it could be promising to combine these models- to use our model as Diffcast's deterministic backbone. Since both models have different strengths, combining them could prove to be a significant improvement.

To investigate this further, we perform some initial experiments on task 1 on the SEVIR dataset to show the promising capacity of integrating our model with Diffcast. Specifically, BlockGPT generates a backbone prediction via full autoregressive rollout, and Diffcast is then used to refine its predictions. By analysing Figure 7.1, we can see that using Diffcast on top of BlockGPT clearly improves the fine detail in predictions, as well as curbs the effect of error accumulation during autoregressive rollout. These improvements are also observed in the metrics as well. In earlier experiments, we saw that BlockGPT underperformed on the continuous metrics on SEVIR. With Diffcast as an overhead, we achieve the best results across all models, as shown in Figure 7.2- we are thus significantly able to improve our continuous metrics, taking BlockGPT from having the worst MSE and MAE- to the best. On CSI, seen in Figure 7.3 BlockGPT had the best values across benchmarks. Using Diffcast improves it even further. However, on FAR (Figure 7.4, we have slightly poorer values than BlockGPT, but still consistently better than the benchmarks. Thus fusing BlockGPT with its ability to predict general shapes better- along with Diffcast, which can predict fine detail is a very promising direction to take in the future.

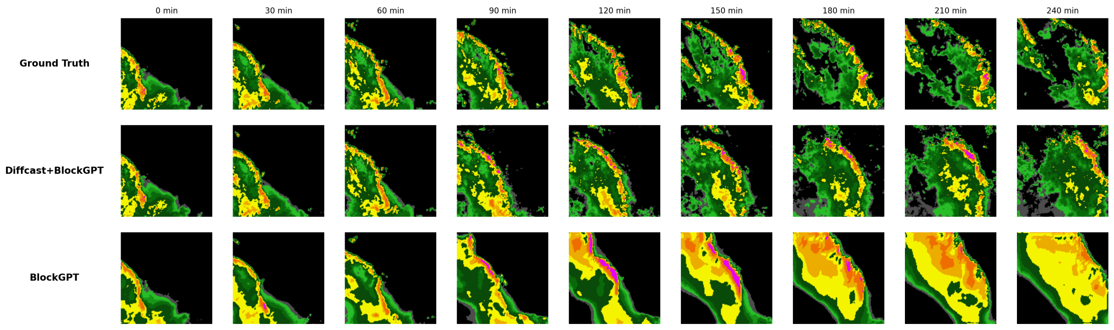


Figure 7.1: An example of a prediction of BlockGPT refined by Diffcast, compared to the backbone BlockGPT prediction. Clearly, Diffcast manages to greatly improve the fine-grained detail in our predictions, as well as moderately eliminating the errors accumulated during autoregressive rollout.

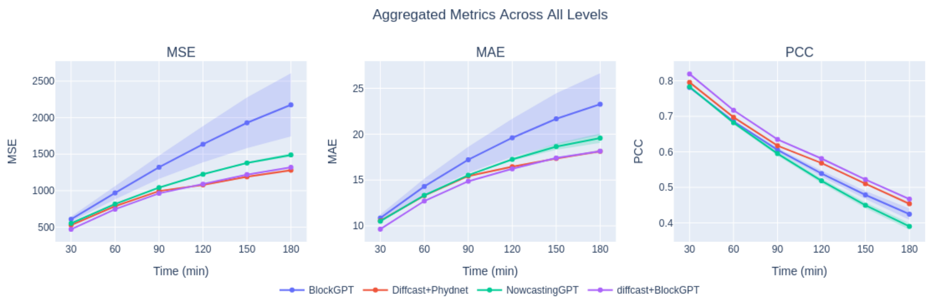


Figure 7.2: Continuous Metrics (MSE, MAE and PCC) on the SEVIR dataset (task 1, predict 3 hours given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds.

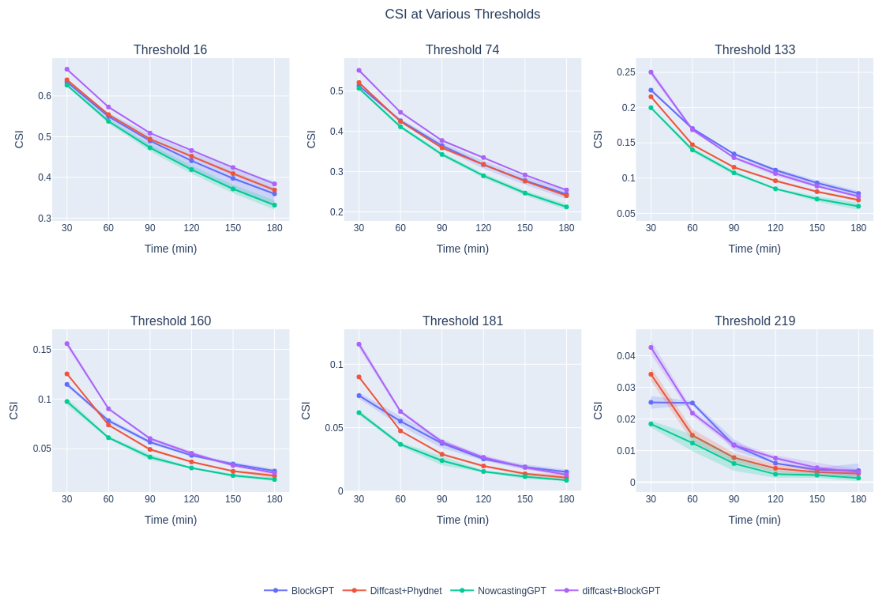


Figure 7.3: CSI values evaluated at different thresholds on the SEVIR dataset (task 1: predicting 3 hours, given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds.

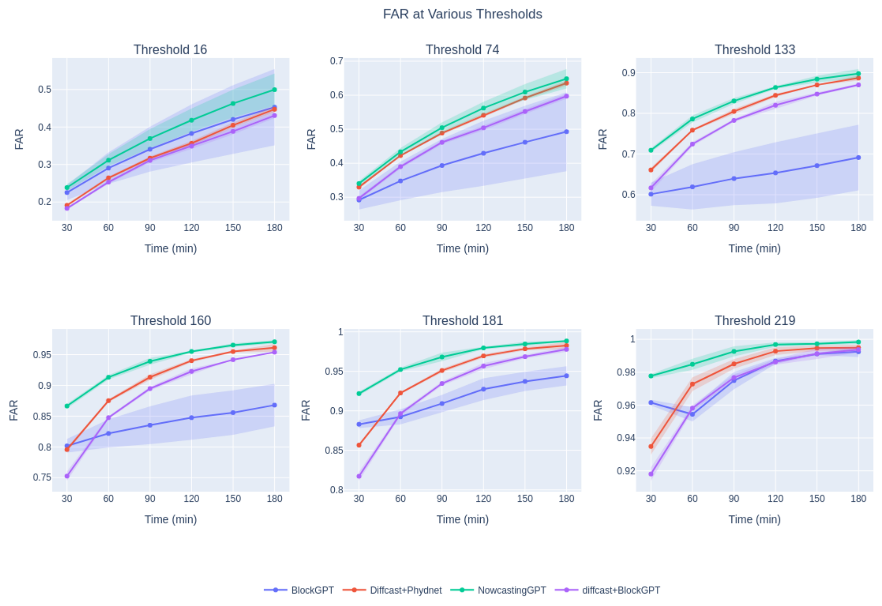


Figure 7.4: FAR values evaluated at different thresholds on the SEVIR dataset (task 1: predicting 3 hours, given 1 hour of context). Solid line and shading indicate average and standard deviation across seeds.

Other potential directions we discuss can be broadly categorised into two: how this core pipeline could be improved, and second, what additional, weather-specific modules can be used with BlockGPT to improve its performance.

NowcastingGPT, the token-by-token benchmark, which we fundamentally built upon, has two main bottlenecks- the token-by-token autoregression and discretisation. In this thesis, we only managed to improve the first bottleneck, but we still suffer from the second. The choice to use discrete latents arises mainly because of computational constraints and the ease of using LLM-based models to predict discrete tokens. Several work in the computer vision field improves upon this exact problem. For instance, [39] introduces a continuous version of an LLM, where instead of predicting discrete tokens, they use a small MLP on top of their autoregressive transformer to predict continuous latents (now vectors) from a VAE. Although we did attempt to include this in our model, training and evaluation times made it infeasible to benchmark it on multiple datasets and experiments. Another interesting solution to the discretisation problem is MAGVIT V2 [40], which introduced a new quantization technique which essentially allowed the model to have extremely large codebook sizes, while essentially having an embedding dimension of 0 (they show that the key to improving predictive performance is decreasing the embedding dimension and increasing the codebook size). However, to model such large codebooks, we need bigger transformers, which means we need more compute. Note that MAGVIT V2 is from Google, and they do not open-source it. But interesting replications exist [41].

Another very interesting direction the image generation community has taken is Visual Autoregressive Modelling [42]. Essentially, instead of generating a whole image at a time in the latent space, the authors first predict a low-resolution version, and then gradually refine it to higher resolutions. All intermediate steps occur in the latent space, meaning that they have separate latent spaces for each resolution. This allows the transformer model to essentially predict on multiple scales- first, predict the general shape in the low resolution, and refine it in later, high-resolution steps. At the time of writing this chapter, no equivalent version exists for videos, and thus building a model for weather in this line would also be a significant contribution to the video generation community. Being able to predict at a higher resolution could also help transformer-based models tackle their blurring effect.

Of course, we do not investigate in depth an entire field of computer vision- that of diffusion. This is because diffusion models in general are slow, and take a lot of compute to train in the same time scales as our existing models. Diffcast was clever in this regard because they only had to model the residual between the ground truth and its backbone prediction, and could thus afford smaller models. However, a flaw in their model is that they wait for the backbone's entire prediction, and then refine it. However, most backbones being autoregressive, accumulate error over time. An interesting direction to take residual modelling in weather could also be to refine predictions frame-by-frame, instead of only after the whole rollout.

On top of these computer vision advancements, which could fundamentally help our weather prediction pipeline, there is also much to be done in fine-tuning the architecture, particularly for weather. On top of our core architecture, future work could be deploying extreme value loss like [15] or imposing physics constraints, as was done in other work in the research group [43].

Bibliography

[1] J. Côté, C. Jablonowski, P. Bauer and N. Wedi, *Seamless prediction of the earth system: From minutes to months*, 2015 (cit. on p. 1).

[2] P. Bauer, A. Thorpe and G. Brunet, 'The quiet revolution of numerical weather prediction,' *Nature*, vol. 525, no. 7567, pp. 47–55, 2015 (cit. on p. 1).

[3] 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,' *Advances in neural information processing systems*, vol. 28, 2015 (cit. on p. 1).

[4] S. Ravuri, K. Lenc, M. Willson *et al.*, 'Skilful precipitation nowcasting using deep generative models of radar,' *Nature*, vol. 597, no. 7878, pp. 672–677, 2021 (cit. on pp. 1, 5, 6).

[5] S. Agrawal, L. Barrington, C. Bromberg, J. Burge, C. Gazen and J. Hickey, 'Machine learning for precipitation nowcasting from radar images,' *arXiv preprint arXiv:1912.12132*, 2019 (cit. on p. 1).

[6] European Environment Agency, *Economic losses from climate-related extremes in europe*, `https://www.eea.europa.eu/en/analysis/indicators/economic-losses-from-climate-related`, Accessed: 2025-05-22, 2024 (cit. on p. 2).

[7] Our World in Data, *Natural disasters by type*, `https://ourworldindata.org/grapher/natural-disasters-by-type`, Accessed: 2025-05-22, 2024 (cit. on pp. 2, 3).

[8] Centre for Research on the Epidemiology of Disasters (CRED) and UN Office for Disaster Risk Reduction (UNDRR), *The human cost of disasters: An overview of the last 20 years (2000–2019)*, `https://www.undrr.org/publication/human-cost-disasters-overview-last-20-years-2000-2019`, Accessed: 2025-05-22, 2020 (cit. on p. 3).

[9] X. Shi, Z. Gao, L. Lausen *et al.*, 'Deep learning for precipitation nowcasting: A benchmark and a new model,' *Advances in neural information processing systems*, vol. 30, 2017 (cit. on pp. 5, 6, 8).

[10] V. L. Guen and N. Thome, 'Disentangling physical dynamics from unknown factors for unsupervised video prediction,' in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 474–11 484 (cit. on pp. 5, 7, 9).

[11] Z. Gao, X. Shi, B. Han *et al.*, 'Prediff: Precipitation nowcasting with latent diffusion models,' *Advances in Neural Information Processing Systems*, vol. 36, pp. 78 621–78 656, 2023 (cit. on pp. 5, 8, 13).

[12] D. Yu, X. Li, Y. Ye *et al.*, *DiffCast: A Unified Framework via Residual Diffusion for Precipitation Nowcasting*, en, arXiv:2312.06734 [cs], Mar. 2024. [Online]. Available:

http://arxiv.org/abs/2312.06734 (visited on 02/09/2024) (cit. on pp. 5, 9, 10, 13, 49, 74).

[13] Z. Chang, X. Zhang, S. Wang *et al.*, 'Mau: A motion-aware unit for video prediction and beyond,' *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 950–26 962, 2021 (cit. on pp. 5, 10).

[14] Z. Gao, X. Shi, H. Wang *et al.*, 'Earthformer: Exploring space-time transformers for earth system forecasting,' *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 390–25 403, 2022 (cit. on pp. 5, 11, 12).

[15] C. Meo, A. Roy, M. Lică *et al.*, *Extreme Precipitation Nowcasting using Transformer-based Generative Models*, arXiv:2403.03929 [cs], Mar. 2024. DOI: 10.48550/arXiv.2403.03929. [Online]. Available: http://arxiv.org/abs/2403.03929 (visited on 19/09/2024) (cit. on pp. 5, 12, 13, 80, 98).

[16] M. Veillette, S. Samsi and C. Mattioli, 'Sevir: A storm event imagery dataset for deep learning applications in radar and satellite meteorology,' *Advances in Neural Information Processing Systems*, vol. 33, pp. 22 009–22 019, 2020 (cit. on pp. 10, 26, 30–32).

[17] L. Chen, Y. Cao, L. Ma and J. Zhang, 'A deep learning-based methodology for precipitation nowcasting with radar,' *Earth and Space Science*, vol. 7, no. 2, e2019EA000812, 2020 (cit. on p. 10).

[18] MeteoNet Project, *Meteonet: An open weather radar and observation dataset*, https://meteonet.umr-cnrm.fr, Accessed: 2025-05-22, 2024 (cit. on p. 10).

[19] Alibaba Cloud Tianchi, *Cikm 2017 competition: Precipitation nowcasting dataset*, https://tianchi.aliyun.com/dataset/1085, Accessed: 2025-05-22, 2024 (cit. on p. 10).

[20] W. Yan, Y. Zhang, P. Abbeel and A. Srinivas, 'Videogpt: Video generation using vq-vae and transformers,' *arXiv preprint arXiv:2104.10157*, 2021 (cit. on pp. 12, 15).

[21] A. Van Den Oord, O. Vinyals *et al.*, 'Neural discrete representation learning,' *Advances in neural information processing systems*, vol. 30, 2017 (cit. on pp. 12, 15, 16, 18).

[22] A. Vaswani, N. Shazeer, N. Parmar *et al.*, *Attention Is All You Need*, Version Number: 7, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762 (visited on 04/09/2024) (cit. on pp. 12, 19, 21).

[23] W. Yan, D. Hafner, S. James and P. Abbeel, *Temporally Consistent Transformers for Video Generation*, en, arXiv:2210.02396 [cs], May 2023. [Online]. Available: http://arxiv.org/abs/2210.02396 (visited on 02/09/2024) (cit. on p. 12).

[24] S. Pulkkinen, D. Nerini, A. A. Pérez Hortal *et al.*, 'Pysteps: An open-source python library for probabilistic precipitation nowcasting (v1. 0),' *Geoscientific Model Development*, vol. 12, no. 10, pp. 4185–4219, 2019 (cit. on p. 12).

[25] P. Esser, R. Rombach and B. Ommer, 'Taming transformers for high-resolution image synthesis,' in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 12 873–12 883 (cit. on p. 15).

[26] S. Ren, S. Ma, X. Sun and F. Wei, 'Next block prediction: Video generation via semi-auto-regressive modeling,' *arXiv preprint arXiv:2502.07737*, 2025 (cit. on pp. 15, 70).

[27] Y. Bengio, N. Léonard and A. Courville, 'Estimating or propagating gradients through stochastic neurons for conditional computation,' *arXiv preprint arXiv:1308.3432*, 2013 (cit. on p. 18).

[28] A. Overeem and R. Imhoff, *Archived 5-min rainfall accumulations from a radar dataset for the netherlands*, en, 2020. DOI: `10.4121/UUID:05A7ABC4-8F74-43F4-B8B1-7ED7F5629A01`. [Online]. Available: `https://data.4tu.nl/articles/_/12675278/1` (cit. on p. 26).

[29] National Oceanic and Atmospheric Administration, *Radar images: Reflectivity*, Accessed: 2025-05-06, Mar. 2025. [Online]. Available: `https://www.noaa.gov/jetstream/reflectivity` (cit. on p. 26).

[30] J. Marshall, W. Hitschfeld and K. Gunn, 'Advances in radar weather,' in *Advances in geophysics*, vol. 2, Elsevier, 1955, pp. 1–56 (cit. on p. 27).

[31] A. Roy, 'Extreme precipitation nowcasting using transformer-based generative models,' Accessed: 2025-05-02, Master's thesis, Delft University of Technology, 2024. [Online]. Available: `https://repository.tudelft.nl/record/uuid:8b54b913-56ac-46c1-9f95-a3bd20f00d80#files` (cit. on p. 27).

[32] H. Bi, 'Extreme precipitation nowcasting using deep generative model,' Accessed: 2025-05-02, Master's thesis, Delft University of Technology, 2022. [Online]. Available: `https://resolver.tudelft.nl/uuid:88b5ce73-5078-424c-a637-06d34efb475c` (cit. on p. 27).

[33] R. O. Imhoff, C. Brauer, A. Overeem, A. H. Weerts and R. Uijlenhoet, 'Spatial and temporal evaluation of radar rainfall nowcasting techniques on 1,533 events,' *Water Resources Research*, vol. 56, no. 8, e2019WR026723, 2020 (cit. on p. 27).

[34] Atlassian, *A complete guide to violin plots*, `https://www.atlassian.com/data/charts/violin-plot-complete-guide`, Accessed: 2025-05-02, 2024 (cit. on p. 28).

[35] University of Virginia Library, *Understanding q-q plots*, `https://library.virginia.edu/data/articles/understanding-q-q-plots`, Accessed: 2025-05-02, 2023 (cit. on p. 28).

[36] T. J. Schmit, P. Griffith, M. M. Gunshor, J. M. Daniels, S. J. Goodman and W. J. Lebair, 'A closer look at the abi on the goes-r series,' *Bulletin of the American Meteorological Society*, vol. 98, no. 4, pp. 681–698, 2017 (cit. on p. 31).

[37] S. J. Goodman, R. J. Blakeslee, W. J. Koshak *et al.*, ‘The goes-r geostationary lightning mapper (glm),’ *Atmospheric research*, vol. 125, pp. 34–49, 2013 (cit. on p. 31).

[38] National Weather Service, *Vil density*, Accessed: 2025-05-07, n.d. [Online]. Available: `https://www.weather.gov/lmk/vil_density` (cit. on p. 31).

[39] T. Li, Y. Tian, H. Li, M. Deng and K. He, ‘Autoregressive image generation without vector quantization,’ *Advances in Neural Information Processing Systems*, vol. 37, pp. 56 424–56 445, 2024 (cit. on p. 79).

[40] L. Yu, J. Lezama, N. B. Gundavarapu *et al.*, ‘Language model beats diffusion–tokenizer is key to visual generation,’ *arXiv preprint arXiv:2310.05737*, 2023 (cit. on p. 79).

[41] Z. Luo, F. Shi, Y. Ge, Y. Yang, L. Wang and Y. Shan, ‘Open-magvit2: An open-source project toward democratizing auto-regressive visual generation,’ *arXiv preprint arXiv:2409.04410*, 2024 (cit. on p. 79).

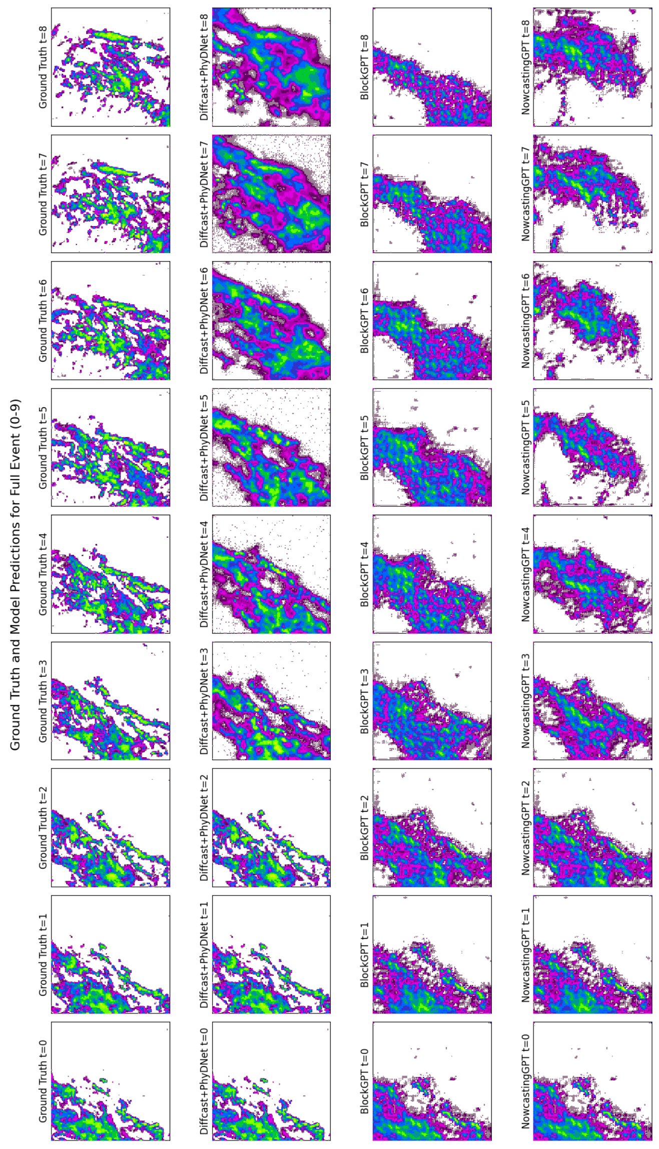
[42] K. Tian, Y. Jiang, Z. Yuan, B. Peng and L. Wang, ‘Visual autoregressive modeling: Scalable image generation via next-scale prediction,’ *Advances in neural information processing systems*, vol. 37, pp. 84 839–84 865, 2024 (cit. on p. 79).

[43] J. Yin, C. Meo, A. Roy *et al.*, ‘Precipitation nowcasting using physics informed discriminator generative models,’ in *2024 32nd European Signal Processing Conference (EUSIPCO)*, IEEE, 2024, pp. 967–971 (cit. on p. 80).

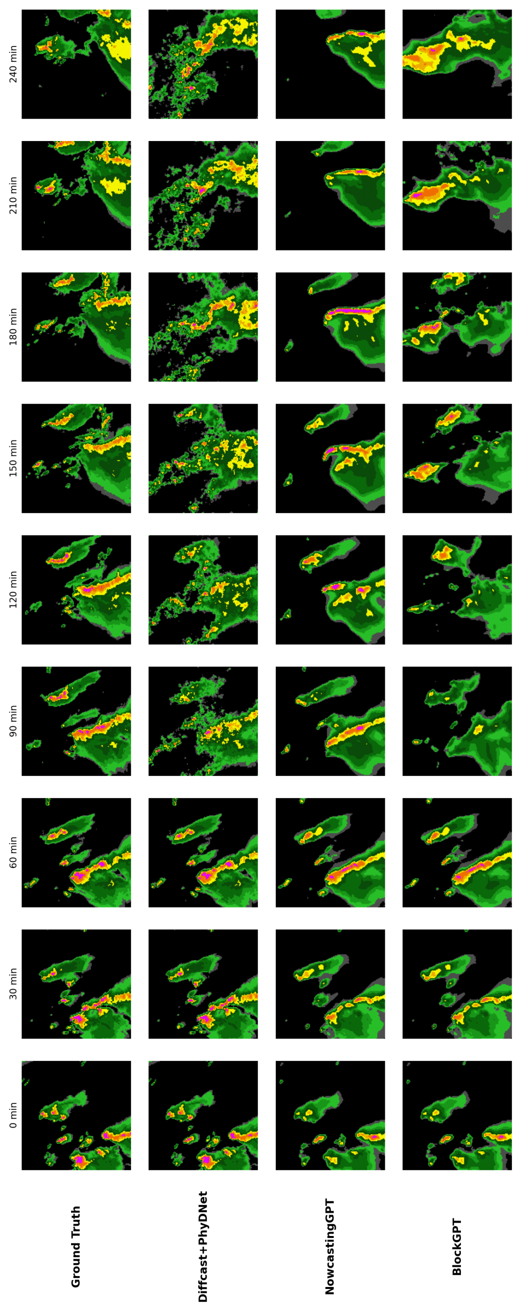
A Prediction Visualisations

A.1 Task 1: 3 Hour Prediction with 1 Hour Context

A.1.1 KNMI



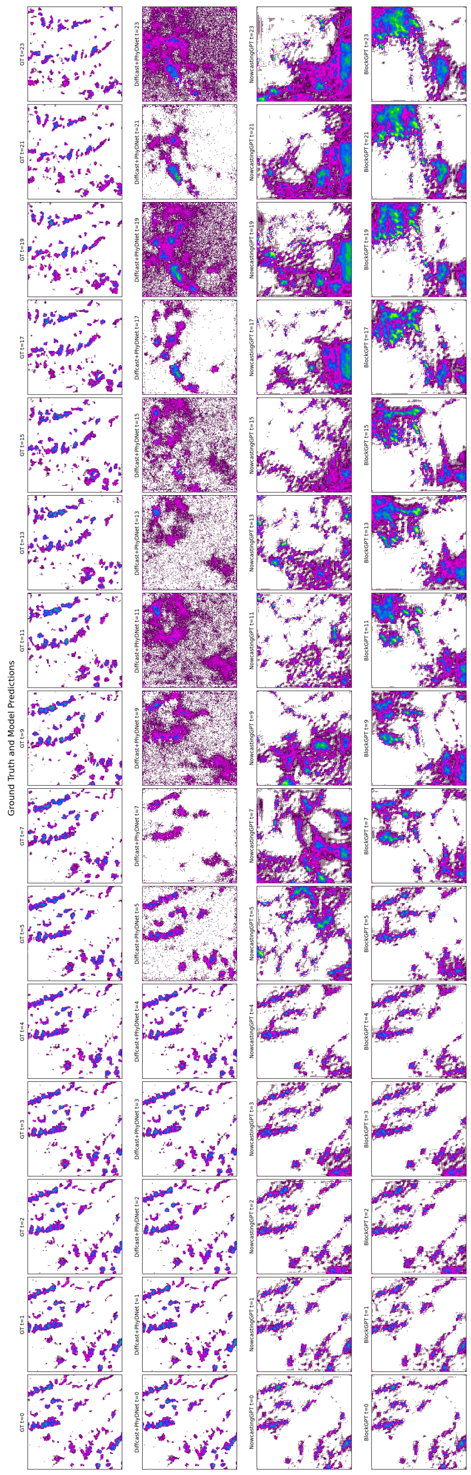
A.1.2 SEVIR



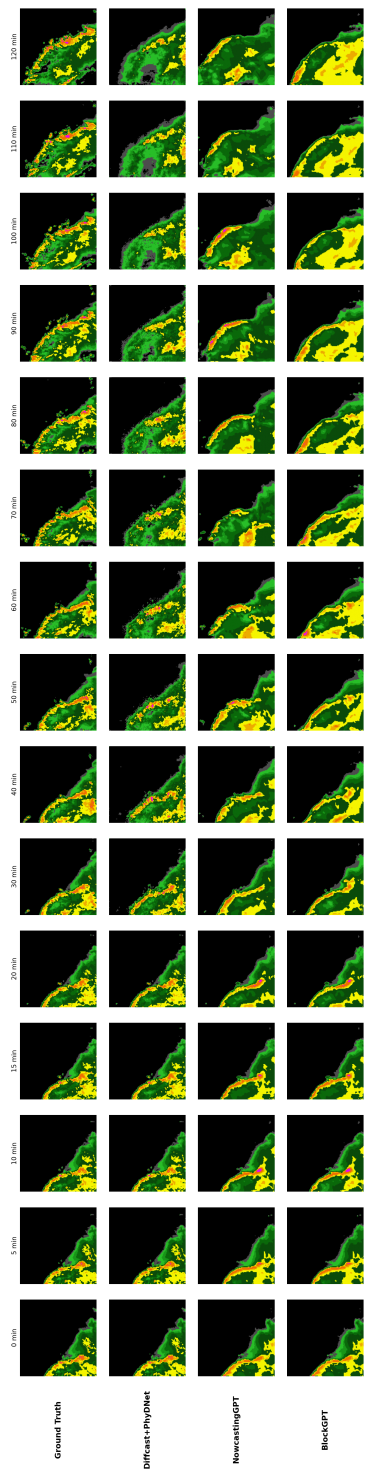
240 min
210 min
180 min
150 min
120 min
90 min
60 min
30 min
0 min
Ground Truth
Diffcast+PhyDNet
NowcastingGPT
BlockGPT

A.2 Task 2: 100 Minute Prediction with 25 Minutes Context

A.2.1 KNMI



A.2.2 SEVIR



120 min
110 min
100 min
90 min
80 min
70 min
60 min
50 min
40 min
30 min
20 min
15 min
10 min
5 min
0 min
Ground Truth
Diffcast+PhyDNet
NowcastingGPT
BlockGPT

Metrics on All Percentile Levels B

B.1 Task 1: KNMI

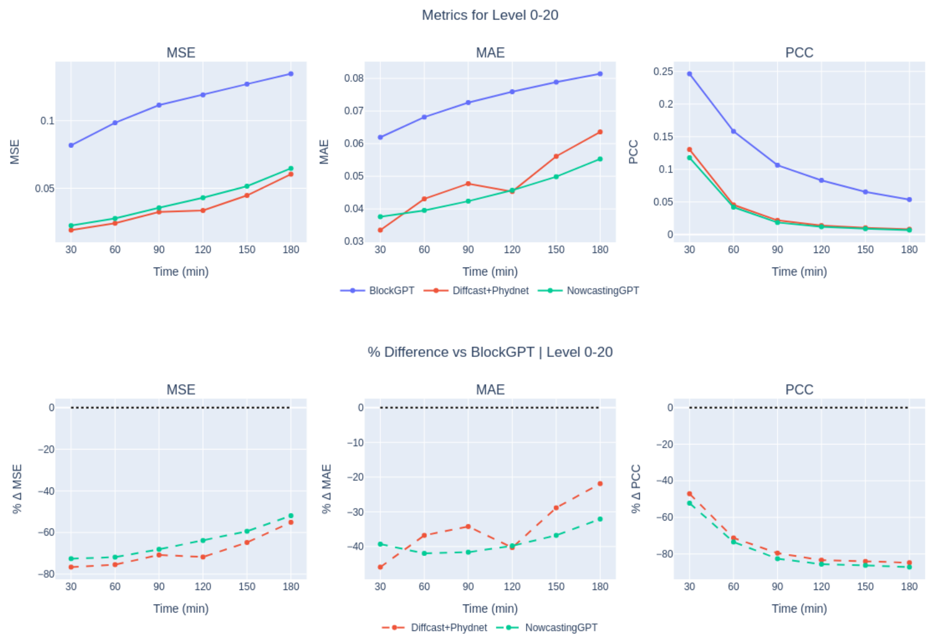


Figure B.1: KNMI Task 1. Metrics on Level 0-20 percentile.

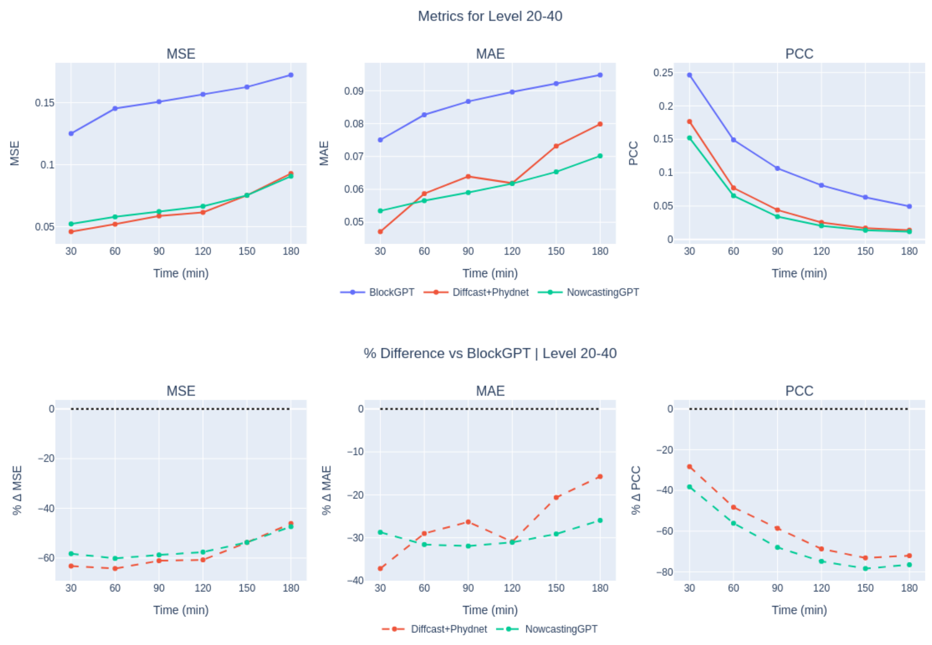


Figure B.2: KNMI Task 1. Metrics on Level 20-40 percentile.

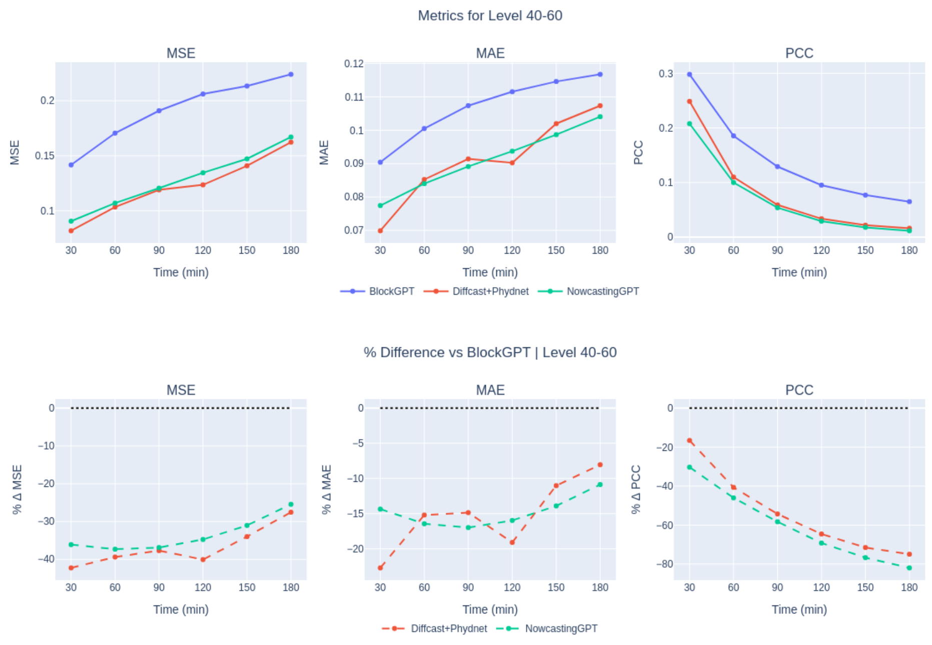


Figure B.3: KNMI Task 1. Metrics on Level 40-60 percentile.

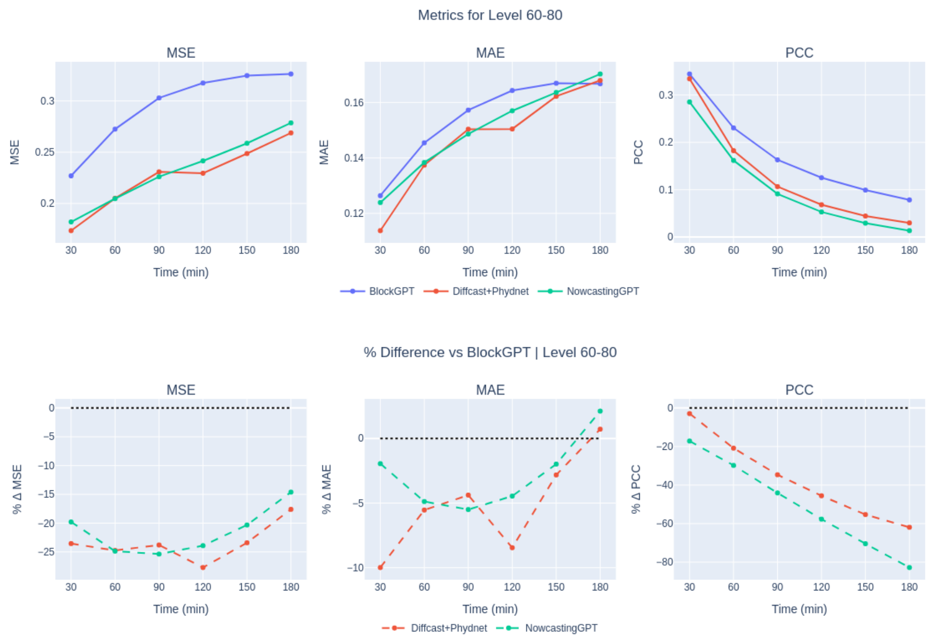


Figure B.4: KNMI Task 1. Metrics on Level 60-80 percentile.

B.2 Task 2: KNMI

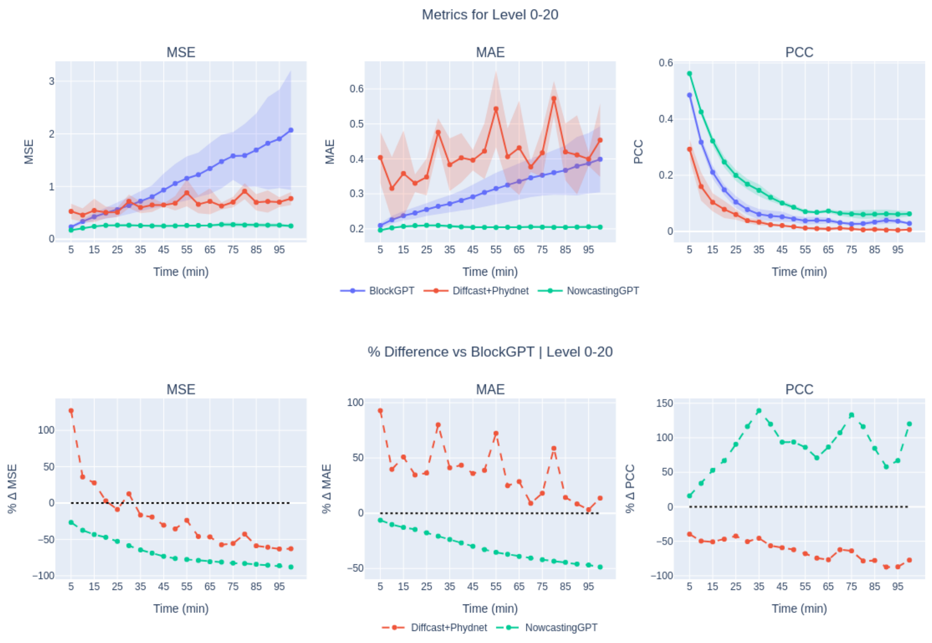


Figure B.5: KNMI Task 2. Metrics on Level 0-20 percentile.

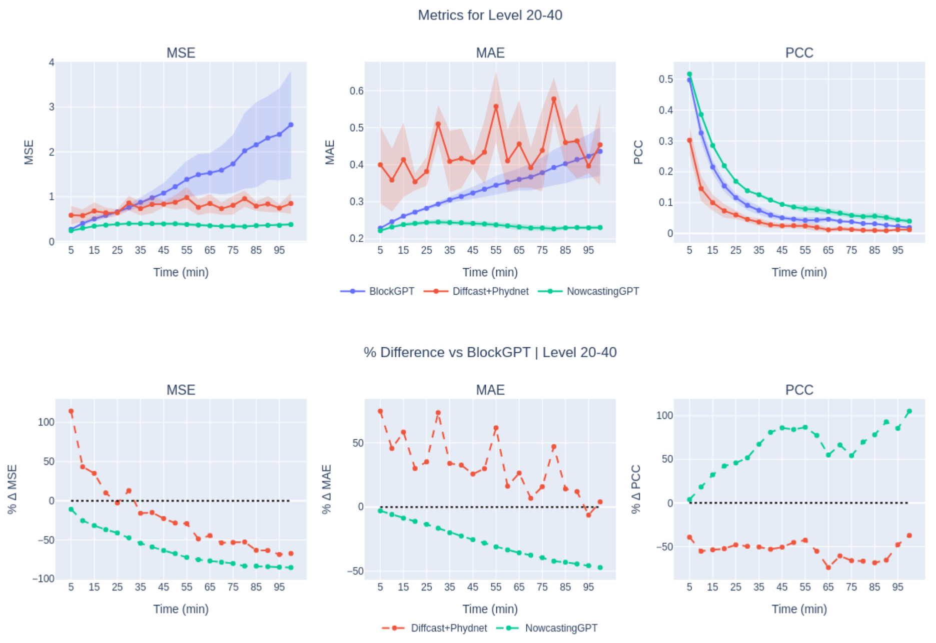


Figure B.6: KNMI Task 2. Metrics on Level 20-40 percentile.

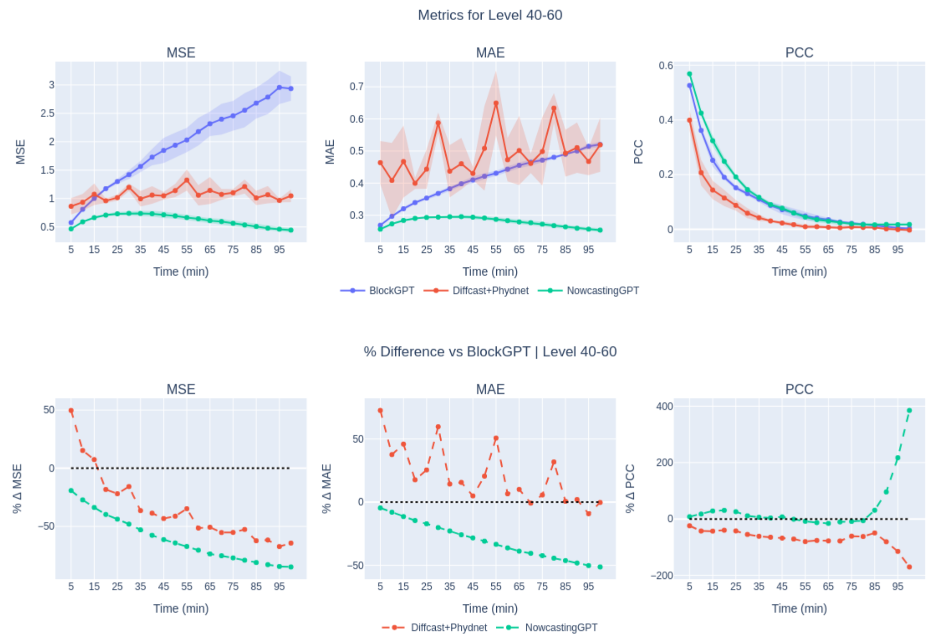


Figure B.7: KNMI Task 2. Metrics on Level 40-60 percentile.

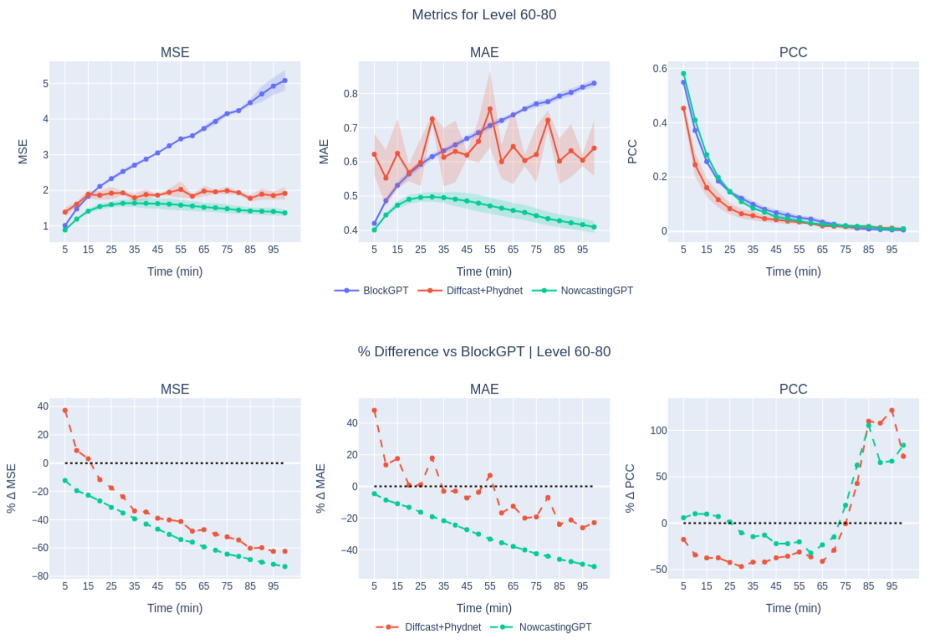


Figure B.8: KNMI Task 2. Metrics on Level 60-80 percentile.

B.3 Task 1: SEVIR

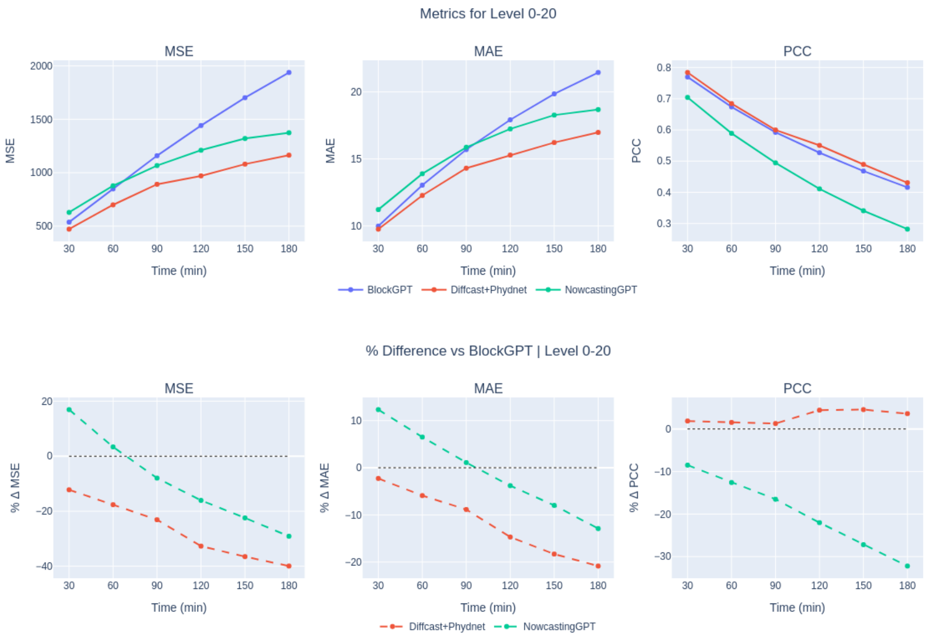


Figure B.9: SEVIR Task 1. Metrics on Level 0-20 percentile.

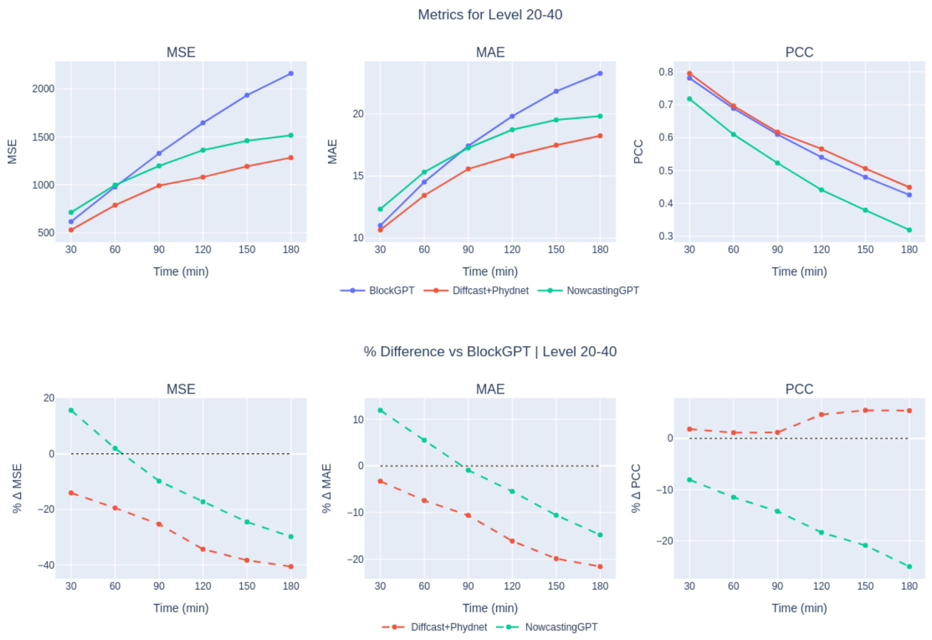


Figure B.10: SEVIR Task 1. Metrics on Level 20-40 percentile.

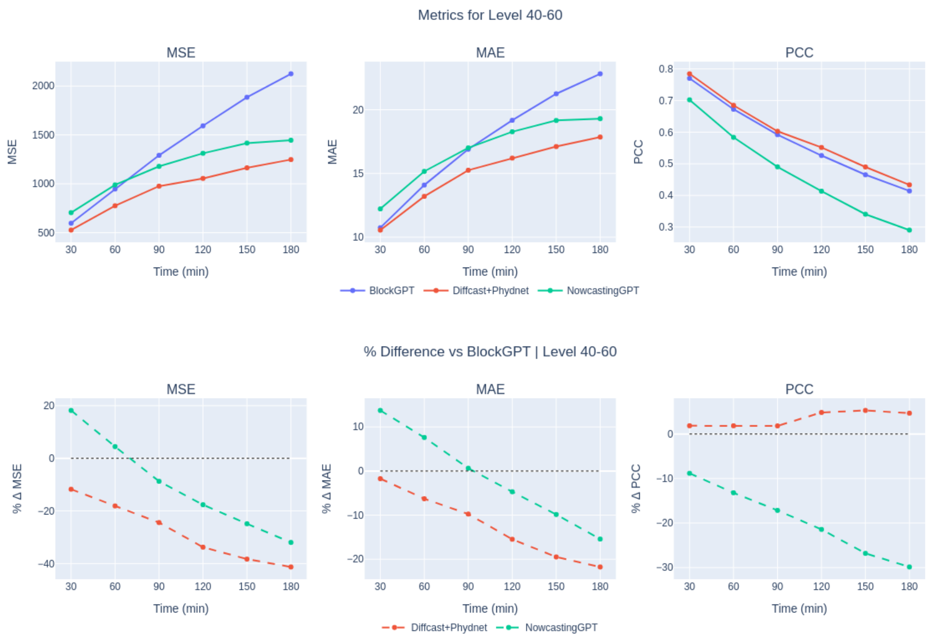


Figure B.11: SEVIR Task 1. Metrics on Level 40-60 percentile.

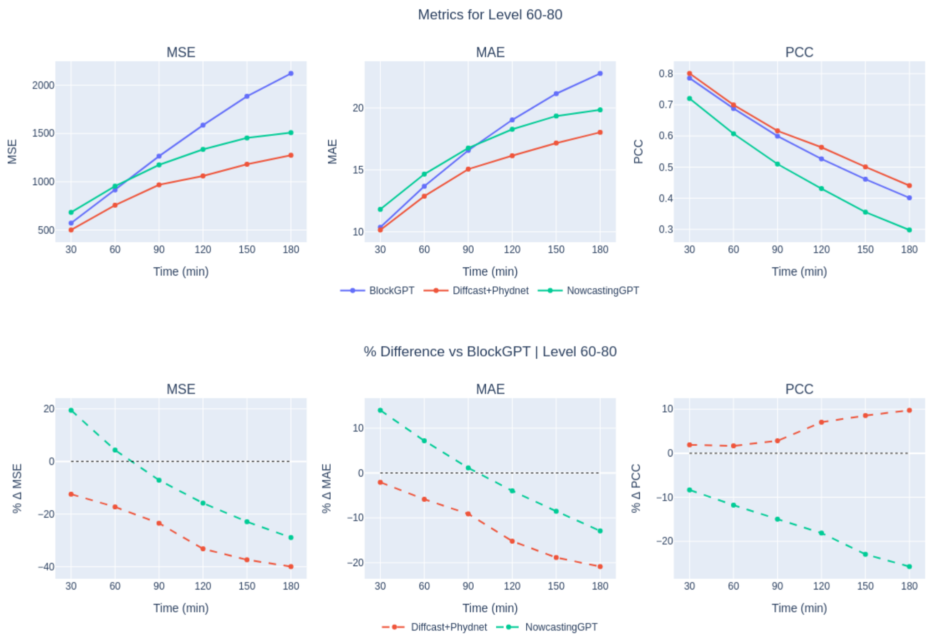


Figure B.12: SEVIR Task 1. Metrics on Level 60-80 percentile.

B.4 Task 2: SEVIR

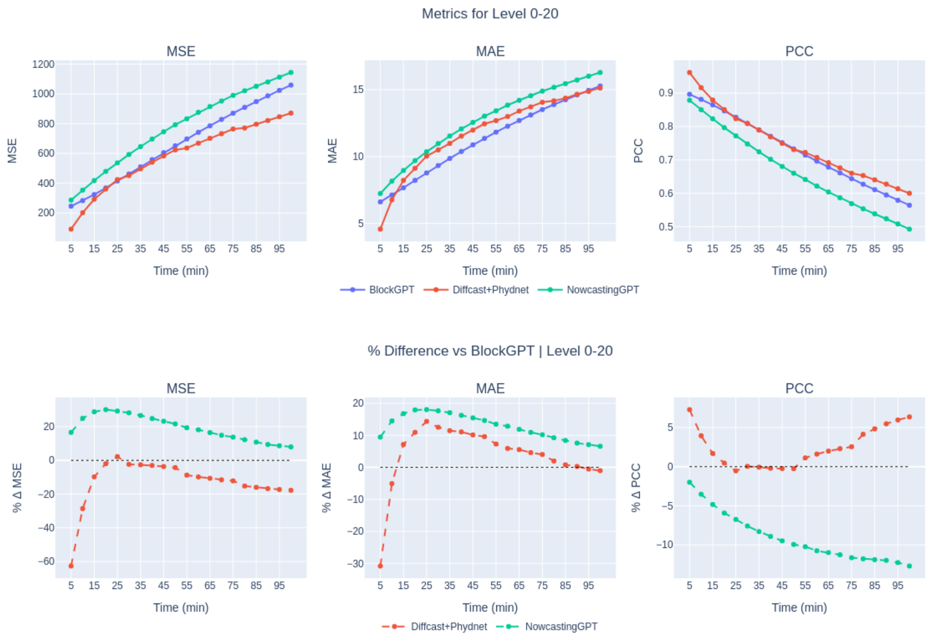


Figure B.13: SEVIR Task 2. Metrics on Level 0-20 percentile.

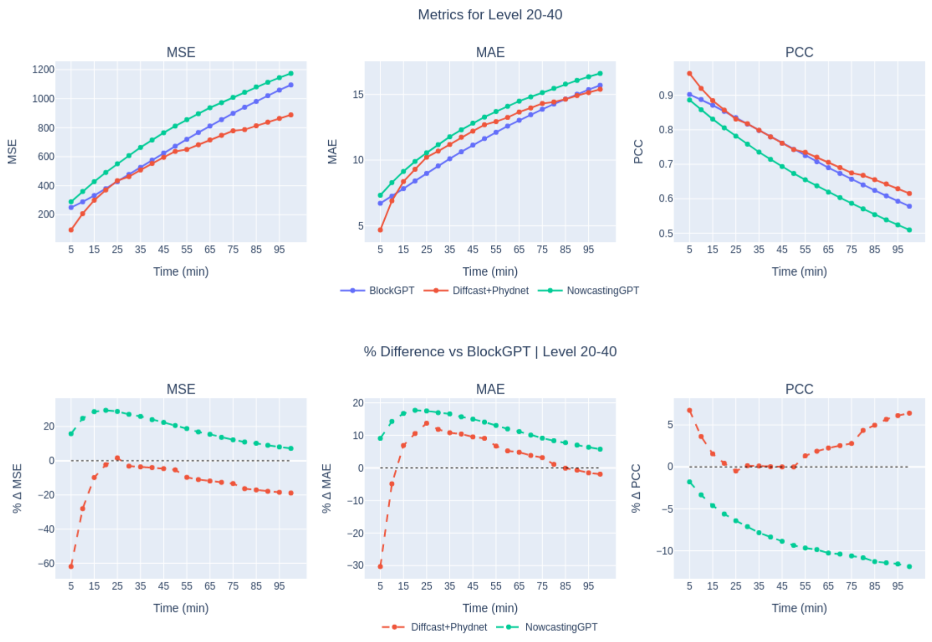


Figure B.14: SEVIR Task 2. Metrics on Level 20-40 percentile.

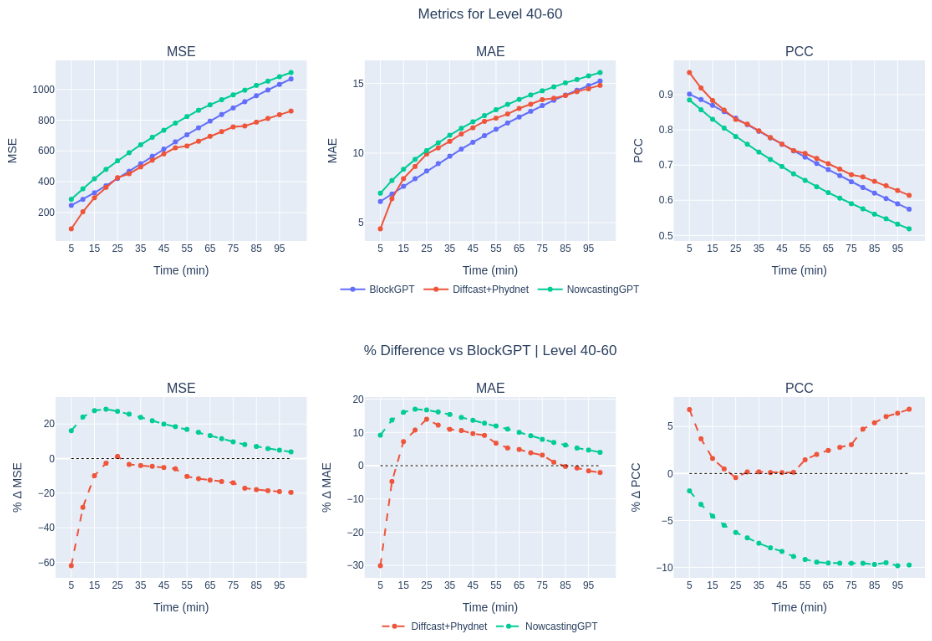


Figure B.15: SEVIR Task 2. Metrics on Level 40-60 percentile.

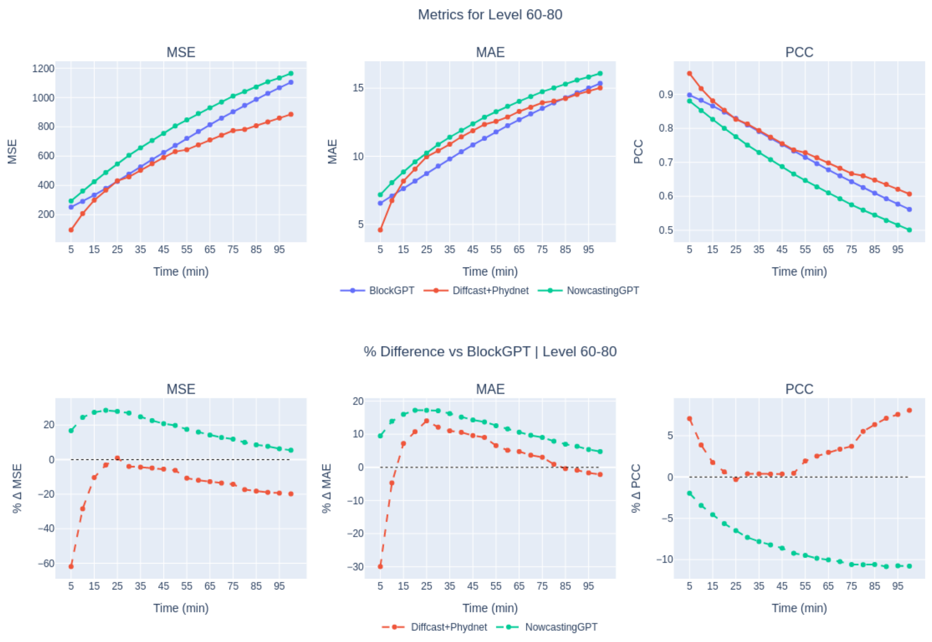


Figure B.16: SEVIR Task 2. Metrics on Level 60-80 percentile.

Model Parameters and Training Time

C

In this section, we summarize the training configurations and compute profiles of the three models compared in this work. A key goal in our design of BlockGPT was to match or outperform the benchmark models in terms of training time, while scaling model capacity up to the point of overfitting. The table below presents a comparison of parameter counts and training durations across all models.

Model	**Parameters**	**Training Time**	**Hardware / Epochs**
DiffCast	49.35M	∼15 hours	2 × A100 GPUs / 20 epochs
NowcastingGPT	150M	∼6 hours	2 × A100 GPUs / 20 epochs
BlockGPT (Ours)	103.37M	∼6 hours	2 × A100 GPUs / 20 epochs

Table C.1: Training time and parameter comparison across all models.

Our model, BlockGPT, was designed under the constraint of maintaining a training budget that is no greater than that of our benchmarks. Within this constraint, we maximize model capacity by increasing the number of parameters up to the point of overfitting or until the training time matches that of the benchmarks. This approach ensures a fair and efficient comparison while allowing us to explore the benefits of larger model capacity within realistic computational limits. We retain the original model configurations of Diffcast and NowcastingGPT. For the latter, we retain the same model parameters as those in the checkpoints in the github repository of [15]. The embedding dimension however, was originally only 128. We therefore retrain with the same embedding dimension as ours, for fair comparison.

BlockGPT's model configuration is as follows:

Table C.2: Essential architecture specifications for VQGAN and BlockGPT Transformer.

Component	**Parameter**	**Value**
VQGAN	Codebook Size	1024
	Latent Channels	128
	Attention Resolution	[8]
	Dropout	0.2
	Tokens per Frame	64
BlockGPT (Transformer)	Number of Layers	8
	Number of Heads	8
	Embedding Size	1024
	Token Block Size	576
	Vocabulary Size	1024