EPFL

Thèse n° 10 821

Extending Capabilities of Attention-based Models

Présentée le 27 novembre 2024

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 4
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Arnaud PANNATIER

Acceptée sur proposition du jury

Prof. A. M. Alahi, président du jury
Prof. P. Frossard, Prof. F. Fleuret, directeurs de thèse
Dr M. A. Kagan, rapporteur
Prof. R. Sznitman, rapporteur
Prof. A. Popescu-Belis, rapporteur

École polytechnique fédérale de Lausanne

2024

À ma famille.

Acknowledgements

This thesis is the end of a long journey and would definitively not have been possible without the help of many people.

First, I would like to thank my supervisor, Pr. François Fleuret. François has all the qualities that you can expect from a great mentor. He deeply loves what he is doing and he communicates effectively his passion to others. He is also one of the sharpest minds I know, which is great when you are stuck on a problem as it too often happens in the course of a thesis. And last, he is truly supportive. This PhD was hard in many aspects, but I could always count on his support. He always understands the complexity of the situation and offers concrete solutions.

I also extend my gratitude to Pr. Pascal Frossard for directing this thesis and helping me with administrative burdens coming from finding a good work-thesis balance. Thanks for helping to smooth things out with the doctoral school and for your trust in the process.

I would also like to thank Andrei Popescu-Belis, Michael Kagan, and Raphaël Sznitman for accepting to serve on my jury.

This PhD was done in partnership with Skysoft-ATM, and I would like to thank the company for the financial support and the opportunity to work on a real-world problem. In particular I would like to thank Didier B, Aurélien G, Ricardo P, Ludovic P, Robin P, Remi Z and Garabed H for their support and the fruitful discussions we had.

During my time at Idiap, I had the chance to meet awesome people. It's always nice to know people who are going through the same experience and the discussion around the coffee machine is often inspiring. Credits go to Alexandre B, Cédric T, Emmanuel P, Fabio F, Florian M, Florian P, François M, Juan Pablo Z, Ketan K, Louise C, and Laurent C.

The end of my PhD is also the end of my former group at Idiap, and I take the occasion to thank former group members for the interesting group meetings and the precious support. Many thanks to Angelos K, Evann C, Kyle M, Nikos D, Prabhu T, Sepher J, and Suraj S.

This PhD would not have been possible without the support of many lifelong friends and

family, whom I would like to acknowledge here. Thanks to Adrien C, Anne-Christine C, Antoine H, Bastian N, Benjamin & Aude E, Camille & Nico P, Cédric V, Chloé B & Tat D, Charline, Gaëlle & Marco C, Deb & Matteo G, Eliot J, Erwan, Tressy & Dimitri B, Etienne B, Gaëlle B, Ghika N & Amandine G, Romane & Giona L, Greg R, Juliane & Guillaume P, Laurel F, Ludo Z, Mathieu & Laetitia R, Mathilde G, Matteo, Sabine & Pablo L, Mélanie & Basile B, Nadia & Guy B, Quentin B, Romain D, Romain F, Samuel P, Simon M, Stéphane E, Valentin D, Vlad P.

To the grand-parents crew, I am forever grateful for always going the extra mile to ease our burdens throughout these years. Thanks to Mamimich, Papipich, Grand-Maman Béatrice, and Grand-Papa Polo for taking care of the kids and supporting us relentlessly during these demanding but beautiful times.

During these four and a half years, I have also experienced many life-changing events. Among them were your birth, Élise and Charlie. You both definitively disturbed the calm of our lives. But with the noise and the fatigue, came also the light and the meaning that makes our life whole now. Watching you grow up to become smart, happy, and playful kiddos has been the most rewarding experience of my life.

Finally, I want to thank you, Marion. This period was definitively challenging in many aspects, and between a global pandemic, a wedding, two tough pregnancies, and even tougher kids, we've been busy . . . Thank you for always being there, for stepping out of your comfort zone for me, and for sacrificing a lot to make it work. Now that this challenge might be nearly over, I'm looking forward to a maximum of other adventures with you on this fascinating journey of life.

Sierre, September 26, 2024 Arnaud

Abstract

Deep learning has brought incredible progress in areas where traditional algorithms struggle. It can be because they need rules that are sometimes hard to conceptualize, it can also be that the number and manner of combining these rules are too complex to grasp. Learning strategies, on the other hand, automatically leverage the hidden structures present in the data to achieve good performance on these complex tasks.

These progresses have recently reached general public consideration. Text-based assistants using Large language models (LLMs) are the main example, but deep learning models have proven their usefulness in many subfields of language, vision, and speech. While these domains are general purpose and benefit from data sets of colossal size, many more specific areas can similarly benefit from deep learning. This thesis focuses on attention models, which are the most versatile architectures in the field. We extend their capabilities making them capable of handling data coming from more unexplored application domains with specificities of real data.

First, we introduce the problem of high-altitude wind nowcasting and introduce a smart averaging model, which serves as a baseline. Next, we discuss the limitations of attention models and, consequently, present a sub-quadratic alternative based on multilayer perceptrons (MLPs). In the second part, we present an attention-based model as a solution to the high-altitude wind nowcasting problem. We show that this model is simple and robust and that attention layers help during learning. We also demonstrate that models do not need to have an encoder-decoder structure to do context-conditional inference. We conclude this work with the introduction of a new method for generating more complex signals. It challenges a traditional take: the assumption that the order of autoregression must be similar to the natural order of the data. We realized that during generation, models generally have a good estimate of the rest of the sequence. This allowed us to introduce a generation method with a sub-linear number of steps as a function of the number of elements, by exploiting the fact that it is often possible to generate several elements in parallel and validate those that are self-consistent.

These methods contribute to a current effort to broaden the application domains of deep learning to specific domains and data that are structured differently than those belonging

to the canonical fields of vision, speech, and language.

Keywords: Deep learning, Neural networks, Machine learning, Transformer models, Attention mechanisms, Air Traffic Control, Wind Nowcasting, Altitude Modeling, Random Order Autoregression, Sequence modeling

Résumé

L'apprentissage profond a apporté d'incroyables avancées dans des domaines où les algorithmes classiques peinent à être mis en place. Cela peut être dû au fait qu'ils nécessitent des règles parfois difficiles à conceptualiser, ou encore que le nombre et la manière de combiner ces règles sont trop complexes à comprendre. Les stratégies d'apprentissage, en revanche, exploitent automatiquement les structures implicites présentes dans les données pour obtenir de bonnes performances sur ces tâches complexes.

Ces progrès ont récemment atteint l'attention du grand public. Les assistants textuels utilisant des grands modèles de langage (LLMs) en sont l'exemple principal, mais les modèles d'apprentissage profond ont prouvé leur utilité dans de nombreux sous-domaines du langage, de la vision et de la parole. Si ces domaines ont la chance de bénéficier d'ensemble de données de taille gigantesque, il existe de nombreux domaines plus spécifiques qui peuvent bénéficier de manière similaire de cet apprentissage automatique. Cette thèse se concentre sur les modèles à attention, qui sont les architectures les plus polyvalentes du domaine. Nous étendons leurs capacités pour les rendre capables de gérer des données provenant de domaines d'application encore inexplorés avec les spécificités des données réelles.

Tout d'abord, nous présentons le problème de la prévision de vent à haute altitude et présentons comme référence un modèle de moyennage intelligent. Ensuite, nous discutons des limitations des modèles à attention et nous présentons une alternative sous-quadratique et basées sur les réseaux de neurones multicouches (MLPs). Dans une deuxième partie, nous présentons un modèle à attention comme solution au problème de la prédiction de vent à haute altitude. Nous montrons que ce modèle est simple et robuste, et aussi comment les couches à attention aident pendant l'apprentissage. Nous montrons également que les modèles n'ont pas besoin d'avoir une structure encodeur-décodeur pour être capable de faire de l'inférence conditionnée sur un contexte. Nous concluons ce travail par l'introduction d'une nouvelle méthode de génération de signaux plus complexes. Elle remet en question une approche classique : l'hypothèse que l'ordre de l'autorégression doit être similaire à l'ordre naturel des données. Nous avons réalisé que durant la génération, les modèles ont généralement une bonne estimation du reste de la séquence à produire. Cette réalisation nous a permis d'introduire une méthode de génération avec un nombre

d'étapes sous-linéaire en fonction du nombre d'éléments, en exploitant le fait qu'il est souvent possible de générer en parallèle plusieurs éléments et de valider ceux qui sont cohérents entre eux.

Ces méthodes participent à un effort actuel d'étendre les applications de l'apprentissage profond à des domaines spécifiques et à des données qui sont structurées différemment que celles qui appartiennent aux champs canoniques de la vision, de la parole et du langage.

Mots-Clés : Apprentissage profond, Réseaux neuronaux, Apprentissage automatique, Transformers, Mécanismes à attention, Contrôle du trafic aérien, Prévision immédiate du vent, Modélisation d'altitudes, Autorégression dans un ordre aléatoire, Modélisation de séquences

Contents

Introduction

Since the AlexNet revolution (Krizhevsky et al., 2012), deep learning has spread through academia and industry. The most recent deep learning models offer state-of-the-art results in image recognition (Dosovitskiy et al., 2020), image generation (Betker et al., 2023; Rombach et al., 2022), speech recognition (Radford et al., 2022), speech generation (Le et al., 2023), text classification (Devlin et al., 2019) and text generation (Radford et al., 2018, 2019; Brown et al., 2020). The burgeoning industry based on text generation alone is estimated to add 4.4 trillion annually to the global economy (Bakhtourine et al., 2023). These models leverage the availability of increasingly large and curated data sets for training their parameters. Pre-trained models are regularly made available, allowing companies to quickly fine-tune them for their custom needs. However, these pre-trained models mostly apply to common input modalities, while the specific nature of data sets in many industrial applications requires custom engineering.

This thesis focuses on adapting transformer architectures in situations encountered in real-world applications. It focuses on variants of the transformer architecture, which is known for its flexibility and scalability. Our work extends the capabilities of attention-based models to handle sparse data sets and to generate sequences, following a joint law, in only a few steps. Our key idea for handling sparse sets of measurement is to use a ViT-like architecture (Dosovitskiy et al., 2020) without sequence positional encoding to process both input and query points at the same time. For generating sequences, we train a GPT-like architecture (Radford et al., 2018) using a shuffled order to be able to generate sequences in any order, which we leverage in a burst-sampling scheme.

We illustrate the benefits of our approach by solving two key problems in Air Traffic Control (ATC): *wind nowcasting* and *ascent/descent forecasting*. In both tasks, we proposed a novel and efficient learning-based solution that outperforms previously used industry methods.

The MALAT project: Air Traffic Control in the Deep Learning Era

This thesis is articulated around the `MALAT: MAchine Learning for Air Traffic control` project, which intends to use a data-driven approach to solve two specific problems in ATC. The first task consists of inferring wind speed along flight tracks in a 30-minute window, a problem known as wind nowcasting. The second one consists of forecasting the ascent or descent rates of airplanes based on historical data. The following sections describe each of these tasks individually.

Better Wind Nowcasting

Wind nowcasting consists of inferring current local weather conditions based on the latest measurements taken instead of relying on infrequent and coarse updates given by national weather agencies. This data is valuable as the wind speed has some persistence, which implies that live measurements are usually a good predictor of the next ones in its vicinity. Furthermore, it is easily available, as this information is continuously broadcasted by airplanes to the control tower. This type of data, however, is challenging to process as it is sparse in space, and clustered along the main flight routes.

Better Ascent/Descent forecasting

The second part of the project focuses on forecasting the ascent and descent rates of airplanes. Controllers need reliable estimations of a Trajectory Predictor (TP) to manage airspace efficiently. This task is typically achieved through computational simulations based on a physical model. These physical models make several assumptions about aircrafts, rely on unknown parameters such as their current weight, and do not take into account behavioral parameters such as company policy, time of day, day of the week, or typically flight plans. Consequentially, the output of the TP regarding the ascent and descent of the airplanes is often an imprecise upper and lower bound. The large corpus of historical data offers the opportunity to switch from a physical-model-based approach to a deep-learning model trained on real data.

Challenges of specific domains

In both cases, the abundance of data and the flaws of current methods call for a deep learning solution. However, this approach brings several challenges:

- The domain lacks a large pool of previous models or approaches.

- The data sets have not been curated and preprocessing methods are nonexistent.
- The problem lacks benchmarks and metrics.

These problems are common once we move from general-purpose deep learning to specific domains. They require engineering to adapt architectures and methods to the specificities of the data and the tasks.

Getting the Best from Deep Learning: Attention and The Transformer Architecture

In recent years, transformers have emerged as the most versatile approach among the various deep learning architectures. Transformers were introduced in 2017 (Vaswani et al., 2017) as a novel state-of-the-art encoder-decoder architecture for translation. The novelty of the transformers was to let go of any kind of recurrent structure only to use attention mechanisms in a larger architecture composed of multilayer perceptrons and normalization layers.

These attention mechanisms were introduced in 2015 to allow Recurrent Neural Networks (RNN) to move information between distant tokens (Bahdanau et al., 2015). Since the original RNNs process sequences one token after another, they tend to forget previous tokens as they move forward into the sequence. This issue is particularly problematic in translation tasks, where corresponding words can be located at different positions due to differences in sentence structures. Attention mechanisms, as illustrated in Figure 1, can help with this issue. A key point that explains the success of the transformer architecture over an RNN with an attention model is that RNNs are complicated to parallelize. However, the recent introduction of the Mamba architecture (Gu and Dao, 2024) has proposed a novel parallelization procedure that could make RNNs competitive once again.

Motivated by their efficiency in solving translation tasks, transformers were adapted for text classification (Devlin et al., 2019) and text generation (Radford et al., 2018), achieving state-of-the-art performance in both domains. Various models have since emerged, scaling from 65 million parameters (`transformer-base`,Vaswani et al. (2017)) to architectures with hundreds of billions of parameters, such as LLama 3, with 400 billion parameters(AI@Meta, 2024).

These past years of scaling large models have led to the observation of several positive properties. First, as the data set and model size increase, their performance improvements do not plateau but seem to follow a power-law relationship (Kaplan et al., 2020). Additionally, as model performance increases, so does its out-of-domain capabilities. The gap between zero-, one-, and few-shot performance widens as the model scales, suggesting

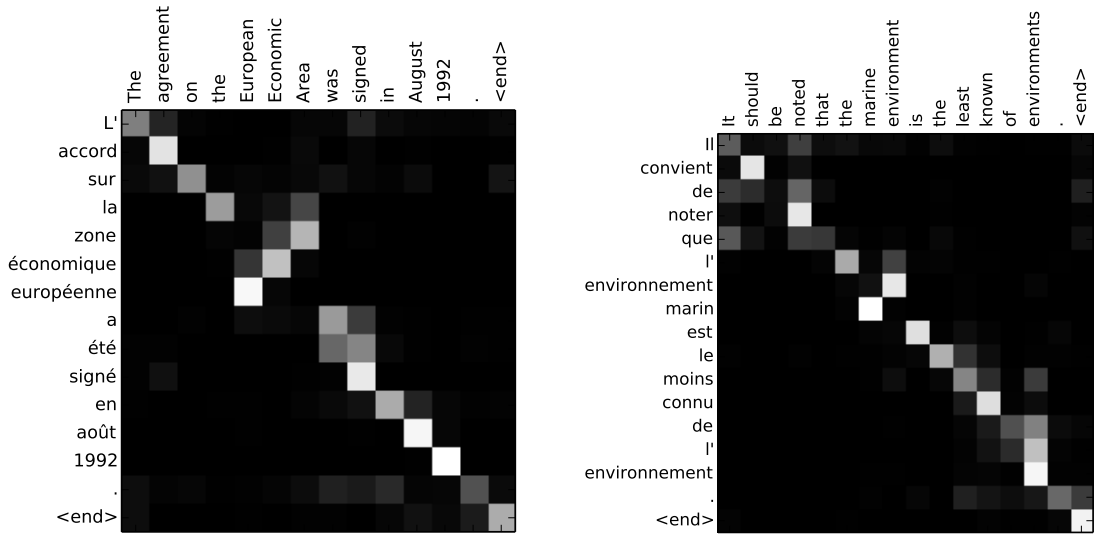


Figure 1: Two samples of attention maps from an RNN-based model for translation (Bahdanau et al., 2015), which puts weights on corresponding words in the source sentence for each word in the target sentence.

that larger models generalize better (Brown et al., 2020). Furthermore, limitations such as hallucinations, repetitions, and biases significantly decrease once the model reaches a certain size (Li et al., 2023a). Scaling these models is also data-efficient. As the models grow larger, they require fewer parameter updates, and hence fewer data points, to achieve the same performance as smaller versions of these models (Kaplan et al., 2020). Despite requiring careful engineering, large-scale computing, and vast data sets, scaling these models can be achieved almost without architectural changes.

The scalability and general-purpose nature of transformers have led to their use in various domains, including vision, speech, and text, where they led to significant performance increases. Before the paradigm shift induced by transformers, each domain had its specific architecture, each with its own inductive biases tailored to the target domain. For instance, in vision, Convolutional Neural Networks (CNNs) were the primary architecture for handling large input data, such as high-resolution images. CNNs were inspired by traditional visual filters, which were added to deep learning architectures. Relying on convolutions makes the model translation-equivariant, which is the ideal inductive bias for 2D signals. Engineering and refinements over multiple decades have led to efficient, parallel, and high-performance implementations of CNNs, and their large adoption in the broader community.

Despite not being designed to handle images, minimal adaptations of transformers have led to matching state-of-the-art CNN performances in vision. This was originally achieved by the introduction of Vision Transformer (Dosovitskiy et al., 2020), which is a standard transformer encoder applied to image patches. More surprisingly, transformers can now

also be used for image generation, despite a quadratic memory cost (Esser et al., 2021; Peebles and Xie, 2023; Sun et al., 2024).

To summarize, the transformer architecture exhibits the following properties:

- **General purpose:** Transformers have been successfully adapted to a wide range of domains.
- **Robust:** These adaptations do not require major tweaks.
- **Scalable:** Performances do not seem to plateau when increasing data set and model sizes. This scaling is data-efficient.

For the `MALAT` project, having a versatile architecture such as the Transformer is key, because we are dealing with specific domains that have not been extensively studied in the literature, and we need to ensure that the architecture can adapt to the peculiarities of the data and the tasks. Another key advantage of this across-domain flexibility is its applicability for multi-channel problems As transformers can be used for vision, NLP, and speech, it is well-adapted to combine multi-domain signals into the same processing architecture

Thesis outline and contributions

This thesis is structured in three parts: the first part describes alternatives to attention-based architectures and is based on (Pannatier et al., 2022) and (Mai et al., 2023a). The second part describes our solution for wind nowcasting using attention and is based on (Pannatier et al., 2024b). The last part tackles the generation of the joint signal and is based on (Pannatier et al., 2024a).

Part 1: Alternatives to attention-based architectures

In this part, we present two alternatives to attention-based architectures, focusing in particular on the self-attention mechanism. In Chapter 1, we present a simple neural-modulated Gaussian kernel averaging method as a baseline for the wind nowcasting problem. This averaging can be seen as a distance-based alternative to self-attention. In Chapter 2, we take a more general approach and replace the self-attention layer with a linear-memory low cost MLP-based alternative.

Nowcasting consists of extrapolating from the most recent context to make accurate predictions. The simplest baseline for this task is to create a meaningful context of

recent measurements and to average it. When these measurements are recent and close to the prediction point, this method is surprisingly effective due to the persistence of wind. However, this approach has two main problems. This first issue is that the scale of the different dimensions differs, requiring scaling for accurate comparisons. This point pushed us to introduce a neural-modulated Gaussian kernel, which acts as a distance-based attention method. The second issue is that a per-point approach is necessary to maintain the context's relevance; otherwise, the context becomes too diverse to be meaningful. In Chapter 1, we show how we implemented this solution and how we made it efficient.

In the second chapter of this part, we introduce a drop-in replacement to the self-attention mechanism. The biggest limitation of the self-attention mechanism is its quadratic complexity relative to the sequence length. We build upon MLPMixer (Tolstikhin et al., 2021), which first introduces the idea of replacing attention with MLPs for vision. An issue of MLPMixer is that the sequence length of the input is hard-coded in the first MLP, which makes it unusable for sequences of different lengths. We solve this problem by using hypernetworks applied to the input sequence to construct dynamic-size weights of the token-mixing MLP. We show that it is capable of mimicking the attention operation with a memory cost linear in terms of sequence length.

Part 2: An attention-based solution for wind nowcasting

In Chapter 3, we present our definitive solution for wind nowcasting. Our baseline presented in Chapter 1, had some complexity due to its per-point approach. To be able to allow a model to have a global instead of a per-point context, we need to have a flexible attention mechanism, and we need to be able to allow more processing than simple averaging.

This is exactly what a transformer does. In this chapter, we adopt a ViT-like model that takes as input both a set of measurements and corresponding positions alongside query positions and outputs forecasts at the queried positions. We compare it to other architectures known to be able to use irregular sets of sparse data. We show that our architecture reaches the best performance while being simpler than compared baselines.

We show that the attention mechanism allows each queried position to attend to every measurement in the context, without being restricted by a bottleneck which leads to better gradient flow during training. We also introduced a better way of encoding spatial measurements by sharing encoders between input measurements and queried positions and showed that it improved the performance on several real-world tasks.

Part 3: Modeling Sequences from a Joint Distribution using Transformers

Chapter 4 studies deep learning models that generate samples valid under a joint law. This was not a problem for wind nowcasting, as we made the implicit assumption that predictions depend only on the provided context and are not dependent on each other. This allows forecasting at all positions in one forward pass.

Airplane ascent and descent modeling was the motivating application of this chapter, and our goal was to generate valid trajectories based on historical data. We propose to see the problem as sequence modeling and use a causal transformer as a base model. We encountered two problems with this approach.

The first one arose from the nature of the data that we model. It is relatively small-scale and contains a large number of plateauing sequences. This biases the model and if, during autoregression, the same altitude is sampled twice in a row, it pushes the model to output a plateauing sequence even if the control altitude is still not reached. This problem is related to the repetition problem that appears sometimes with large language models (LLMs), or which can even be found more rarely in translation or traduction models and called 'hallucinations' (Li et al., 2023a).

As we have an explicit control, we tried to solve the problem in the following manner: instead of generating the sequence in a left-to-right manner, we can learn to model the sequences and generate them in a randomized order. This reduces drastically the repetition problem, as the model usually does not generate neighboring tokens one after another and it should therefore rely more on the control for early prediction.

We had a series of interesting findings during this research. First, even if substantially more costly in terms of training time, models were generally able to reach the same modeling performance as models trained in left-to-right models. Second, this allowed the model to make predictions anywhere in the remaining part of the signal. Showing that the model has good estimates of the rest of the task.

Part I

Alternatives to Attention-Based Architectures

Chapter 1

Efficient Wind Speed Nowcasting with GPU-Accelerated Nearest Neighbors Algorithm

This chapter is a post-print version of

> A. Pannatier, R. Picatoste, and F. Fleuret. *Efficient Wind Speed Nowcasting with GPU-Accelerated Nearest Neighbors Algorithm*, pages 325–333. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2022. doi: 10.1137/1.9781611977172.37. (pdf). (url). (code)

1.1 Introduction

In this chapter, we present a baseline for wind speed nowcasting. It introduces a neural averaging method that can be seen as a kind of proto-attention. It also shows the difficulties that can arise from working with sparse temporal data. In Chapter 3, we'll present a more advanced solution for wind speed nowcasting using transformers.

High-Altitude Wind speed nowcasting is crucial for air traffic management, as it directly impacts the behavior of airplanes. Even if the accuracy of numerical weather prediction still increases at an incredible pace (Bauer et al., 2015), and if traditional methods are well-suited for mid-long term forecasting (between 6h and 14 days), it remains less accurate than extrapolating the last measurements in the first few hours (Pulkkinen et al., 2019). Major actors are developing new weather nowcasting pipelines (Suman et al., 2021a) where the overall strategy for all nowcasting persists in getting good quality

measurements and extrapolating from them. In the case of high-altitude wind speed nowcasting, recent measurements are always available, as airplanes record the wind speed along their trajectory. To predict the wind at a given time and space, one needs to extrapolate from contexts made of the last measurements recorded in the vicinity of the desired point. Simple machine learning methods such as k-Nearest Neighbors (k-NN) can be employed to build such contexts. However, the required linear search for such approaches is demanding both in terms of computation and memory footprint, to the point that it may make them prohibitively expensive. A standard strategy used to reduce this cost consists of partitioning the space while relying on exact or approximate criteria to reject groups of samples, cutting the temporal and memory cost to a sub-linear function of their number. This is used for example in Locality Sensitive Hashing (LSH) (Gionis et al., 1999), or other tree-based methods (Bentley, 1975; Yianilos, 1970).

Besides the computational aspect, an essential characteristic of k-NN is the soundness of the distance function. Considering both the units and dynamic ranges of distinct features differ, their combination may be meaningless. A sound and straightforward strategy to address this issue is to scale features individually. The scaling factors can come from prior knowledge (*e.g.* sampling rates, instrument calibration, physical laws) or can be optimized from the data to maximize the performance. When training a model with temporal data, another problem arises: to make a forecast at any time, we have to exclude data points in the future, or according to a similar criterion, that may, for instance, integrate additional constraints related to processing time. This implies that some processing must be done before using k-NN and possibly for each data set point.

With this in mind, we propose a novel algorithm called Trajectory Nearest Neighbors (TNN) that tackles these two problems while being particularly adapted to point data sets that can be covered with cylinders, such as smooth trajectories. Each of these cylinders can be represented by a segment and an error distance term. In priority, our method explores points that belong to the nearby cylinders. It stops when all the nearest neighbors are guaranteed to be retrieved. Our approach is exact and uses basic linear algebra subprograms (BLAS) (NVIDIA, 2007) that can be computed efficiently on GPU using standard frameworks (Paszke et al., 2019; Abadi et al., 2016). This algorithm's strength comes from its simplicity while still offering a substantial increase in performance when used in the correct setup.

This chapter also presents a high-altitude wind nowcasting pipeline while comparing it with baselines and a particle model developed to reconstruct wind fields using a similar data set (Sun et al., 2017). Using TNN, we trained our model over a data set of 35 days and over 33 million points. Furthermore, our model managed to beat the other methods evaluated. We analyzed the speed-up of our approach, and it shows that in the context of GPU computing, our method seems to give the best speed-up compared to linear search and traditional methods while avoiding using any other libraries or language. We also demonstrate that when a data set cannot be well separated into trajectories, our approach

does not offer a substantial increase in performance. With this approach, we could speed up our pipeline's training by two orders of magnitude, making it achievable in a reasonable time. The main contributions of this chapter are the following:

- The introduction of a novel algorithm called Trajectory Nearest Neighbors (TNN) based on simple linear algebra and its theoretical analysis.
- An extensive comparison with traditional approaches (linear search, KDTrees.)
- A concrete application in the context of high-altitude wind nowcasting.
- An implementation of this algorithm using PyTorch (Paszke et al., 2019).
- The publication of a data set containing 33 million points measured along plane trajectories throughout 35 days. [1]

1.2 Related Works

Our works differ from other nowcasting pipelines that often remain oriented towards the forecasting of radar images such as rainfall (Suman et al., 2021a; Shi et al., 2017; Czibula et al., 2021) as it can leverage the grid structure of the data and use standard computer vision strategies. There are a few other works that were conducted on similar data sets (Sun et al., 2017; Kikuchi et al., 2018). The first one uses a particle model and we considered it as a baseline. The second one uses these measurements to select the best subset of an ensemble of weather forecasts. Recent works use CUDA (NVIDIA et al., 2020) to fetch nearest neighbors (Garcia et al., 2010; Chen et al., 2017). However, the algorithms they study require an Euclidean metric and no temporal coordinates, making any comparison with our method complicated.

1.3 Methods and Technical Solutions

1.3.1 General Description of Wind Nowcasting Pipeline

We have at our disposal Mode-S data recordings for 35 different days over European airspace, and we represent a subset in Figure 1.1. Mode-S data is exchanged between Secondary Surveillance Radars (SSR) and the aircraft radar transponders (Wikipedia contributors, 2021), and contains, among others, the position of the plane and measurements of the wind. SSRs rotate with a period of 4 seconds, setting the sampling time for these

[1] Code and data sets are available at github.com/idiap/tnn

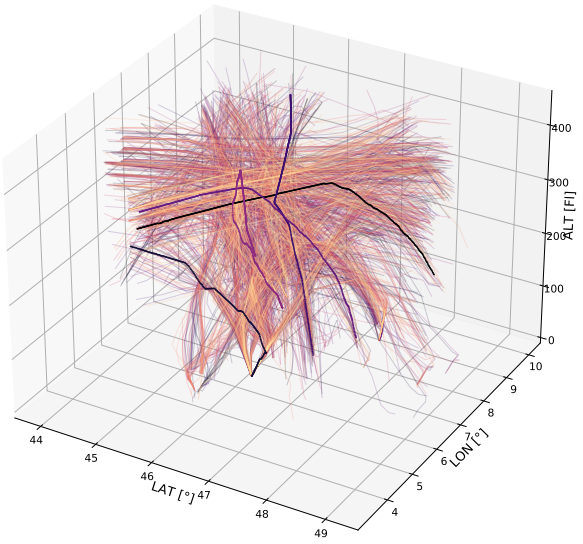


Figure 1.1: All the plane trajectories recorded on `Day #1` are plotted. Most of them are clustered along some main routes. A few trajectories are highlighted, which show the density of the tracking points.

variables. The pipeline for training the model is the following: batches of random samples are sampled iteratively. For each of these samples, a context of past measurements is retrieved, where each context contains only points that were measured at least t_w minutes in the past (t_w is typically 30 minutes) and from which the forecasts are then extrapolated. We use the Root Mean Square Error (RMSE) to measure the performance of the models.

1.3.2 High-Altitude Wind Nowcasting

We extrapolate the wind speed measurements using Gaussian Kernel Averaging (GKA) (Friedman et al., 2001) to give the forecast $\hat{s}$ at point (x, y, z, t):

$$\hat{\vec{s}}(x,y,z,t) = \frac{\sum_{k\in C} e^{\sigma_{xy}\left[(x-x_k)^2+(y-y_k)^2\right]+\sigma_z(z-z_k)^2+\sigma_t(t-t_k)^2}\vec{s_k}}{\sum_{k\in C} e^{\sigma_{xy}\left[(x-x_k)^2+(y-y_k)^2\right]+\sigma_z(z-z_k)^2+\sigma_t(t-t_k)^2}} \tag{1.1}$$

Where $\vec{s_k}$ is the wind speed measured at (x_k, y_k, z_k, t_k), $(\sigma_{xy}, \sigma_z, \sigma_t)$ are parameters of the model and C is the context of the point.

1.3.3 A Proto-attention Model for Wind Nowcasting

In the context of wind, the physical (x, y, z) and temporal dimensions (t), where the measurements are recorded, are not directly comparable. The x, y units are in latitude and longitude, z is in flight levels, and t is in seconds. To make the comparison meaningful, we standardize each dimension and then scale them using a scaling factor $\vec{\sigma}$. The parameters $\vec{\sigma}$ are physical constants that are obtained through careful geostatistical analysis but, more practically, by training them with backpropagation. This is true for our model but also for the baselines (kNN, GKA with a single scalar) that we consider.

The context C of eq. (1.1) can *a priori* contain all the previous measurements, which makes the computation prohibitively expensive. To make the training feasible, we decided to reduce the number of points by only taking the few contributing the most to the sum. This approach is reasonable, as most points will only have a negligible contribution to the final weighted average, and in later sections, we will show how our method can speed up this process.

Additionally, to make the comparison meaningful, we can also let the parameters $\vec{\sigma}$ be a function of the space $\vec{\sigma}\colon (x, y, z) \mapsto (\sigma_{xy}, \sigma_z, \sigma_t)$. This allows the model to resize the weights anywhere in the space. For example, in a region with numerous measurements, it could be wise to reduce the scaling factor to put more weight on only a few closer neighbors. Conversely, increasing the scaling might be better to average more points in a region where the measurements are sparse. It also allows the model to adapt the output based on the region of the space, by learning some topographical patterns that impact the wind field.

We model this relationship with a multilayer perceptron, composed of a few linear layers interleaved with ReLU non-linearities, that takes the spatial coordinates as input and outputs the scaling factors.

$$(\sigma_{xy}, \sigma_z, \sigma_t) = \mathrm{MLP}(x, y, z) \tag{1.2}$$

Combining this model with the Gaussian Kernel Averaging model (Equation (1.1)) gives us a model that can be seen as a proto-attention mechanism: the model learns the weights, which are then used to scale the distances in the averaging process.

1.3.4 Partitionning Measurements in Segments

Retrieving a context of points involves finding their nearest neighbors in the data set recorded before a specific time. The traditional nearest neighbors algorithm typically partitions the space and explores only a tiny number of partitions. Given that our measure-

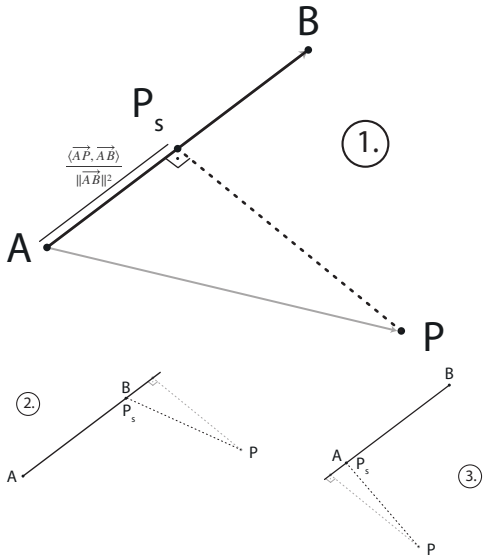


Figure 1.2: Illustration of the projection and the clamping process used to find the distance between one point P and a segment s. 1. If the projection of P falls between the point starting point A and endpoint B, no clamping is needed, and we can retrieve the points using a simple geometric formula. 2-3. If the projection falls outside the segment, the closest point in s would be A or B.

ments are clustered along some routes, as shown in Figure 1.1, and that we have to filter according to a temporal criterion, the traditional approaches such as Locality Sensitive Hashing (LSH) (Gionis et al., 1999), KDTrees (Bentley, 1975), Ball Trees (Omohundro, 1989), and Vantage Trees (Yianilos, 1970) are not well suited. Moreover, because masking is complex to implement for these structures, the algorithm explores many partitions containing only non-valid measurements. When working with data recorded along trajectories, the natural way of splitting the space is to split these trajectories into segments. Masking can then be done efficiently by only looking at the timestep of the first point associated with the segment. Once the measurements are split into segments, one can filter the valid segments and retrieve the nearest neighbors by only exploring nearby segments.

1.3.5 Theoretical Analysis

We have a data set of N measurements from wind speed at location $\vec{x}_i = (x_i, y_i, z_i, t_i) \in \mathbb{R}^4, i \in \{1, \dots N\}$ and planes only record the x, y components of the wind speed $\vec{s}_i = (s_i^x, s_i^y) \in \mathbb{R}^2, i \in \{1, \dots N\}$. For the sake of simplicity, we split the measurements information $\vec{x}_i$ into a point $P_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ and a time-step t_i. We will use the following scalable metric:

$$\|\vec{x_1} - \vec{x_2}\|_{\vec{\sigma}}^2 = \sigma_{xy}[(x_1 - x_2)^2 + (y_1 - y_2)^2] + \sigma_z(z_1 - z_2)^2 + \sigma_t(t_1 - t_2)^2 \tag{1.3}$$

This is split into a spatial and a temporal part:

$$\|P_1 - P_2\|_{\sigma_{xyz}}^2 = \sigma_{xy}[(x_1 - x_2)^2 + (y_1 - y_2)^2] + \sigma_z(z_1 - z_2)^2 \tag{1.4}$$

$$\|t_1 - t_2\|_{\sigma_t}^2 = \sigma_t(t_1 - t_2)^2 \tag{1.5}$$

Where $\vec{\sigma} = (\sigma_{xy}, \sigma_z, \sigma_t)$ is a scaling factor that allows a correct comparison between dimensions. All the points in the data set belong to trajectories that we split into sets $T_j = \{\vec{x}_{j,1}, \ldots, \vec{x}_{j,K}\}$, $j \in \{1, \ldots, \frac{N}{K}\}$ of exactly K elements and where the points are sorted along the time dimension. We pad the last set with copies of the first and last elements if the cardinality of the sets is not K. Each set T_j can be approximated by a segment s_j between the first point (A) and the last point (B) of the set, where A and B correspond to the spatial part of the measure $\vec{x}_{j,1}$ and $\vec{x}_{j,K}$.

The minimum distance d between a point P recorded at time t and a segment s is given by:

$$d = \text{dist}((P, t), s) = \|P - P_s\|_{\sigma_{xyz}}^2 + \|t - t_s\|_{\sigma_t}^2 \tag{1.6}$$

where P_s is the projection of P on the segment AB, and t_s^w is the projection of t on $[t_A, t_B]$ [Figure 1.2]:

$$P_s = A + \max\left(0, \min\left(\frac{\langle \overrightarrow{AP}, \overrightarrow{AB} \rangle}{\|\overrightarrow{AB}\|_2^2}, 1\right)\right) \overrightarrow{AB} \tag{1.7}$$

$$t_s = \max(t_A, \min(t, t_B)) \tag{1.8}$$

When approximating a subtrajectory $T = \{\vec{x}_1, \ldots, \vec{x}_K\}$ by a segment s, we are making an approximation error for each point $i \in \{1, \ldots, K\}$ which can be upper bounded:

$$\begin{aligned} E_{\text{app}} &= \max_{i \in 1, \ldots, K} \text{dist}(\vec{x}_i, s) \\ &= \max_i \|P_i - P_{s_i}\|_{\sigma_{xyz}}^2 + \underbrace{\|t_i - t_{s_i}\|_{\sigma_t}^2}_{0} \\ &\leq \sigma_{\max} \max_i \|P_i - P_{s_i}\|_2^2 \end{aligned} \tag{1.9}$$

with $\sigma_{\max} = \max(\sigma_{xy}, \sigma_z)$ and where the temporal part of the distance vanishes as $t_A < t_i < t_B$ by construction. This upper bound can be computed efficiently, as only the maximum Euclidean distance of the points to the segments is required.

By slightly modifying Equation (1.8), one can mask all the segments that contain points only measured after a certain time window t_w (typically 30 minutes) before t:

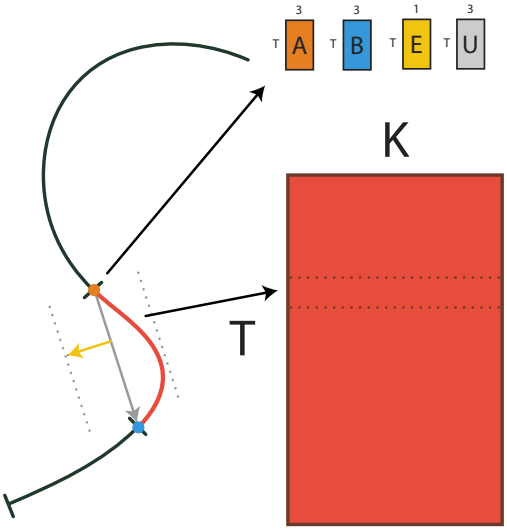


Figure 1.3: Each trajectory is split into sub-trajectories (here in red) that are approximated by a segment (gray) and an error term (yellow). A $T \times K$ matrix holds the indices of all points belonging to each segment. To be able to compute the distance efficiently, we group the segments' information in matrices **A**, **B**, **E**, **U**.

$$t_s^w = \begin{cases} \infty & \text{if } t_A > t - t_w \\ \min(t, t_B) & \text{otherwise} \end{cases} \tag{1.10}$$

The masked distance is then given by:

$$d_w = \|P - P_s\|^2_{\sigma_{xyz}} + \|t - t_s^w\|^2_{\sigma_t} \tag{1.11}$$

If we take into account the error that we are making when approximating a set of points by a segment s, we can know that all these points are at least at a distance $(d_w - E_{\text{app}})$ from a point P. Thus, this criterion can be used to exclude segments in search of neighbors if one already has a candidate for the k-th neighbors that are closer than $d_w - E_{\text{app}}$.

1.3.6 Trajectory Nearest Neighbors Algorithm (TNN)

The algorithm takes as an input a batch of M points and for each retrieves its k nearest neighbors. It is written in a tensorial way, making extensive use of broadcasting, which is efficiently implemented in standard libraries such as PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2016). The first task, described in Algorithm 1, is to compute the distance from all the batch points to all the segments, using Equation (1.11) while

taking the approximation error into account. If the approximation error is larger than the estimated distance from the point to the segment, this distance is set to zero. The pieces of information required to compute the distance to a segment are its starting point A measured at time t_A, ending point B measured at time t_b, and the error term E. As the unitary vector $\vec{u} = \frac{\overrightarrow{AB}}{\|\overrightarrow{AB}\|_2^2}$ needs to be known for all queries, we can precompute it and cache its value. All these quantities are arranged in matrices $\mathbf{A}$, $\mathbf{t_A}$, $\mathbf{B}$, $\mathbf{t_B}$, $\mathbf{U}$, $\mathbf{E}$ for efficiency [Figure 1.3]. A pseudocode version showing how this formula can be broadcasted can be found in Appendix A.2.

Algorithm 1 Distance from point P to a segment, details about the notation can be found in Section 1.3.6

Given P, t,$\vec{\sigma}$, A, t_A, B,t_B,U, E, t_w
Return Distance from P to segment with error
if $t_A > t - t_w$ **then**
  return ∞
end if
$\delta = \text{clamp}((P - A) \cdot U, 0, 1)$
$P_s = A + \delta * (B - A)$
$D = \|P_s - P\|^2_{\sigma_{xyz}} + \sigma_t \max(t_B - t, 0)^2$
$D = D - E \max(\sigma_{xy}, \sigma_z)$
return $\text{clamp}(D, 0)$

Algorithm 2 Trajectory Nearest Neighbors

Given A batch of points, information about the segments
Output k-Nearest Neighbors
Distances = compute distances to segments (Algorithm 1)
Distances = sort distances
Furthest neighbors = ∞
Next distance = Distances[:, 0]
i = 1
d = 0
while Furthest neighbors $\geq$ Next distance or $d = M$ **do**
  Fetch F segments of K points for the remaining $(M - d)$ points in the batch
  Compute distance from batch points to segment points
  Current nearest neighbors = sort previous (k) and new points (FK)
  Furthest neighbor = Current nearest neighbors[:, k]
  d = nb of completed lines
  Put completed lines (d) at the end of the batch
  Next distances = Distances[:, $i * F$]
  i += 1
end while
return k-Nearest Neighbors

Once the first task is completed, the algorithm searches for the neighbors in the closest F segments [Algorithm 2], which consists of fetching all the points contained in a few of them at a time, then computing the actual point-to-point distance between the points in the batch and all the points contained in the segments. At that stage, points that are recorded after a time window t_w before t are masked. The algorithm then sorts the point by distance and keeps only the first k ones. At this point, it has k candidates for the nearest neighbors and has to decide if it is necessary to continue the search. If the distance from the furthest candidate is smaller than the distance to the next segment while taking the error term into account, it is guaranteed that no other points can be closer, which means that the neighbors are found. If the algorithm is already completed for some points in the batch, it puts them at the end of the batch and continues to process only the remaining points. For that, it fetches points in the next closest segments until the following segments are too far to include any valid candidates. This continues until the neighbors for all the points in the batch are found.

1.3.7 Performance and Memory analysis

By splitting the dataset into segments, TNN reduces the number of comparisons needed to find the nearest neighbors. The following table describes the number of operations and the memory required by the linear search and the TNN algorithm [Table 1.1]. To compute the linear search, we used the distance matrix trick, which consists of rewriting the distance matrix $||x_i - x_j||_2^2$ in an expanded form $||x_i||_2^2 + 2x_i^T x_j + ||x_j||_2^2$, which is more efficiently computed (Albanie, 2019). In both cases, the bottleneck is the top-k algorithm and the sorting subroutine needed to find the nearest neighbors. The distance-to-segment subroutine [Algorithm 1] requires computing the distance matrix from all points to the segments, using [Equation (1.11)]. As the number of segments is roughly $\frac{N}{K}$, this offers a substantial improvement over the linear search.

Let us define the mean number of segments n_f that we have to consider for one point in the nearest neighbors' search. Hence the algorithm has to fetch a $n_f \times F \times K$ points for all N points. This factor depends heavily on the data set. For example, it is lower when the segments are a good approximation of the trajectories and when they are well separated. In the following section, we illustrate how this factor's variability applies in an experiment with synthetic data through consideration of both a usual and a pathological case One can see that our method offers a substantial decrease in storage footprint as well. This is crucial when working with GPU as the memory is often limited. We computed the theoretical cost for a single batch as most of the memory can be freed directly after usage.

Table 1.1: Time-complexity and Space-Complexity estimation for computing the nearest neighbors of all points in the data set. N is the size of the data set. n_f is the mean number of times we have to fetch all K points in F segments in the nearest neighbors' search. The storage is described for a batch of M points as most space can be directly freed after the operation. The total considers only the sorted matrix and the top-k neighbors as it is what the memory contains at peak use.

Method	Steps	Time Complexity	Space Complexity
Linear search	Distance Matrix	$O(N^2D)$	$M(2N+1)$
	Top-k	$O(N^2)$	$2Mk$
TNN	Distance Segments	$O(\frac{N^2}{K}D)$	$M\frac{N}{K}D$
	Sort	$O(\frac{N^2}{K}\log(\frac{N}{K}))$	$2M\frac{N}{K}$
	Distance to points	$O(Nn_fFKD)$	$M(k+FK)D$
	Top-k	$O(Nn_f(k+FK))$	$2M(k+FK)$
	Total	$O(\frac{N^2}{K}\log(\frac{N}{K}) + Nn_fFKD)$	$M\frac{N}{K}D + M(k+FK)D$

1.4 Empirical evaluation

The wind-speed nowcasting pipeline and the trajectory nearest neighbors algorithm (TNN) are evaluated in different settings. Some baselines are used to compare the performance of our model, while TNN is tested against some other nearest neighbor algorithms.

1.4.1 High-Altitude Wind Nowcasting

In Table 1.2, some simple baselines, a particle model (Sun et al., 2017), and our GKA models are evaluated. We use Root Mean Square Error (*RMSE*) to assess the accuracy of our forecasts. We start with simple and understandable models as baselines: The average wind of the whole day and the last hour before the prediction point. Then we evaluate k-NN, where k is optimized to give the best accuracy on a validation set. When $k = 1$, this model is equivalent to the persistence model. For the day average model, we used a non-causal approach and predicted for each point in the test set the mean wind speed value for the day. The first causal baseline we considered is the hour average model. This model outputs the mean of all the points measured in the interval $[t - 1\text{h}30; t - 30\text{min}]$ for the prediction of a given point a time t. As expected, those baselines are far from optimal, but they give an informative way of comparing our model. Training the naive k-NN and GKA baselines was feasible on a single data day, but it becomes intractable to train on the whole data set without using a local context. The particle model (Sun et al., 2017) is giving results similar to our baseline, but it probably could be improved by optimizing the hyperparameters of this model. TNN allowed us to retrieve a small number of neighbors based on a scaled and masked metric. This procedure made the training over the whole

Table 1.2: Results of the different models for three days in the data set, where the mean wind speed is specified for each day. The baselines are in *italic* and the total duration for one epoch is mentioned for a single-day data set (around 1mio measurements) and for a five-week data set (around 33mio measurements). The *k*-NN and GKA baselines' optimizations are done with SciKit-Opt using Random Forest. We compare our work with a particle model (Sun et al., 2017) and a *k*-NN model, which relies on the persistence of the wind speed. Our methods are optimized with Adam using the whole training set, but the averaging set for each point is restricted by TNN, which reduces the training time by about two orders of magnitude.

	RMSE [kn]			Epoch duration	
Model	**Day #1**	**Day #2**	**Day #3**	**1 day data set**	**5 weeks data set**
Mean wind	95 [kn]	49 [kn]	39 [kn]	hh:mm:ss	hh:mm:ss
Day Average	27.87	20.19	13.86	0:03	2:05
Hour Average	26.19	17.51	12.67	0:34	20:00
Particle Model	9.98	10.07	7.84	6:57:15	1121:54:30
GKA	9.07	9.64	7.66	2:39:18	481:47:20
k-NN | *Persistence*	9.02	9.86	7.57	4:31:47	558:37:05
GKA - TNN	8.71	9.19	7.55	**4:13**	**1:35:30**
GKA - MLP - TNN	**8.01**	**8.51**	**6.87**	**4:21**	**1:37:39**

data set possible while beating the existing baselines' performance. The time taken to evaluate all the points in the data set is also mentioned in Table 1.2. It shows that using TNN to build a proper context is mandatory to train our models by offering a speedup of 320 times compared to the fastest baseline. Training models on the whole data set is mandatory to test more advanced strategies. And when letting the parameters of GKA vary through space, we managed to increase the precision of our model.

1.4.2 Comparison with Linear Search

A comparison between TNN and Linear Search is made on two synthetic data sets and a real one. The first one consists of smooth random walks. The second represents a pathological case where all points are taken randomly and are considered to be measured simultaneously. The final one uses actual wind speed measurements. Our method substantially increases the performance by drastically restricting the search space, as shown in Figure 1.4 and table 1.3. The average number of comparisons on CPU is approximately 30 times lower than the linear search. Changing the number of segments F that are fetched simultaneously in Algorithm 2 impacts the speed-ups, This is because when bringing many segments at a time, more comparisons can be made simultaneously, but it may perform some unnecessary ones. This explains why there is a sweet spot for the average speed-up in Figure 1.4. The same sweet spot effect can be observed by increasing the number of measurements per segment K.

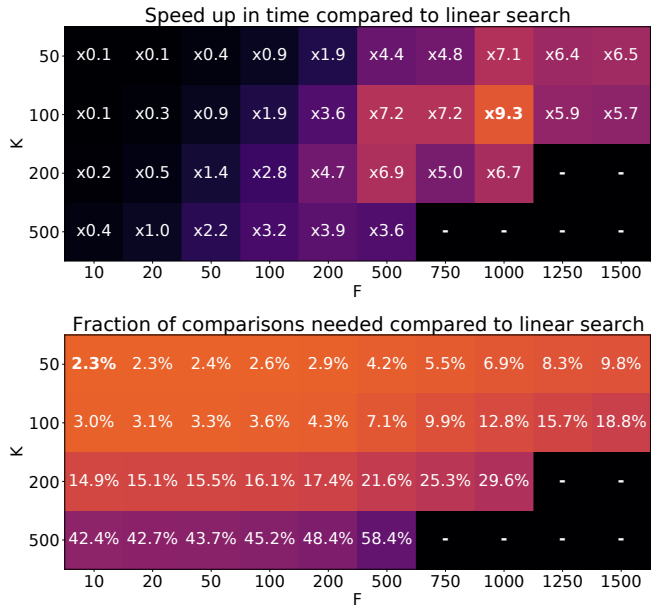


Figure 1.4: A comparison between linear search and TNN is made on GPU for different parameters K (number of points per segment) and F (number of segments fetched at the same time) as explained in Section 1.3.6. The first grid refers to the average speedup reached by TNN compared to linear search (higher is better). The second grid shows the average percentage of comparisons that TNN needed to find the nearest neighbors (lower is better).

Our method substantially increases the CPU efficiency by dividing the search time by almost one order of magnitude. We see that running the linear search on GPU instead of CPU already offers a substantial decrease in query time due to the parallelization properties of GPUs. TNN exploits this property well, and it runs 16 times faster than linear search on GPU both for the smoothed random walk and the original data set, as can be seen in Table 1.3. The trajectories are well separated in both data sets, and TNN increases the efficiency similarly. We evaluate a pathological case where points are randomly distributed over the space and randomly grouped in segments, leading to a significant approximation error E_{app}. In that setting, while querying for nearest neighbors, the algorithm cannot filter far away segments and thus, resorts to checking all the points. In that case, TNN uses the same number of comparisons as the linear search and does not offer any speed-up as expected. Indeed, the approximation error is significant in such a case, as the algorithm will have to consider almost all segments in the data set. One solution to this problem could be reorganizing the data points so segments can approximate the resulting sets more appropriately.

Table 1.3: Mean number of comparisons and time needed for querying 1000 nearest neighbors on the original wind and smoothed random walk (SRW) data sets. As TNN is an exact method, the query results are the same for both methods. It shows as well the total duration (`hh:mm:ss`) needed to retrieve the neighbors for the whole data set. All these comparisons are made on a data set of a million points.

	Device	Algorithm	Comparisons	Query [ms]	Total duration
Original Data set	CPU	Lin. Search	811'372	9.03	2:30:32
		TNN	**28'579**	1.03	17:08
	GPU	Lin. Search	811'372	2.55	42:28
		TNN	81'611	**0.16**	**2:43**
SRW Data set	CPU	Lin. Search	1'000'000	11.95	3:19:09
		TNN	**58'408**	1.60	26:35
	GPU	Lin. Search	1'000'000	2.51	41:48
		TNN	93'555	**0.39**	**6:28**
Random points	CPU	Linear Search	1'000'000	7.43	2:03:51
		TNN	999'940	15.51	4:18:34
	GPU	Linear Search	1'000'000	1.72	28:42
		TNN	**998'588**	**1.36**	**22:40**

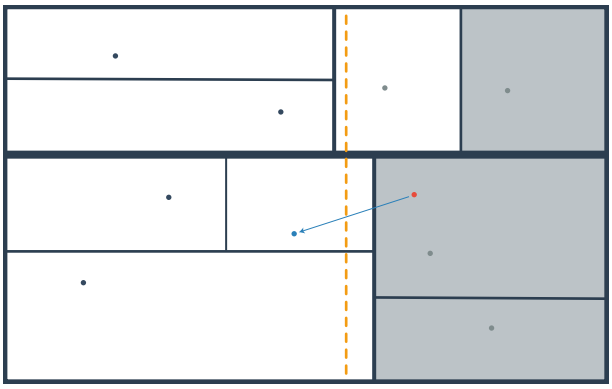


Figure 1.5: The KDTree algorithm explores cells that are close to the query point first. Setting a hard limit on a given axis complicates the procedure as the cells close to the goal might not contain any valid candidate. The black lines represent the KDTree structure. The query point is depicted in red, and the nearest neighbor that respects the mask (in orange) is blue. The cells that should not be considered are colored in grey.

Table 1.4: Comparison with KDTrees and linear search on GPU and CPU for the original data set. The creation of the different structures has to be performed once at the beginning of the program. The comparison details are given in Section 1.4.3. It shows the average time for querying 1000 nearest neighbors on the different data sets and the total duration (`hh:mm:ss`) needed to retrieve the neighbors for the whole data set. All these comparisons are made on a data set of a million points.

Method	Creation [s]	Query [ms]	Total duration
Linear search CPU	–	9.03	2:30:32
TNN CPU	1.00	1.03	17:08
Scaled masked KDTree	0.11	9.25	2:35:06
Scaled masked cKDTree	0.03	**0.70**	**11:35**
TNN GPU	7.00	**0.16**	**2:43**
Linear search GPU	–	2.55	42:28

1.4.3 Comparison with KDTrees

We compared our method to a KDTree implementation that extends the one proposed by Scikit Learn (Pedregosa et al., 2011). The goal here is to compare the two methods on concrete examples and see which one is better suited for finding nearest neighbors in a Machine Learning context. We evaluated a version in pure python (*Scaled Masked KDTree*) and an optimized one in Cython (*Scaled Masked cKDTree*) and show the results in Table 1.4. We changed the metric in KDTrees to have a scalable query and retrieve only the point after a given time window. One can see that, on the original data set, the scaled and masked KDTree implementation in pure Python is on par with the linear search, which highlights that KDTrees are not designed to work with masked data. This was not their intended use, as KDTrees explore in priority cells close to the query point. However, if one dimension is masked, many of these cells contain no valid candidates resulting in a substantial decrease in performance as more cells need to be explored. An example of this situation is shown in Figure 1.5. An extensive ablation study is made in Appendix A showcasing this phenomenon. This bad performance can be mitigated by using an optimized implementation in Cython, and we see that, on CPU, cKDTree is the fastest method. However, our approach still offers a substantial increase even when used on CPU where it cannot benefit as much from the GPU parallelization capabilities. Furthermore, the KDTree algorithm is not appropriate for running on GPU as it requires many non-parallelizable operations. The real advantage of our method is that it runs efficiently on GPU, which allows it to take advantage of the parallelization speedup. Comparing the optimized version of cKDtree on CPU to TNN on GPU, one can see that TNN seems to offer the best running time. Furthermore, a GPU implementation is particularly well suited for Machine Learning applications where most of the data is processed on the GPU, so it should benefit additionally from the spared communication time.

1.4.4 Conclusion

High altitude wind nowcasting differs from weather forecasting. In the first few hours, extrapolation of high-quality measurements is still a very efficient approach because of the persistence of weather phenomena and because weather forecasts use numerical grids whose resolution is too large. Working on unstructured data measured along the trajectories of airplanes offers another challenge, as creating contexts for a prediction is not an easy task: Restricting the context to a small set of neighbors is mandatory to reduce the different models' costs in time and space. This alone reduces by almost two orders of magnitude the duration of the models' training. Nevertheless, finding a good context is not straightforward because some data should be masked to ensure time consistency, and because the retrieval depends on the scaling of the different dimensions. Traditional methods are not well suited to work with masked data and encompass many non-tensorial operations, making it difficult to adapt to GPUs. In comparison, TNN splits the data set following the recording order of the data and runs efficiently on GPUs.

We proposed a novel algorithm for searching the k-Nearest Neighbors (k-NN) for the specific case of points organized along piece-wise linear trajectories in an Euclidean space, which allows masking points along a given dimension. The general required property is that the data sets admit a coverage with cylinders. This algorithm is formulated with parallelizable tensorial operations and works well on GPUs. We provide a Pytorch (Paszke et al., 2019) implementation making its integration easy for most machine learning pipelines. It allowed us to reach a substantial increase in efficiency in the case of this study.

In this chapter, we presented an MLP-modulated averaging baseline. It also shows that even for this simple model, the nature of the data can lead to substantial complications in the training process. In Chapter 3, we will see how attention mechanisms can alleviate these problems and make the approach simpler, more robust, and more efficient.

Chapter 2

HyperMixer: An MLP-based Low Cost Alternative to Transformers

This chapter is a post-print version of

> F. Mai, A. Pannatier, F. Fehr, et al. HyperMixer: An MLP-based Low Cost Alternative to Transformers. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15632–15654, Toronto, Canada, jul 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.871. (pdf). (url). (code)

2.1 Author Contributions

This chapter is based on (Mai et al., 2023a). The work began as a group project for the course "Deep Learning for NLP," taught by James Henderson at EPFL, with the goal of reproducing the results of the MLPMixer paper (Tolstikhin et al., 2021) applied to NLP tasks. Arnaud Pannatier initiated the project, made significant contributions to the model's implementation and benchmarking suite, adapted and executed the pseudo-attention experiments (Section 2.5.7 and appendix B.2.4), authored the comparison with Lambda Layers and Linear Transformers (Appendix B.5) and contributed to the writing and illustrations. Florian Mai developed the core concept of using HyperNetworks to handle variable-length inputs, conducted the experiments, and authored the majority of the paper. Fabio Fehr also contributed to the writing and illustrations.

2.2 Introduction

In this chapter, we focus on arguably the most important flaw of attention-based models: their computational cost. Attention-based architectures, such as the Transformer (Vaswani et al., 2017), have accelerated the progress in many natural language understanding tasks. Part of their success is a result of a parallelizable training scheme over the input length. This improves training times and allows for larger volumes of data which makes these models amenable to pretraining (Radford et al., 2018; Devlin et al., 2019). This is now the default scheme and many current state-of-the-art models are fine-tuned extensions of large pre-trained Transformers (Bommasani et al., 2021).

However, these models come at a significant computational cost. They require considerable resources for pretraining and fine-tuning, which induces high energy consumption (Strubell et al., 2019) and limits access to research (Bommasani et al., 2021). Subsequently, Schwartz et al. (2020) argue the need for "*Green AI*". They propose a cost evaluation of a result R:

$$Cost(R) \propto E \cdot D \cdot H,$$

where E is the computational cost measured in floating point operations (FPO) of a single example, D is the data set size, and H is the number of hyperparameter configurations required during tuning.

To achieve a cost reduction, this chapter proposes a simpler alternative to Transformers. We take inspiration from the computer vision community, which has recently seen a surge of research on Multi-Layer Perceptrons (MLPs). Most prominently, MLPMixer (Tolstikhin et al., 2021), which is a simple architecture based on two MLPs: one for token mixing and one for feature mixing. However, the token mixing MLP learns a *fixed-size* set of *position-specific* mappings, arguably making MLPMixer's architecture too detached from the inductive biases needed for natural language understanding, in contrast to Transformers (Henderson, 2020).

In this chapter, we propose a simple variant, *HyperMixer* (Figure 2.1), which creates a token mixing MLP dynamically using hypernetworks (Ha et al., 2017). This variant is more appropriate, as it learns to generate a *variable-size* set of mappings in a *position-invariant* way, similar to the attention mechanism in Transformers (Vaswani et al., 2017). In contrast to Transformer's quadratic complexity, HyperMixer's complexity is linear in the input length. This makes it a competitive alternative for training on longer inputs.

Empirically, we demonstrate that HyperMixer works substantially better on natural language understanding tasks than the original MLPMixer and related alternatives. In comparison to Transformers, HyperMixer achieves competitive or improved results at a substantially lower cost $Cost(R) \propto E \cdot D \cdot H$: improved inference speeds (E), especially for long inputs; favorable performance in the low-resource regime (D); and efficient tuning

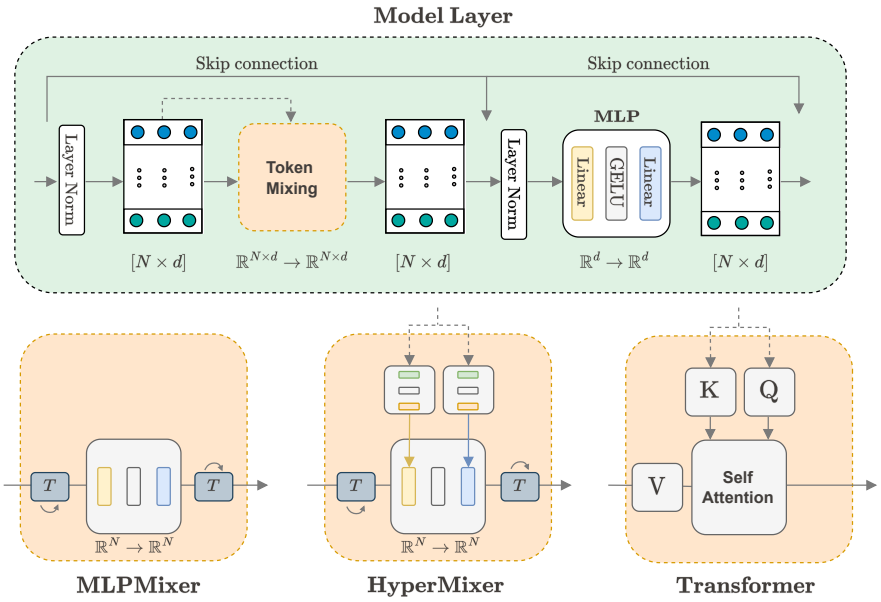


Figure 2.1: The figure outlines a general model layer consisting of a token mixing component and a feature mixing component (MLP). For token mixing, MLPMixer uses an MLP with a *fixed* size, maximum input length N, and *position-specific* weights. In contrast, HyperMixer generates an appropriately sized MLP based on the *variable* size of the input in a *position-invariant* way, similar to the attention mechanism. When using attention as token mixing the whole layer is equivalent to a Transformer encoder layer.

	Transformer	MLPMixer	HyperMixer
Complexity	$O(N^2)$	$O(N)$	$O(N)$
Pos. invariance	✓	✗	✓
Variable-length	✓	✗	✓

Table 2.1: Properties of models under consideration. N denotes the length of the input sequence.

for hyperparameters (H). We attribute HyperMixer's success to its ability to approximate an attention-like function. Further experiments on a synthetic task demonstrate that HyperMixer indeed learns to attend to tokens in a similar pattern to the attention mechanism. In summary, our contributions can be enumerated as follows:

1. A novel all-MLP model, HyperMixer, with inductive biases similar to Transformers. (Section 2.3)
2. A performance analysis of HyperMixer against alternative token mixing methods based on controlled experiments on the GLUE benchmark. (Section 2.5.3)
3. A comprehensive comparison of the cost $Cost(R)$ of HyperMixer and Transformers. (Sections 2.5.4 to 2.5.6)
4. An ablation demonstrating that HyperMixer learns attention patterns similar to Transformers. (Section 2.5.7)

2.3 Method

2.3.1 Inductive Biases in NLP Models

In machine learning, the inductive biases of a model reflect implicit modeling assumptions which are key to facilitating learning and improving generalization on specific tasks. In NLP, well-known models with strong inductive biases include recurrent neural networks (Elman, 1990), which assume the input to be a sequence; and recursive neural networks (Socher et al., 2013), which assume a tree structure. While both these inductive biases are reasonable, empirically, Transformers have been more successful in recent years. Furthermore, we reiterate the arguments of Henderson (2020) for inductive biases in language and apply them to our model design. Henderson (2020) attributes the Transformer's success to two concepts: *variable binding* and *systematicity*. Variable binding refers to the model's ability to represent multiple entities at once. This is arguably challenging in single-vector representations such as recurrent neural networks. However, Transformers represent each token with its own vector which accounts for variable binding as each

token can be interpreted as an entity. Systematicity refers to the model's ability to learn generalizable rules that reflect the structural relationship between entities (Fodor and Pylyshyn, 1988). Transformers achieve systematicity through the attention mechanism which is a learnable set of functions that determines the interaction between entities by matching query representations to key representations (as shown in Figure 2.1). The mechanism *modulates*, for every position in the sequence, how to functionally process any other position. Moreover, these function parameters are learnable and shared across all entities.

2.3.2 MLPMixer

A general layer of MLPMixer is shown in Figure 2.1. Similarly to Transformers, each token is represented as a vector of features, which undergo (non-linear) transformations in multiple layers. MLPMixer employs two MLPs at each layer, one for *feature mixing* and one for *token mixing*. The feature mixing component is applied to each token vector independently, which models the interactions between features. The Token Mixing MLP (TM-MLP) is applied to each feature independently (i.e. its vector of values across tokens), which models the interactions between spatial locations or positions. This could be interpreted as a global attention mechanism that is static and position-modulated. Practically, this is achieved by transposing the dimension representing the features and the dimension representing the positions. Each vector $\mathbf{x}_i^T \in \mathbb{R}^N$, representing feature $i \leq d$, of some input of fixed length N, is input into TM-MLP, which has the following form:

$$\text{TM-MLP}(\mathbf{x}_i^T) = \mathbf{W}_1(\sigma(\mathbf{W}_2^T\mathbf{x}_i^T)), \tag{2.1}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{N\times d'}$, and σ represents the GELU non-linearity (Hendrycks and Gimpel, 2016). Finally, to facilitate learning, layer normalization (Ba et al., 2016) and skip connections (He et al., 2016) are added around each MLP, respectively. How to best arrange these components is still an open question (Wang et al., 2019; Bachlechner et al., 2021). We experiment with different variants in Appendix B.6.

Considerations for NLP The token mixing MLP assumes an input of fixed dimension, which is necessary as the parameters need to be shared across all examples. However, unlike images, textual input is generally of a variable dimension. Therefore, to apply MLPMixer to texts of variable length, a simplistic approach is to assume a maximum length (e.g. the maximum in the data set). Thereafter, all inputs are padded to the maximum length, and masks are applied in the token mixing MLP. This model can do variable binding since each token has its own vector representation. However, this model lacks systematicity because the rules learned to model interactions between tokens (i.e. the MLP's weights) are not shared across positions.

2.3.3 HyperMixer

HyperMixer includes systematicity into the MLPMixer architecture by introducing a novel token mixing mechanism, *HyperMixing*[1], which can be regarded as a drop-in replacement for attention. For ease of understanding, we provide pseudo-code in Listing 2.1. While the queries, keys, and values in HyperMixing need not be the same, we will assume they are identical in the following formulation. HyperMixing relies on the use of hypernetworks, which are used to generate the weights $\mathbf{W}_1, \mathbf{W}_2$ of TM-MLP (Equation (2.1)) dynamically as a function of the input. Let $\mathbf{x}_j \in \mathbb{R}^d$, $j \leq N$, where N is the (variable) dimension of the input, represent token j (i.e., query, key, and value). $\mathbf{W}_1$ and $\mathbf{W}_2$ are generated by parameterized functions $h_1, h_2 : \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^{N \times d'}$. Theoretically, h_1 and h_2 could be any function, including sophisticated networks that consider non-linear interactions between tokens, such as the attention mechanism. However, this would defeat the purpose of our model, which is simplicity. Therefore, we choose to generate the rows of the weight matrices from each token independently via another MLP. Concretely, a HyperNetwork function can be defined as

$$h_i(\mathbf{x}) = \begin{pmatrix} \text{MLP}^{\mathbf{W}_i}(\mathbf{x}_1 + \mathbf{p}_1) \\ \vdots \\ \text{MLP}^{\mathbf{W}_i}(\mathbf{x}_N + \mathbf{p}_N) \end{pmatrix} \in \mathbb{R}^{N \times d'},$$

where $\text{MLP}^{\mathbf{W}_1}, \text{MLP}^{\mathbf{W}_2} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ are themselves multi-layer perceptrons with GELU non-linearity. $\mathbf{p}_j \in \mathbb{R}^d$ is a vector that can encode additional information such as the position via absolute position embeddings (Vaswani et al., 2017).

Intuitively, for each token, $\mathbf{x}_j$, h_1 decides which information to send to the hidden layer of TM-MLP, where the information from all tokens are mixed, and h_2 decides for each token how to extract information from the hidden layer. Note that, even though h_1 and h_2 only consider one token at once, non-linear interactions between tokens are still modeled through the hidden layer of TM-MLP.

Finally, layer normalization (Ba et al., 2016) can be applied to the output of TM-MLP. We found this helpful in facilitating training with a wide variety of Transformer layouts (Appendix B.6).

Tying h_1 and h_2 To reduce the number of parameters and operations in the model, and thereby the complexity, we found it useful to tie h_1 and h_2 by setting $\mathbf{W}_2 = \mathbf{W}_1$.

Considerations for NLP In comparison to the MLPMixer defined in Section 2.3.2, the use of hypernetworks overcomes two challenges. Firstly, the input no longer has to be of

[1] HyperMixing is to HyperMixer what self-attention is to Transformer encoders.

```
class HyperMixing(nn.Module):
    def __init__(self, d, d'):

        # learnable parameters
        self.hypernetwork_in = MLP([d, d, d'])
        self.hypernetwork_out = MLP([d, d, d'])

        # layer normalization improves training stability
        self.layer_norm = LayerNorm(d)

    def forward(self, queries, keys, values):
        # queries: [B, M, d]
        # keys/values: [B, N, d]

        # Add token information (e.g. position embeddings)
        hyp_in = add_token_information(keys)
        hyp_out = add_token_information(queries)

        W1 = self.hypernetwork_in(hyp_in) # [B, N, d']
        W2 = self.hypernetwork_out(hyp_out) # [B, M, d']

        # TM-MLP(x) = W_2 ( GELU ( W_1^T x) )
        # maps [B, d, N] -> [B, d, M]
        token_mixing_mlp = compose_TM_MLP(W1, W2)

        # transpose so MLP is applied to sequence dimension
        values = values.transpose(1, 2) # [B, d, N]

        output = token_mixing_mlp(values) # [B, d, M]

        # transpose back
        output = output.transpose(1,2) # [B, M, d]

        # optionally apply LayerNorm
        return self.layer_norm(output)
```

Listing 2.1: HyperMixer pseudo-code

fixed dimensionality. The hypernetwork generates a token mixing MLP of appropriate dimension as a function of the input. Secondly, the hypernetwork models the interaction between tokens with shared weights across all positions in the input. Hence, systematicity is ensured.

2.4 Related Work

Research on all-MLP models like MLPMixer (Tolstikhin et al., 2021) is widespread in the computer vision community (Tu et al., 2022; Yu et al., 2022a; Wang et al., 2022, among many others). However, they lack some desirable inductive biases for NLP, which we discuss in length in Appendix B.1.2. Specifically, in contrast to HyperMixer, none of the previously proposed methods simultaneously provide **i**) *position invariance*, which is important for generalization, **ii**) *adaptive size* for variable-length inputs, **iii**) a *global receptive field*, which allows interactions to not be limited to small token neighborhoods, **iv**) *learnabilty* allowing for universal applicablility to various tasks, and **v**) *dynamicity*, which means that token mixing is a function of the input. Consequently, only a few works have used MLP-based models as their backbone in NLP tasks. gMLP (Liu et al., 2021) serves as one of our baselines and pnlp-mixer (Fusco et al., 2023) employs standard MLPMixer on top of a novel token embedding method.

Apart from all-MLP models, there is an abundance of research on efficient alternatives to standard attention layers (Katharopoulos et al., 2020; Bello, 2021, et cetera). While they don't qualify as all-MLP models, they have close connections to our work (see Appendix B.5) and aim at lowering the cost of AI, albeit it on fewer dimensions than our work (Appendix B.1.1). We employ FNet (Lee-Thorp et al., 2022) and Linear Transformers (Katharopoulos et al., 2020) as representatives of these as a baseline.

2.5 Experiments

Our experiments are designed to test the following three hypotheses. **H1** (Section 2.5.3): Since HyperMixer reflects more inductive biases that are adequate for NLP, we hypothesize that HyperMixer performs better at NLP tasks than MLPMixer and similar MLP-based alternatives, specifically at those tasks that require to model the interactions between tokens. **H2**: Since HyperMixer has similar inductive biases as transformers but is considerably simpler conceptually and in terms of computational complexity, it can be seen as a low cost alternative to Transformers, reducing the cost in terms of single example processing time (Section 2.5.4), required data set size (Section 2.5.5), and hyperparameter tuning (Section 2.5.6). **H3** (Section 2.5.7): Due to its inductive biases mirroring those of Transformers, HyperMixer also learns similar patterns as the attention mechanism.

2.5.1 Data sets

We evaluate on four sentence-pair classification tasks and one single-sentence classification task. The sentence-pair tasks are QQP (Iyer et al., 2017), QNLI (Rajpurkar et al., 2016), MNLI (Williams et al., 2018) and SNLI (Bowman et al., 2015). For uniformity, data sets are formatted as in the GLUE benchmark (Wang et al., 2018). We choose these tasks for two properties: firstly, they have large training data sets (Table 2.2) enabling reasonable performances without pretraining; secondly, solving these tasks requires good modeling of the interactions between tokens from different sentences, which is the main focus of this chapter. As a control, we experiment on the single-input data set SST2 (Socher et al., 2013), which is a sentiment classification task. Many examples in this data set can be solved by identifying key sentiment words, rather than modeling the token interaction.

	Training Samples	Validation Samples	Test Samples
MNLI	392.702	9.815	9.796
SNLI	549.367	9.842	9.824
QQP	363.846	40.430	390.965
QNLI	104.743	5.463	5.463
SST	67.349	872	1.821

Table 2.2: Number of examples in each data set.

2.5.2 Baselines

The following baselines can be categorized into *MLP-based* (to support **H1**) and *not MLP-based* (e.g., Transformers, to support **H2**). Note that our study is about the design of the *token mixing* module. Therefore, we only compare to models that fit into the general framework displayed in Figure 2.1, where there is a feature mixing module and a token mixing module for textual inputs. As a result, models such as RNNs are excluded. To enable a controlled experiment, we use the same feature mixing module in all models; the models only differ in their token mixing module.

MLP-based The conceptually closest baseline is **`MLPMixer`** (Tolstikhin et al., 2021), which combines both token and feature mixing using fixed dimensional MLPs, as described in Section 2.3.2. Concurrently, (Liu et al., 2021) proposed **`gMLP`**, in which token mixing is achieved through weighted summation of all other inputs, similar to the attention mechanism. However, rather than computing weights as function of the inputs like in attention, in gMLP the weights are fixed learnable parameters. Additionally, linear gating initialized close to one is introduced to facilitate training. The original gMLP method does not employ feature mixing modules, as their token mixing module is capable of modeling

feature interactions as well in a single gMLP block. However, for comparability, we inject gMLP blocks as token mixing modules in our general architecture and keep feature mixing modules as well.

Non MLP-based **`Transformers`** (Vaswani et al., 2017) are used in the current state of the art in virtually all NLP tasks. Their key component is the *softmax*-based self-attention module, which we use for token mixing. **`Linear Transformer`** (Katharopoulos et al., 2020) replaces softmax attention with a *feature-map based dot-product attention*. Finally, **`FNet`** (Yu et al., 2022b) replaces the self-attention part of Transformers with a fixed, non-learnable set of Fourier transforms for token mixing.

2.5.3 Performance

Initially, we compare the performance of HyperMixer in comparison to our baselines. Thereafter, we further explore the model's benefits relative to its cost.

For comparability, we adjust the size of the token mixing components such that all models have the same number of parameters (11M). FNet is an exception since it has no learnable parameters in its token mixing component. We tune the learning rate of each model via grid-search and report the performance of the best configuration. Further experimental details on all experiments can be found in Appendix B.2.

Results Validation and test set results are shown in Table 2.3. On the test and the validation set, HyperMixer performs the best among MLP-based models on all data sets, although for SST the difference on the validation set is smaller than one standard deviation. MLPMixer generally achieves good performances, outperforming Transformers on two data sets.

Compared to non-MLP-based methods, HyperMixer also outperforms vanilla Transformers on all data sets. The differences are generally small ($\leq$ 2 points), except on QNLI, where the difference is 3.9 points. We suspect that this discrepancy is due to the relatively small training set of QNLI. We investigate the low-resource behavior of Transformers in comparison to HyperMixer in Section 2.5.5. FNet performs substantially worse than the other methods, particularly on SNLI and QQP. Linear Transformers achieve excellent performance on MNLI and SNLI but perform poorly on QNLI and QQP.

In Appendix B.3.2, we discuss ablations such as untied HyperMixer.

	MNLI	SNLI	QQP	QNLI	SST	Params
Validation set results (average accuracy / standard deviation over 10 seeds)						
FNet	59.7 (0.27)	75.3 (0.46)	79.4 (0.28)	59.9 (0.46)	79.7 (0.71)	9.5 M
Lin. Transformer	**66.9** (0.48)	**82.7** (0.22)	81.7 (0.28)	61.3 (0.29)	80.5 (0.46)	11 M
Transformer	65.4 (0.51)	80.9 (0.40)	82.8 (0.22)	67.3 (2.03)	79.0 (0.86)	11 M
MLPMixer	63.9 (0.34)	79.6 (0.11)	83.7 (0.42)	68.1 (2.10)	80.1 (0.67)	11 M
gMLP	60.8 (0.95)	80.5 (0.55)	82.8 (0.21)	60.5 (0.49)	78.7 (0.74)	11 M
HyperMixer (ours)	66.2 (0.21)	81.9 (0.27)	**85.6** (0.20)	**78.0** (0.19)	80.7 (0.84)	11 M
Test set results (best model)						
FNet	59.8	75.3	78.4	59.6	80.0	9.5 M
Lin. Transformer	66.9	83.0	82.3	61.7	80.8	11 M
Transformer	65.8	80.7	82.4	73.2	79.4	11 M
MLPMixer	62.9	80.1	83.5	70.5	81.2	11 M
gMLP	61.2	80.9	82.5	60.2	79.5	11 M
HyperMixer (ours)	66.1	81.7	84.1	77.1	81.4	11 M

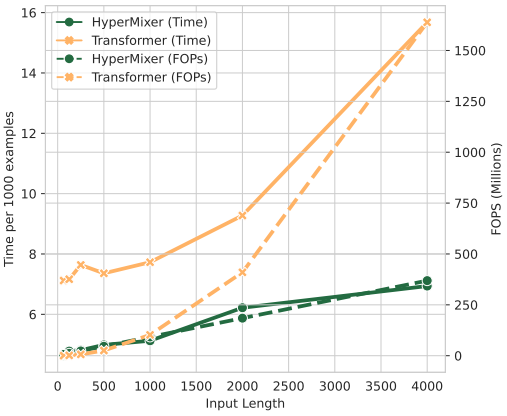
Table 2.3: *Top*: Mean validation set accuracy and standard deviation over 10 different seeds of the best hyperparameter configuration. Results are printed in **bold** font if they exceed the second-best result by at least one standard deviation. Underline marks the best MLP-based model. *Bottom*: Test set results on natural language understanding tasks when using the best model on the validation set. We evaluate on a single seed due to the limited test set access of GLUE.

2.5.4 Time per Example

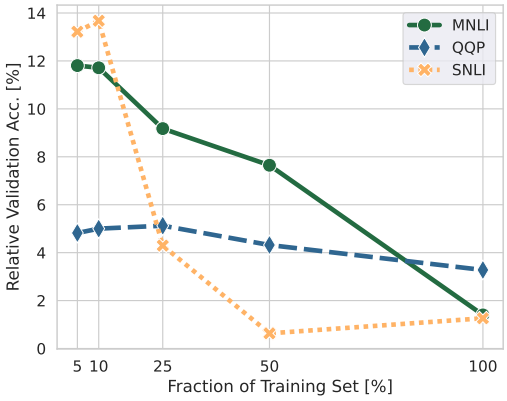
To assess the efficiency of our model, we measure the wallclock-time of processing a single input (repeated 1,000 times) through the token mixing stages of HyperMixer and Transformer, respectively. As Schwartz et al. (2020) point out, wallclock time has the downside of being dependent on the specific implementation, and they recommend reporting the number of floating point operations (FOPs) required by one forward pass. In Figure 2.2a, we show wallclock time and theoretical FOPs as a function of the input length N. For short input sequences, the number of FOPs is dominated by the size of the hidden layer and hence slightly lower for Transformers than for HyperMixer. However, in practical terms, we observe that HyperMixer is still faster than Transformers. At longer input sequences, the size of N starts to dominate the total complexity of Transformers, so that it becomes exceedingly slower than HyperMixer.

2.5.5 Low Resource Performance

Like MLPMixer, HyperMixer is a conceptually simple architecture, as it only applies multi-layer perceptrons at its core. Simpler architectures often make for better performance on smaller-scale data sets. We investigate this by varying the number of examples used for training on the three large data sets MNLI, SNLI, and QQP. For these experiments, we use the best-performing learning rate found in the grid search from Section 2.5.3. In Figure 2.2b, we plot the relative performance change of HyperMixer compared to



(a) WCT / FOPs of propagating a single example through the token mixing of HyperMixer vs. Transformer depending on the input length.



(b) Relative improvement of HyperMixer over Transformer depending on what percentage of the training set is used.

Transformers as a function of subsample size. On all data sets, the relative improvement of HyperMixer over Transformers is larger when training with 10% of the data set than with the full data set. While the effect is small on QQP, it is particularly large on SNLI and MNLI, where HyperMixer performs almost 12-14% better with 10% of the data, while the relative improvement with the full data set is less than 2%.

2.5.6 Ease of Hyperparameter Tuning

MLP-based token mixing has the advantage that it is conceptually simpler than self-attention and that it is well-known how to facilitate training via mechanisms such as skip connections and layer normalization. Both these aspects suggest that it might be easier to find hyperparameter configurations that yield good performances. In these experiments, we compare HyperMixer (with tied hypernetworks) to Transformers in this regard. As recommended in Schwartz et al. (2020), we perform a random search to tune hyperparameters and compute the expected validation performance (Dodge et al., 2019, 2021). Specifically, we tune the learning rate, whose logarithm is drawn from $\mathcal{U}(-8,-1)$, and the dropout probability drawn from $\mathcal{U}(0,0.5)$ for 20 trials.

Results In Figure 2.3, we show the *relative* expected validation performance, i.e., the relative performance change of HyperMixer compared to Transformer, for all five data sets. With the notable exception of QNLI, the relative improvement of HyperMixer is higher at smaller budgets than at larger budgets on all data sets. The effect is particularly strong on SNLI, where HyperMixer is 6.5% better at small tuning budgets, but less than 2% better at high budgets. These results indicate that HyperMixer is substantially easier to tune than Transformers.

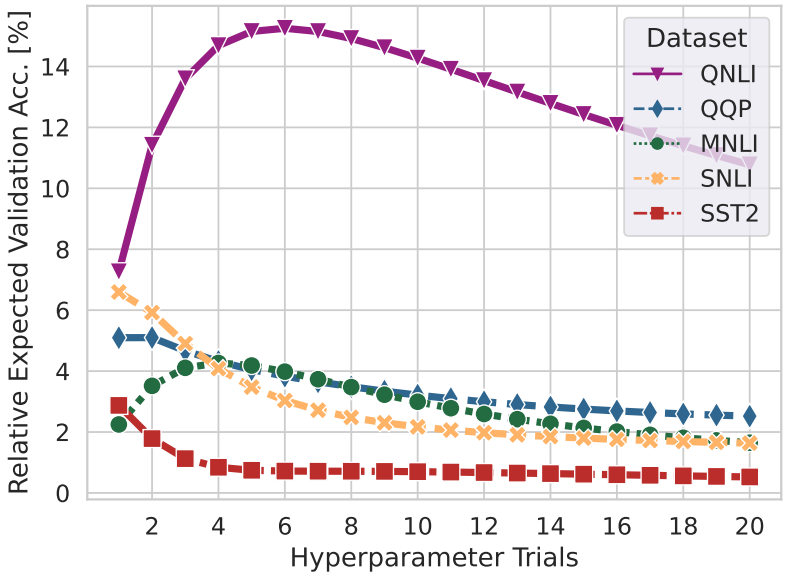


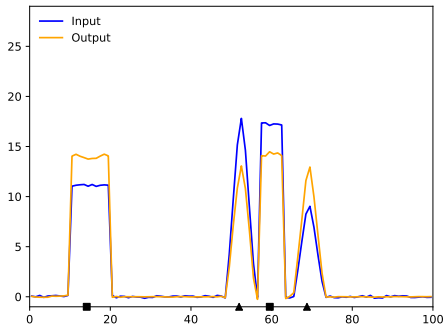
Figure 2.3: Relative expected validation performance of HyperMixer compared to Transformer after tuning the learning rate and dropout via random search.

2.5.7 HyperMixer Learns Attention Patterns

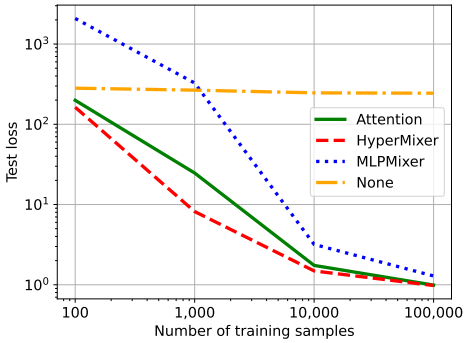
We hypothesized that the token mixing layer of HyperMixer offers a mechanism similar to attention. To show this, we consider a toy problem with 1d sequences composed of shape pairs of different heights as described in Fleuret (2019). The target value is the average height in each pair of shapes. An example input is shown in Figure 2.4a. To solve the task well, for each position, the model must attend to other positions with the same shape.

Models We compare the token mixing layer of HyperMixer to three other models: i) *None* does not model token interactions. All predictions are thus only made based on local information. This model should thus fail. ii) *MLPMixer* does model token interactions. Still, since its token mixing weights are position-specific, each position has to learn to recognize each shape, which we expect to be difficult, especially with little data. iii) *Self-attention* can be considered the upper bound, as it models the interaction between every two positions explicitly.

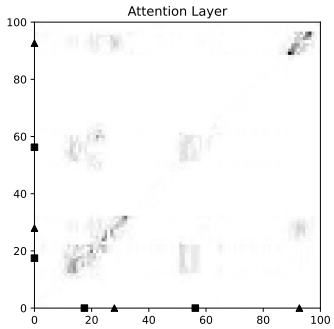
Results Figure 2.4b shows the mean squared error on the test examples depending on the number of training examples. As expected, *None* fails on this task. While all other models can solve the task with enough training data, MLPMixer is considerably less data-efficient than the other two models, requiring 5-10 times more data to reach the same performance. This is expected, since in contrast to HyperMixer and self-attention, MLPMixer's token mixing module is not position-invariant. HyperMixer and self-attention



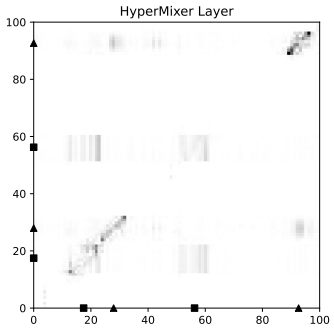
(a) Example from the synthetic task. The y-axis represents heights to be averaged.



(b) Test loss depending on number of examples.



(c) Pseudo-attention map of Attention



(d) Pseudo-attention map of HyperMixer

Figure 2.4: Results and pseudo-attention maps on the synthetic task, which consists of a sequence of 100 tokens with two similar patters whose heights in arbitrary units needs to be averaged. (Fleuret, 2019).

reach approximately the same performance when training on 100k examples. However, HyperMixer is more data-efficient than self-attention, which we attribute to the simpler model architecture.

We can measure the interactions between two tokens by computing the gradient of an output token relative to an input token (pseudo-attention). Figures 2.4c and 2.4d show the pseudo-attention maps of HyperMixer in comparison to attention. We observe that the pseudo-attention weights of HyperMixer and attention are similar. This indicates that HyperMixer indeed learns an attention-like function. In contrast, we find these patterns to be weaker in MLPMixer (Figure B.1, appendix).

2.6 Discussion

In the following, we first discuss the merits of our proposed model, which are the core contributions of our chapter. We then discuss the scope of our analysis.

2.6.1 Impact

Best all-MLP model HyperMixer was designed as an MLP-based architecture with similar inductive biases as Transformers, which are beneficial for natural language understanding. Our hypothesis (**H1**) is that this leads to improvements over other MLP-based methods. Our experimental results support this hypothesis, as we find HyperMixer to outperform all MLP-based baselines on all data sets (Section 2.5.3).

Low cost model The main motivation for an MLP-based architecture is the efficiency benefits induced by its simplicity. Therefore, we hypothesized (**H2**) that HyperMixer would reduce the cost $Cost(R) \propto E \cdot D \cdot H$ to obtain an AI result R. This hypothesis is supported by our experiments. While HyperMixer yields results that are on par with Transformer's results, it reduces the cost of all three cost factors: i) The cost of processing a single example (E) is lower, particularly for long inputs due to its linear complexity compared to the quadratic complexity of self-attention (Section 2.5.4). ii) The number of required training examples (D) is reduced, as HyperMixer's relative performance improvement is larger in the low-resource scenario (Section 2.5.5). iii) HyperMixer requires less hyperparameter tuning than Transformers to reach good results, which is demonstrated by HyperMixer's higher expected relative improvements at low tuning budgets (Section 2.5.6).

Attention-like model Finally, our experiments on a synthetic task indicate that HyperMixer can learn very similar attention patterns as the self-attention mechanism in Transformers (Section 2.5.7), supporting hypothesis **H3**. While MLPMixer can also learn similar patterns given enough training data, we believe that it is the introduction of adequate biases that allows HyperMixer to learn these patterns efficiently. These biases were chosen based on an analysis of Transformer's success by Henderson (2020). HyperMixer's own success hence supports that analysis.

In summary, in our study, HyperMixer is the best-performing MLP-based architecture and shows comparable performance and behavior as self-attention at substantially lower cost. HyperMixer can thus be considered a low cost alternative to Transformers.

2.6.2 Scope

Small resource scenario It is important to note that our study is limited to the small resource scenario: Our models are small, not pre-trained on large general-purpose corpora, and trained on data sets with fewer than 1 million examples. It is unclear if our results will also hold on a larger scale. For example, while gMLP and FNet perform poorly in the low-resource scenario as demonstrated in our experiments, both models are able to

narrow the gap to Transformer-based models as the resources for pretraining increase (Liu et al., 2021; Lee-Thorp et al., 2022). We hypothesize that with enough resources, these models can overcome their shortcomings in terms of inductive biases. However, there is no reason to believe that HyperMixer, being equipped with useful inductive biases, wouldn't perform on par with Transformers in high-resource scenarios while retaining its lower overall cost. Quite the contrary, HyperMixer's linear complexity in sequence length perhaps makes it more appropriate for large-scale pretraining on long contexts than vanilla Transformers.

Versatility One of the most impressive qualities of Transformers is their versatility: Not only are they now the standard architecture for all NLP tasks, but over the years they have also become ubiquitous in a wide range of applications domains outside of NLP. Of course, the present study cannot determine whether HyperMixer is as versatile as Transformers. However, subsequent studies have shown that HyperMixer has uses in speech recognition (Mai et al., 2023b) and neural combinatorial optimization (Drakulic et al., 2023). Still, some modeling advancements are needed. For example, HyperMixing is not yet applicable for decoder models that make use of causal masking. As decoder-only language models have become widely studied, this constitutes promising future work.

2.7 Conclusion

In this chapter, we have proposed HyperMixer, an MLP-based model that is subquadratic in sequence length and has the same inductive biases as Transformers.

While large pre-trained Transformer language models have led to impressive progress, they require so many resources that many research labs are excluded from participation, leading to calls for low cost alternatives. We have proposed an MLP-based method, HyperMixer, that, in contrast to previous MLP-based methods, is equipped with the same inductive biases that made Transformers so successful for natural language understanding. While it performs on par with Transformers, it incurs substantially lower costs in terms of processing time, training data, and hyperparameter tuning.

Hypermixer induces a significant cost reduction in the case where the sequence length is the main bottleneck. For wind nowcasting in a setting where the sequence length is not a problem, transformers have the advantage of letting the model see contributions from all tokens independently. In the next chapter, we show how attention-based models can leverage this property to improve the performance of wind forecasting models.

Part II

An Attention-Based Solution for Wind Nowcasting

Chapter 3

Inference from Real-World Sparse Measurements

This chapter is a post-print version of:

> A. Pannatier, K. Matoba, and F. Fleuret. Inference from Real-World Sparse Measurements. *Transactions on Machine Learning Research*, 2024b. ISSN 2835-8856. (pdf). (url). (code)

3.1 Introduction

This chapter presents our solution to the wind nowcasting problem using attention models. It has several advantages compared to our baseline presented in Chapter 1. Our averaging baseline lacked expressivity and needed the creation of a tailored context per measurement

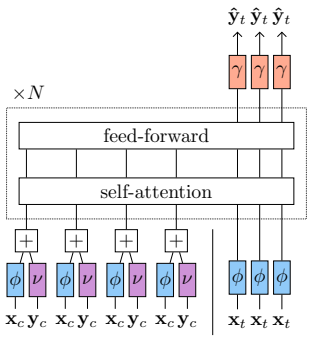


Figure 3.1: Multi-layer Self-Attention

Architecture	Performance	Simplicity	Domain Knowledge Agnostic
CNP	✗	✔✔	✔✔
GEN	✔	✗	✗
TFS	✔	✔	✔✔
MSA (Ours)	✔✔	✔✔	✔✔

Table 3.1: Comparison between our approach and the different baselines. Multi-Layer Self-Attention (MSA) achieves good performance while being simple to implement and does not require practitioner knowledge for proper setup.

to perform well. As it is an averaging method, its output is also constrained to values comprised in the convex hull of the context set. The method presented in this chapter is more powerful and allows the use of a simpler training pipeline. We can simply split the data set into time slices separated by the desired window of prediction. As the model is expressive enough, it is capable of learning in that setup, which alleviates the need for a per-measurement approach. In our setup, we are not limited by the number of measurements that are given to the model, so we do not need the efficient approach devised in Chapter 2. However, if it should be adapted to a more large-scale problem, the method presented in Chapter 2 could be used to reduce the memory footprint of the model.

This approach follows the growing trend of using deep learning methods to exploit the nearly unlimited amount of data available for physical models, in ways that traditional solvers cannot. State-of-the-art weather models are now using deep learning methods as their main forecasting tool, outperforming traditional solvers in the case of precipitation nowcasting (Suman et al., 2021b; Shi et al., 2017) and reaching similar performance in medium-term forecasting (Lam et al., 2022). However, these models are limited to data represented as images or on regular grids, where models such as convolutional networks or graph neural networks are used. In contrast, various real-world data often come from irregularly placed or moving sensors, which means custom architectures are needed to handle it effectively.

An example that can benefit significantly from such an architecture is Air Traffic Control (ATC). ATC needs reliable weather forecasts to manage airspace efficiently. This is particularly true for wind conditions, as planes are highly sensitive to wind and deviations from the initial flight plan can be costly and pose safety hazards. DL models are a promising candidate for producing reliable wind forecasts as a large amount of data is collected from airplanes that broadcast wind speed measurements with a four-second frequency. A model that can effectively model wind speeds using data collected from airplanes, should be able to decode anywhere in space, as we aim to predict wind conditions at future locations of the airplane, conditioned on past measurements taken by that specific airplane or neighboring ones in a permutation invariant manner.

To meet these requirements, we introduce a Multi-layer Self-Attention model (MSA) and compare it to different baselines [Table 3.1]: Conditional Neural Processes (CNP) (Garnelo et al., 2018), Graph Element Networks (GEN) (Alet et al., 2019) and a transformer encoder-decoder baseline (TFS) that we developed. While all of these models possess the aforementioned characteristics, they each adopt distinct strategies for representing measurements within the latent space. CNP models use a single vector as a summary of encoded measures, while GEN models map the context to a graph, based on their distance to its nodes. MSA keeps one latent vector per encoded measurement and can access them directly for forecasting. This latent representation is better, as it does not create a bottleneck. We show that due to that architectural choice, both baselines can fail, in certain cases, to use information from the context they condition on.

Our approach offers better performance than its competitors and is conceptually simpler as it does not require an encoder-decoder structure. To evaluate the effectiveness of our approach, we conducted experiments on high-altitude wind nowcasting, heat diffusion, fluid dynamics, and two-day weather forecasting. Several additional ablation studies show the impact of different architectural choices.

The main contributions of this work are summarized below:

- We develop an attention-based model that can generate prediction anywhere in the space conditioned on a set of measurements.
- We propose a novel encoding scheme using a shared MLP encoder to map context and target positions, improving forecasting performance and enhancing the model's understanding of spatial patterns.
- We evaluate our method on a set of challenging tasks with data irregularly sampled in space: high-altitude wind nowcasting, two-day weather forecasting, heat diffusion, and fluid dynamics.
- We examine the differences between models, and explain the impact of design choices such as latent representation bottlenecks on the final performance of the trained models.

3.2 Related Works

DL performance for weather forecasting has improved in recent years, with DL models increasingly matching or surpassing the performance of traditional PDE-based systems. Initially applied to precipitation nowcasting based on 2D radar images (Suman et al., 2021b; Shi et al., 2017), DL-based models have recently surpassed traditional methods for longer forecast periods (Lam et al., 2022). These methods usually work with highly

structured data. Radar precipitation data, for example, can be organized as images and analyzed using convolutional neural networks. For 3D regular spherical grid data, graph neural networks or spherical CNNs are employed (Lam et al., 2022; Esteves et al., 2023). However, in our study, the data set is distributed sparsely in space, which hinders the use of these traditional architectures. The use of DL for modeling dynamical systems, in general, has also seen recent advancements (Li et al., 2021; Gupta and Brandstetter, 2023; Pfaff et al., 2020) but most approaches in this field typically operate on regularly-spaced data or irregular but fixed mesh.

Neural Processes (Garnelo et al., 2018; Kim et al., 2019), Graph Element Networks (Alet et al., 2019) and attention-based models (Vaswani et al., 2017) are three DL-based approaches that are capable of modeling sets of data changing from set to set. In this study, we conduct a comparison of these models by selecting a representative architecture from each category. Additionally, attention-based models have been previously adapted for set classification tasks (Lee et al., 2019), and here we adapt them to generate forecasts.

Pannatier et al. (2022) use a kernel-based method for wind nowcasting based on flight data. This method incorporates a distance metric with learned parameters to combine contexts for prediction at any spatial location. However, a notable limitation of this technique is that its forecasts are constrained to the convex hull of the input measurements, preventing accurate extrapolation. We evaluate the efficacy of our method compared to this approach, along with the distinct outcomes obtained, in Appendix C.7.

While previous studies have utilized transformers for modeling physical systems (Geneva and Zabaras, 2022), time series (Li et al., 2019) or trajectory prediction (Girgis et al., 2022; Nayakanti et al., 2023; Yuan et al., 2021) these applications do not fully capture the specific structure of our particular domain, which involves relating two spatial processes at arbitrary points on a shared domain. Although we model temporal relationships, our approach lacks specialized treatment of time. Therefore, it does not support inherently time-based concepts like heteroskedasticity, time-series imputation, recurrence, or seasonality. Further details distinguishing our approach from other transformer-based applications are elaborated in Appendix C.3.

3.3 Methodology

3.3.1 Context and Targets

The problem addressed in this chapter is the prediction of target values given a context and a prediction target position. Data is in the form of pairs of vectors $(\mathbf{x}_c, \mathbf{y}_c)$ and $(\mathbf{x}_t, \mathbf{y}_t)$ where $\mathbf{x}_c$ and $\mathbf{x}_t$ are the position and $\mathbf{y}_c$ and $\mathbf{y}_t$ are the measurements (or values), where we use c for context, t for target, x for spatial position and y for the corresponding vector

value. The positions lie in the same underlying space $\mathbf{x}_c, \mathbf{x}_t \in \mathbb{X} \subseteq \mathbb{R}^X$, but the context and target values do not necessarily. We define the corresponding spaces as $\mathbf{y}_c \in \mathbb{I} \subseteq \mathbb{R}^I$ and $\mathbf{y}_t \in \mathbb{O} \subseteq \mathbb{R}^O$, respectively, where X, I, O are integers that need not be equal. The data set comprises multiple pairs of context and target sets, each possibly varying in length. We denote the length of the j-th context and target set as N_c^j and N_t^j. All models take as input a set of context pairs $\{(\mathbf{x}_c, \mathbf{y}_c)_i^j\}_{i=1}^{N_c^j}$, as well as target positions, denoted $\{(\mathbf{x}_t)_i^j\}_{i=1}^{N_t^j}$.

As an example, to transform a data set of wind speed measurements into context and target pair, we partitioned the data set into one-minute time segments and generated context and target sets with an intervening delay, as depicted in Figure 3.2. The underlying space, denoted by $\mathbb{X}$, corresponds to 3D Euclidean space, with both $\mathbb{I}$ and $\mathbb{O}$ representing wind speed measurements in the x, y plane. The models are given a set of context points at positions $\mathbf{x}_c$ of value $\mathbf{y}_c$ and query positions $\mathbf{x}_t$ and are tasked to produce a corresponding value $\mathbf{y}_t$ conditioned on the context. Detailed descriptions of the data set, including illustrations of the different problems, and the respective spaces for other scenarios and the ablation study can be found in Table C.1.

3.3.2 Encoding Scheme

We propose in this section a novel encoding scheme for irregularly sampled data. Our approach leverages the fact that both the context measurements and target positions reside within a shared underlying space. To exploit this shared structure, we adopt a unified two-layer MLP encoder ϕ for mapping both the context and target position to a latent space representation. Then, we use a second MLP ν to encode the context values and add them to the encoded positions when available. This differs from the approach proposed in Garnelo et al. (2018); Alet et al. (2019) where both the context position and value are concatenated and given to an encoder, and the target position is encoded by another. The schemes are contrasted as:

Traditional methods

$$\mathbf{e}_c = \varphi(\mathbf{x}_c, \mathbf{y}_c) \tag{3.1}$$
$$\mathbf{e}_t = \psi(\mathbf{x}_t) \tag{3.2}$$

Proposed scheme

$$\mathbf{e}_c = \phi(\mathbf{x}_c) + \nu(\mathbf{y}_c) \tag{3.3}$$
$$\mathbf{e}_t = \phi(\mathbf{x}_t) \tag{3.4}$$

Where ϕ, ν, φ, ψ are two hidden-layers MLPs, and $\mathbf{e}_c, \mathbf{e}_t \in \mathbb{R}^E$ are context positions and values which are concatenated and encoded, and target position which is encoded respectively.

3.3.3 Multi-layer Self-Attention (MSA, Ours)

Our proposed model, Multi-layer Self-Attention (MSA) harnesses the advantages of attention-based models [Figure 3.1]. MSA maintains a single latent representation per input measurement and target position, which conveys the ability to propagate gradients easily and correct errors in training quickly. MSA can access and combine target position and context measurements at the same time, which forms a flexible latent representation. Our model is similar to a transformer-encoder, as the backbone of a ViT (Dosovitskiy et al., 2020), it can be written as:

$$\{(\mathbf{l}_c)_i^j\}_{i=1}^{N_c^j}, \{(\mathbf{l}_t)_i^j\}_{i=1}^{N_t^j} = \text{Transformer-Encoder}(\{(\mathbf{e}_c)_i^j\}_{i=1}^{N_c^j}, \{(\mathbf{e}_t)_i^j\}_{i=1}^{N_t^j}) \tag{3.5}$$

$$\{(\hat{\mathbf{y}}_t)_i^j\}_{i=1}^{N_t^j} = \{\gamma((\mathbf{l}_t)_i^j)\}_{i=1}^{N_t^j} \tag{3.6}$$

where γ is a two hidden-layers MLP that is used to generate readouts from the latent space. As a transformer-encoder outputs the same number of outputs as inputs, we define its output as $\{(\mathbf{l}_c)_i^j\}_{i=1}^{N_c^j}, \{(\mathbf{l}_t)_i^j\}_{i=1}^{N_t^j}$ corresponding to the latent representations of the context and target positions respectively. However, only $\{(\mathbf{l}_t)_i^j\}_{i=1}^{N_t^j}$ is used to generate the readouts.

In MSA, the whole transformer encoder model is fully preserved; in particular, we keep all the normalization layers and the attention layer with the Query-Key-Value structure. The major change is the input data. These inputs are absolute geographical or planar positions encoded with a simple MLP. We considered using sinusoidal positional encoding for these geographical positions, but it did not improve the performance of the model. We did not use relative positional encoding to ensure that the model could learn important topographical features. Training with relative positional encoding is possible and could be beneficial if we aim to have a model that is invariant to the position of the context and target points, but it is not the focus of this work.

MSA does not use positional encoding for encoding the order of the inputs. This model is permutation equivariant due to the self-attention mechanism and it uses full attention, allowing each target feature to attend to all other targets and context measurements. MSA generates all the output in one pass in a non-autoregressive way and the outputs of the model are only the units that correspond to the target positions, which are then used to compute the loss.

3.3.4 Baselines

Transformer(s) (TFS) We also modify an encoder-decoder transformer (TFS) model (Vaswani et al., 2017) to the task at hand. The motivation behind this was the intuitive appeal of

the encoder-decoder stack for this specific problem. TFS in our approach deviates from the standard transformer in a few ways: Firstly, it does not employ causal masking in the decoder and secondly, the model forgoes the use of positional encoding for the sequence positions. It can be written as:

$$\{(\mathbf{l}_c)_i^j\}_{i=1}^{N_c^j} = \text{Transformer-Encoder}(\{(\mathbf{e}_c)_i^j\}_{i=1}^{N_c^j}) \tag{3.7}$$

$$\{(\mathbf{l}_t)_i^j\}_{i=1}^{N_t^j} = \text{Transformer-Decoder}(\{(\mathbf{l}_c)_i^j\}_{i=1}^{N_c^j}, \{(\mathbf{e}_t)_i^j\}_{i=1}^{N_t^j}) \tag{3.8}$$

$$\{(\hat{\mathbf{y}}_t)_i^j\}_{i=1}^{N_t^j} = \{\gamma((\mathbf{l}_t)_i^j)\}_{i=1}^{N_t^j} \tag{3.9}$$

In comparison to MSA, TFS uses an encoder-decoder architecture, which adds a layer of complexity. Moreover, it necessitates the propagation of error through two pathways, specifically through a cross-attention mechanism that lacks a residual connection to the encoder inputs.

Graph Element Network(s) (GEN) Graph Element Networks (GEN) (Alet et al., 2019) represent an architecture that leverages a graph $\mathcal{G}$ as a latent representation. This graph comprises N nodes situated at positions $\mathbf{x}_n$ and connected by E edges. Selecting the nodes' positions and the graph's edges is critical, as these are additional parameters that require careful consideration. The node positions can be made learnable, allowing for their optimization during training through gradient descent. It's important to note that the learning rate for these learnable parameters should be meticulously chosen to ensure convergence, typically being smaller in magnitude compared to other parameters.

In the original work, edges are established using Delaunay triangulation and may be altered during training in response to shifts in node positions. The encoder function transforms measurements into a latent space. These are then aggregated to formulate the initial node values, influenced by their proximity to the nodes. Specifically, a measurement at position $\mathbf{x}_c$ impacts the initial value of a node at position $\mathbf{e}_i$ (with $i \in 1, \ldots, N$) based on a weighting function $r(\mathbf{x}_c, \mathbf{e}_i)$, which assigns greater significance to context points nearer to the graph's nodes. In the original work, this weighting function is defined as $r(\mathbf{x}_c, \mathbf{e}_i) = \text{softmax}(-\beta \|\mathbf{x}_c - \mathbf{e}_i\|)$, where β is a learnable parameter. The initial node values undergo processing through L iterations of message passing. For model readouts, the same weighting function is employed, ensuring each node is weighted according to its distance from the query point. Overall, the model can be described as follows:

$$\mathbf{e}_n = \sum_{(\mathbf{x}_c, \mathbf{e}_c) \in \mathcal{E}} r(\mathbf{x}_c, \mathbf{x}_n)\mathbf{e}_c \qquad \forall n \in \{1, \ldots, N\} \tag{3.10}$$

$$\{(\mathbf{l}_n)_i\}_{i=1}^{N} = \text{Message-Passing}(\{(\mathbf{e}_n)_i\}_{i=1}^{N}, \mathcal{G}, L) \tag{3.11}$$

$$(\mathbf{l}_t)_i^j = \sum_{(\mathbf{x}_n, \mathbf{l}_n) \in \mathcal{L}} r(\mathbf{x}_n, \mathbf{x}_t)\mathbf{l}_n \qquad \forall i \in \{1, \dots, N_t^j\} \tag{3.12}$$

$$\{(\hat{\mathbf{y}}_t)_i^j\}_{i=1}^{N_t^j} = \{\gamma((\mathbf{l}_t)_i^j)\}_{i=1}^{N_t^j} \tag{3.13}$$

Where $\mathcal{E} = \{(\mathbf{x}_c, \mathbf{e}_c)_i^j\}_{i=1}^{N_c^j}$ is the set of encoded context and their position and $\mathcal{L} = \{(\mathbf{x}_n, \mathbf{l}_n)_i\}_{i=1}^{N}$ is the set of the graph nodes and their position.

GEN's inductive bias is that a single latent vector summarizes a small part of the space. As it includes a distance-based encoding and decoding scheme, the only way for the model to learn non-local patterns is through message passing. This model was originally designed with a simple message-passing scheme. But it can easily be extended to a broad family of graph networks by using different message-passing schemes, including ones with attention. We present some related experiments in Appendix C.12.

Conditional Neural Process(es) (CNP) CNP (Garnelo et al., 2018) encodes the whole context as a single latent vector. They can be seen as a subset of GEN. Specifically, a CNP is a GEN with a graph with a single node and no message passing. While CNP possesses the desirable property of being able to model any permutation-invariant function (Zaheer et al., 2017), their expressive capability is constrained by the single node architecture (Kim et al., 2019). Despite this, CNP serves as a valuable baseline and is considerably less computationally intensive.

$$\mathbf{l}_c = \text{mean}(\{(\mathbf{e}_c)_i^j\}_{i=1}^{N_c^j}) \tag{3.14}$$

$$\{(\mathbf{l}_t)_i^j\}_{i=1}^{N_t^j} = \{\eta\left(\mathbf{l}_t, (\mathbf{e}_t)_i^j\right)\}_{i=1}^{N_t^j} \tag{3.15}$$

$$\{(\hat{\mathbf{y}}_t)_i^j\}_{i=1}^{N_t^j} = \{\gamma((\mathbf{l}_t)_i^j)\}_{i=1}^{N_t^j} \tag{3.16}$$

Where η is a simple linear projection.

3.4 Experiments

Our experiments aim to benchmark the performance of our models on various data sets with irregularly sampled data. The first task focuses on high-altitude wind nowcasting. The second task is on heat diffusion. Additionally, we evaluate our models on fluid flows, considering both a steady-state case governed by the Darcy Flow equation and a dynamic case modeling the Navier-Stokes equation in an irregularly spaced setting. Finally, we compare the models on a weather forecasting task, utilizing irregularly sampled

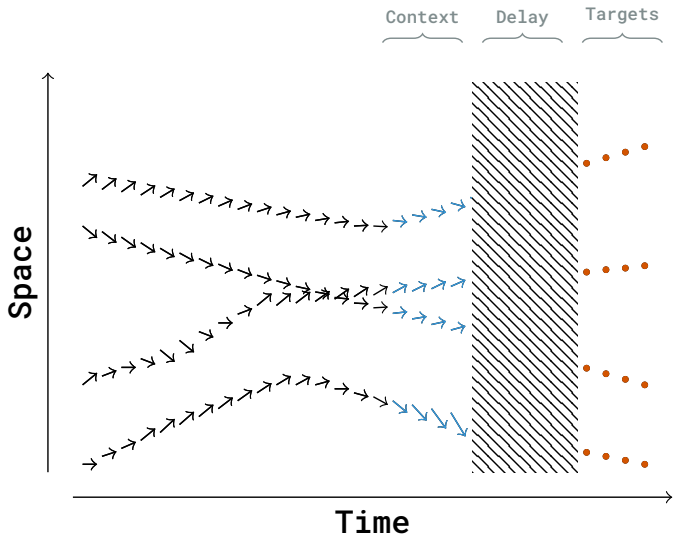


Figure 3.2: Description of the context and target sets in the wind nowcasting case. The context set and the target set are time slices separated by a delay, which corresponds to the forecasting window. The underlying space is in that case $\mathbb{X} \subseteq \mathbb{R}^3$ and the context values and target values both represent wind speed and belong to the same space $\mathbb{I} = \mathbb{O} \subseteq \mathbb{R}^2$.

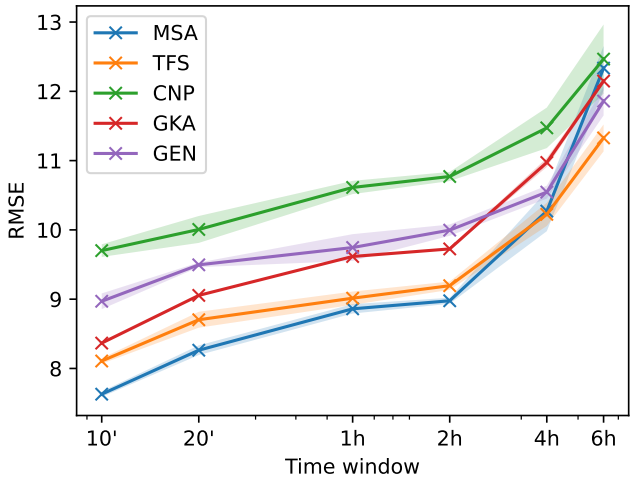


Figure 3.3: Decay of precision in the wind nowcasting case. RMSE of the different models depending on the forecast duration (lower is better). We ran three experiments varying the pseudorandom number generator seeds for each time window and each model to measure the standard deviation. The error does not increase drastically over the first two hours because of the persistence of the wind and the context values are good predictors of the targets in that regime.

Table 3.2: Validation RMSE of the High-Altitude Wind Nowcasting, Poisson, Navier-Stokes, and Darcy Flow equation and the weather forecasting task. Each model ran for 10, 2000, 1000, 100, and 100 epochs respectively on an NVIDIA GeForce GTX 1080 Ti. The low number of epochs for wind nowcasting is due to the amount of data which is considerably larger than in the other experiments. The standard deviation is computed over 3 runs. We present here the original implementation of CNP and GEN compared with TFS and MSA with sharing weights for the position. More details can be found in Table 3.2 of Appendix C. We choose the configuration of the models so that every model has a comparable number of parameters. We underline the best models for each size and indicate in bold the best model overall.

Architecture	# of Params.	Wind Nowcasting	Poisson Equation	Navier-Stokes Equation	Darcy Flow Equation	ERA5
CNP	5k	11.94± 0.78	0.33± 0.004	0.701± 0.0023	0.0311± 0.0008	2.129± 0.0039
	20k	10.19± 1.83	0.32± 0.003	0.672± 0.0011	0.0295± 0.0002	2.117± 0.0018
	100k	10.17± 1.24	0.33± 0.003	0.656± 0.0007	0.0286± 0.0001	2.110± 0.0002
GEN	5k	11.02± 3.19	0.12± 0.006	0.604± 0.0010	0.0304± 0.0003	2.132± 0.0035
	20k	9.98± 0.76	0.13± 0.014	0.599± 0.0006	0.0296± 0.0002	2.124± 0.0031
	100k	9.56± 0.21	0.16± 0.049	0.596± 0.0005	0.0294± 0.0001	2.121± 0.0005
TFS (Ours, baseline)	5k	8.30± 0.03	0.15± 0.036	0.604± 0.0022	0.0275± 0.0014	2.129± 0.0032
	20k	8.20± 0.04	0.09± 0.006	0.596± 0.0008	**0.0258± 0.0003**	2.109± 0.0012
	100k	8.38± 0.13	0.18± 0.014	0.591± 0.0012	0.0269± 0.0004	2.100± 0.0011
MSA (Ours)	5k	8.07± 0.11	0.11± 0.006	0.597± 0.0011	0.0274± 0.0011	2.125± 0.0070
	20k	**7.98± 0.03**	**0.08± 0.003**	0.589± 0.0013	0.0259± 0.0007	2.107± 0.0020
	100k	8.18± 0.14	0.10± 0.009	**0.589± 0.0006**	0.0264± 0.0004	**2.098± 0.0029**

measurements from the ERA5 data set (Hersbach et al., 2023) to predict wind conditions two days ahead.

For the Wind Nowcasting Experiment, the data set, described in Section 3.3.1, consists of wind speed measurements collected by airplanes with a sampling frequency of four seconds. We evaluate our models on this data set [Table 3.2] and we assess the models' performance as a function of forecast duration, as depicted in Figure 3.3, and against different metrics [Table 3.3] to ensure the robustness of our approach. We select model configurations with approximately 100,000 parameters and run each model using three different random seeds in both cases. Our results indicate that attention-based models consistently outperform other models for most forecast durations, except for in the 6-hour range. Notably, we found that the Gaussian Kernel Averaging (GKA) model used in previous work (Pannatier et al., 2022) achieves satisfactory performance, despite its theoretical limitations, which we analyze in Appendix C.7. Moreover, our findings suggest that attention-based models, particularly MSA and TFS, exhibit superior performance in this setup. We also observe that the GKA model performs well for short time horizons when most of the information in the context is still up-to-date. However, as the time horizon increases, the GKA model's lack of flexibility is more apparent, and GEN is more competitive.

For the Heat Diffusion Experiment, we utilize the data set introduced in (Alet et al., 2019),

Table 3.3: Evaluation of the wind nowcasting task according to standard weather metrics, which are described in Appendix C.9. The optimal value of the metric is indicated in the parenthesis. MSA is the best model overall, with the lowest absolute error, a near-zero systematical bias, and output values that have a similar dispersion to GEN. We added the results of the GKA model for comparison as it is the best-performing model in the original chapter.

Model	RMSE (↓)	θ MAE (↓)	r MAE (↓)	Relative $BIAS_x$ (0.0)	Relative $BIAS_y$ (0.0)	rSTD (1.0)	NSE (↑)
CNP	10.99± 0.75	25.55± 1.22	9.22± 0.33	0.00± 0.09	-1.09± 0.03	1.25± 0.07	-0.23± 0.01
GEN	8.97± 0.06	22.56± 0.77	6.97± 0.05	-0.02± 0.03	-0.97± 0.21	**1.09± 0.07**	0.25± 0.02
GKA	8.44± 0.01	21.89± 0.02	6.65± 0.02	-0.02± 0.00	-1.78± 0.02	1.13± 0.00	0.31± 0.01
TFS (Ours)	7.99± 0.15	22.17± 1.20	6.48± 0.50	0.08± 0.10	-2.21± 2.67	1.17± 0.04	0.43± 0.08
MSA (Ours)	**7.36± 0.06**	**20.48± 0.48**	**5.67± 0.11**	**0.00± 0.02**	**-0.04± 0.64**	**1.09± 0.02**	**0.55± 0.05**

derived from a Poisson Equation solver. The data set consists of context measurements in the unit square corresponding to sink or source points, as well as points on the boundaries. The targets correspond to irregularly sampled heat measurements in the unit cube. Our approach offers significant performance improvements, reducing the root mean square error (RMSE) from 0.12 to 0.08, (MSE reduction of 0.016 to 0.007, in terms of the original metric) as measured against the ground truth [Table C.2].

For the Fluid Flow Experiment, both data sets are derived from (Li et al., 2021), subsampled irregularly in space. In both cases, MSA outperforms the alternative [Table C.2]. In the Darcy Flow equation, the TFS model with 20k parameters exhibits the best performance, but this task proved to be relatively easier, and we hypothesize that the MSA model could not fully exploit this specific setup. However, it is worth mentioning that the performance of the MSA model was within a standard deviation of the TFS model.

We conducted a Two-Day Weather Forecasting Experiment utilizing ERA5 data set measurements. The data set consists of irregularly sampled measurements of seven quantities, including wind speed at different altitudes, heat, and cloud cover. Our goal is to predict wind conditions at 100 meters two days ahead based on these measurements. MSA demonstrates its effectiveness in capturing the temporal and spatial patterns of weather conditions, enabling accurate predictions [Table C.2].

We compared our models and the baselines at different parameter ranges (5k, 20k, and 100k) to ensure that the models are comparable in terms of complexity. As the dataset size are limited we found that there was no need of scaling to larger paramters size. We noticed that the best performance was usually obtained by models in the range of 20k parameters and that the models which contained 100k parameters were often overfitting the data. More details can be found in Table C.2.

To summarize, our experiments encompass a range of tasks including high-altitude wind nowcasting, heat diffusion, fluid modeling, and two-day weather forecasting. Across these diverse tasks and data sets, our proposed model consistently outperforms baseline models,

showcasing their efficacy in capturing complex temporal and spatial patterns.

3.5 Understanding Failure Modes

We examine the limitations of CNP and GEN latent representation for encoding a context. Specifically, we focus on the bottleneck effect that arises in CNP from using a single vector to encode the entire context, resulting in an underfitting problem (Garnelo et al., 2018; Kim et al., 2019), and that applies similarly to GEN. To highlight this issue, we propose three simple experiments. (1) We show in which case baselines are not able to pass information in the context that they use for conditioning, and why MSA and TFS are not suffering from this problem. (2) We show that maintaining independent latent representation helped to the correct attribution of perturbations. (3) We show that this improved latent representation leads to better error correction.

3.5.1 Capacity of Passing Information from Context to Targets

Every model considered in this work encodes context information differently. Ideally, each should be capable of using every measure in their context efficiently. We will see that excessive bottlenecking in the latent space can make this difficult or impossible.

To demonstrate this result, we design a simple experiment in which each model encodes a set of 64 measures $(\mathbf{x}_c, \mathbf{y}_c)$, and is then tasked with duplicating the corresponding $\mathbf{y}_t = \mathbf{y}_c$ given the $\mathbf{x}_t = \mathbf{x}_c$. The training and validation set have respectively 10 000 and 1 000 pairs of sets of 64 examples. It is worth noting that the models have access to all the information they need to solve the task with near-zero MSE. We conducted several experiments, starting by randomly sampling 2D context positions $\mathbf{x}_c = (x, y)$ from a Gaussian distribution and computing the associated deterministic smooth function:

$$\mathbf{y}_c = \sin(\pi f x)\cos(\pi f y) \in \mathbb{R} \tag{3.17}$$

Where f is a frequency parameter that governs problem difficulty. The higher f is, the more difficult the function becomes, as local information becomes less informative about the output. We also consider as a harder problem to sample $\mathbf{y}_c$ randomly and independently from the position.

The results of this experiment, as shown in Figure 3.4, indicate that the CNP and GEN models are less effective in learning this task at higher frequencies. This inefficiency is primarily due to a phenomenon we define as a 'bottleneck': a situation where a single latent variable is responsible for representing two distinct context measurements. This

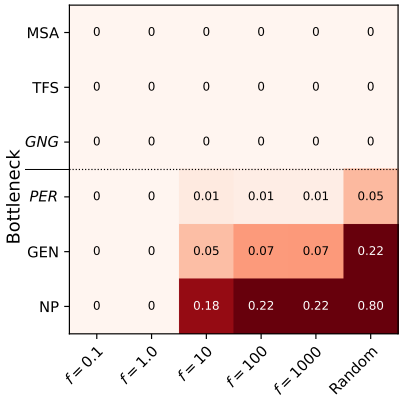


Figure 3.4: The results of the information retrieval experiment, evaluated using MSE, are considered satisfactory if the MSE is below 0.01. The first three rows depict models without bottlenecks. The x-axis represents data sets organized by increasing frequency, with 'Random' as the extreme case where the context value is independent of its position. Models with bottlenecks are sufficient when the learned function varies minimally in space. Models in italics denote hybrid architectures: *GNG* represents 'GEN No Graph', maintaining a latent measure per context, and *PER* indicates a transformer with a perceiver layer (Jaegle et al., 2021), introducing a bottleneck.

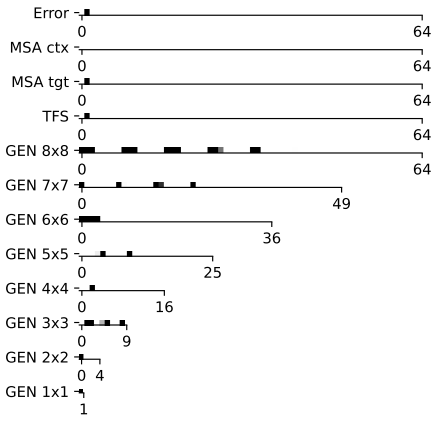


Figure 3.5: Gradients on the last layer of the encoder corresponding to an artificial error of $\epsilon = 10.0$ added to the second output. MSA maintains independent latent representation and gradients are exclusively non-zero for the latent associated with the error. We compare it to different GEN models each initialized with a graph corresponding to a regular grid of size $i \times i$ with $i \in \{1, \ldots, 8\}$. Due to the bottleneck effect, the gradients corresponding to one error are propagated across different latent vectors for GEN. Even when there are enough latents (GEN 8×8), GEN still disperse attribution because their distance-based conditioning that does not allow for a one-to-one mapping between targets and latents.

bottleneck impedes the models' ability to distinguish and retrieve the correct target value. In contrast, models with independent latent representations, like MSA, are not subject to this limitation and thus demonstrate superior performance in learning the task. To ensure that the effect arises from a bottleneck in the architectures, we created two hybrid models, trying to stay as close as possible to the original models but removing the bottleneck from GEN and adding one for transformer-based models. We call the first one GNG (for *GEN No Graph*), which adapts a GEN and instead of relying on a common graph, creates one based on the measure position with one node per measure. Edges are artificially added between neighboring measures which serve as the base structure for L steps of message-passing. This latent representation is computationally expensive as it requires the creation of a graph per set of measurements, but it does not create a bottleneck in the latent representation. We found that GNG is indeed able to learn the task at hand. We then followed the reverse approach and artificially added a bottleneck in the latent representation of attention-based models by using Perceiver Layer (Jaegle et al., 2021)

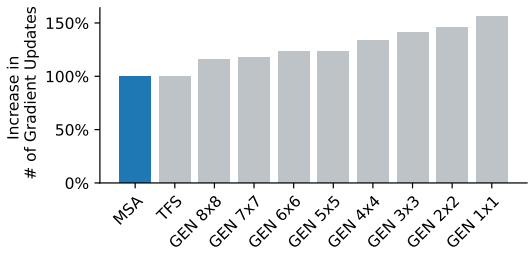


Figure 3.6: Comparison of the number of gradient updates required to correct an artificial error, compared to MSA (lower is better). The y-axis represents the increase in percentage in the number of steps required to reach a perfect accuracy compared to MSA. We compared MSA to different GEN each initialized with a graph corresponding to a regular grid of size $i \times i$ with $i \in \{1, \dots, 8\}$. It can be observed that all GEN take more steps to correct the same mistake, and the more entangled the latent representation is, the more time it requires to correct the problem.

with P learned latent vectors instead of the standard self-attention in the transformer encoder (and call the resultant model *PER*). When P is smaller than the number of context measurements, it creates a bottleneck and *PER* does not succeed in learning the task. If the underlying space is smooth enough, GEN, CNP, and *PER* are capable of reaching perfect accuracy on this task as they can rely on neighboring values to retrieve the correct information. This experiment demonstrates that MSA and TFS can use their independent latent representations to efficiently retrieve context information regardless of the level of discontinuity in the underlying space, while models with bottlenecks, such as CNP and GEN, are limited in this regard and perform better when the underlying space is smooth.

3.5.2 Improving Error Correction

In the following analysis, we explore how independent latent representations can enhance error correction during training.

To do so, we conduct an ablation study focusing on the hypothesis that maintaining independent representations is beneficial during training. We test this hypothesis by restricting the ablation to a straightforward scenario, where the models have perfect accuracy on all their outputs but one. To do so we start by pretraining the models to zero error on a validation set in a smooth case ($f = 1.0$). We then pass a fixed random validation batch to the different models and consider this output as the target for the experiment. We then deliberately add a perturbation $\epsilon = 10.0$ to one of the target values and backpropagate the error through the model, observing how the models adapt to this perturbation and the consequent impact on their latent variables.

In Figure 3.5, we present the norm of gradients at the last encoder layer of the latent

variables for different models, specifically MSA, TFS, and GEN, when correcting an error during training. It shows how MSA and TFS models experience a non-zero gradient only in a single latent variable, whereas the GEN model exhibits changes across multiple latent variables. This disparity suggests that models like MSA and TFS, which maintain one latent variable per output, are less constrained during training. They need only to adjust the specific latent corresponding to the error without affecting others. Conversely, models with entangled representations, as seen in GEN, must manage the dual task of correcting the error while ensuring other dependent outputs remain unchanged.

We think that the additional constraints might lead to slower training and to test that hypothesis, we measure the number of backpropagation passes required to return to zero error on the validation set [Figure 3.6]. We see that as the number of latent variables reduces, it leads to an increase in the number of steps required to correct the error. This result suggests that the entanglement of latent variables in GEN can lead to slower training and that maintaining independent latent representations can improve error correction. We make the hypothesis that this scenario extends to general training dynamics, proposing that models with independent latent representations can correct errors more efficiently due to their reduced constraints.

3.5.3 Encoding scheme

In this section, we evaluate the novel encoding scheme presented in Section 3.3.2, and we present the results in Table 3.4. We found that it reduces the RMSE from 8.47 to 7.98 in the wind nowcasting task and enables the MSA model to achieve the best performance with an RMSE of 0.08 for the Poisson Equation. Sharing the same mapping for positions is the appropriate inductive bias for encoding positions, as it eliminates the need to learn the same transformation twice. Since our data is irregularly sampled in space, the positioning of measurements and target positions significantly influence the prediction, as demonstrated in additional experiments, Appendices C.10 and C.11. We think that sharing the position mapping can link information from the context and target positions, which helps the model to understand better how the space is shaped.

3.6 Conclusion

In this chapter, we introduced an attention-based model to handle the challenges of wind nowcasting data. We demonstrated that the proposed attention-based model was able to reach the best performance for high-altitude wind prediction and other dynamical systems, such as weather forecasting, heat diffusion, and fluid dynamics when working with data irregularly sampled in space. We then explained why attention-based models were capable

Table 3.4: Relative performance improvement with shared encoding for position across various models. This improvement is calculated as $\frac{\text{Error}_{\text{shared}}}{\text{Error}_{\text{no shared}}}$, where $\text{Error}_{\text{no shared}}$ represents the mean error across sizes for models without shared encoding and $\text{Error}_{\text{shared}}$ is the mean error across sizes for models utilizing shared encoding. Detailed values are available in Table C.2. Scores exceeding 100% indicate superior performance without shared encoding. Generally, shared encoding enhances model efficiency, typically by a small margin, but significantly in some cases like the Poisson Equation. The shared encoding approach is likely advantageous as an inductive bias for position encoding. By naturally treating the positions of context and targets similarly, it can effectively use this information to develop more robust representations.

Architecture	Wind Nowcasting	Poisson Equation	Navier-Stokes Equation	Darcy Flow Equation	ERA5
CNP	100.7%	142.1%	154.1%	193.0%	100.6%
GEN	95.2%	63.5%	101.9%	77.9%	101.3%
TFS (Ours, baseline)	95.3%	79.5%	99.5%	99.3%	101.0%
MSA (Ours)	93.7%	24.8%	99.4%	97.2%	100.9%

of outperforming other models on that task and provided an in-depth examination of the differences between models, providing explanations for the impact of design choices such as latent representation bottlenecks on the final performance of the trained models.

Our work builds upon well-established attention models, which have demonstrated their versatility and efficacy in various domains. Although the core model is essentially a vanilla transformer, our architecture required careful adaptation to suit our specific requirements. We designed our model to be set-to-set rather than sequence-to-sequence, handling data in a non-causal and non-autoregressive manner, and generating continuous values for regression. The success of influential models like BERT (Devlin et al., 2019), GPT (Radford et al., 2018), ViT (Dosovitskiy et al., 2020), and Whisper (Radford et al., 2022), also closely resembles the original implementation by Vaswani et al. (2017), which further supports the effectiveness of the transformer framework across different tasks and domains.

Our model's scalability is currently limited by its quadratic complexity in the context size. Although this limitation does not pose a problem in our particular use cases, it can impede the scaling of applications. This is a significant challenge that affects all transformer-based models and has garnered considerable attention. The approach that we used in Chapter 2 is a straightforward way of replacing the self-attention layer if the memory footprint of the model becomes prohibitive, and we compare the strenghts and limitations of both approaches in Appendix C.4. There are other recent developments to tackle this scaling challenge, including flash attention (Dao et al., 2022), efficient transformers (Katharopoulos et al., 2020), and quantization techniques (Dettmers et al., 2022), which can enhance the feasibility of our approach for large-scale applications.

This application of the attention-based model is also limited to a specific use case where we

expect predictions of the model to be independent of each other and depend completely on the context. This product-law hypothesis is a strong requirement that allows the generation of all the forecasts at once and relies only on self-attention layers without masking. In the following chapter, we will present a more general approach to handle time series data with a causal structure and we will show how to adapt the transformer architecture to handle this kind of data.

Part III

Modeling Sequences from a Joint Distribution using Transformers

Chapter 4

σ-GPTs: A New Approach to Autoregressive Models

This chapter is a post-print version of:

> A. Pannatier, E. Courdier, and F. Fleuret. σ-GPTs: A New Approach to Autoregressive Models. In *Machine Learning and Knowledge Discovery in Databases: Research Track*. Springer Nature Switzerland, 2024a. ISBN 978-3-031-70358-4. (pdf). (code)

4.1 Introduction

Transformers demonstrate exceptional autoregressive capabilities across modalities. The traditional take for autoregression is to follow the natural order of the data, for example, left-to-right for text. In the case of vision, the usual scheme is to unfold the images following a raster-scan order and to use transformers to model the obtained sequence. In this work, we make a distinction between the order of the input data and the order of autoregression, highlighting that while they are typically aligned in most applications, they need not be. Our investigation involves training and generating sequences in a randomly shuffled order using transformers. While changing the sequence order is more challenging during training, it also reveals fascinating properties of the models.

This motivating task of this chapter is the second task of the MALAT project, which aims to model the ascent and descent of airplanes. The main difference is the nature of the modeled process. For wind nowcasting, the output of the model was assumed to only depend on the context and not on the other outputs. In the case of aircraft vertical rate prediction, the output is a sequence of altitudes that are dependent on each other.

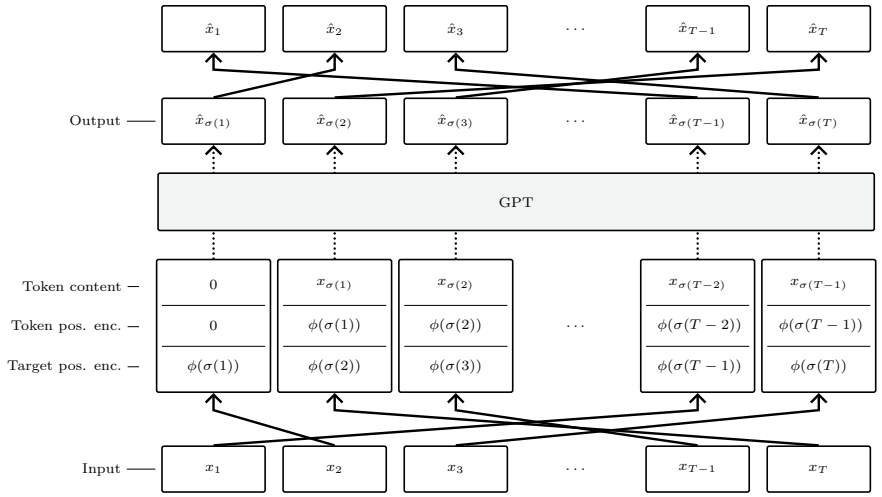


Figure 4.1: In our σ-GPT, an arbitrary shuffling order σ can be chosen on-the-fly for every sample. It induces an input order $0, \sigma(1), \sigma(2), \ldots$ and an output order $\sigma(1), \sigma(2), \sigma(3), \ldots$, where the input is first padded with a 0 to ensure a consistent number of tokens. Tokens are shuffled accordingly, and these orders are both encoded separately with two positional encodings concatenated to the input, allowing the model to sample consistently in the autoregressive process. The output is finally shuffled back to the true order.

	σ-GPT	GPT	Diffusion Models
Sample Anywhere	✔	✗	✗
Conditional Density Estimation	✔	✗	✗
Arbitrary Conditioning	✔	✔	∼
Infilling	✔	✗	∼
Burst-Sampling	✔	✗	✔
Log-likelihood Training	✔	✔	✗

Table 4.1: Comparison between our approach, a standard GPT, and diffusion models. Our model allows the sampling of a token at any position in the sequence, to model the remaining density according to a partially sampled sequence, naturally supports infilling, and can be used to sample the sequence by burst allowing faster generation. Compared to diffusion models, it can be trained easily using cross-entropy.

By breaking away from the standard autoregression order, one can use the model to predict the tokens in any particular order. With this scheme, the model is capable of predicting at any moment of the generation the conditional distribution of the remaining tokens. Having these estimates allows quantifying the possible outcomes of the generation at any given point. More interestingly, they can be leveraged to do rejection sampling, allowing to generate sequences by burst with a dynamical number of steps.

This work is structured as follows, we first introduce σ-GPTs and shuffled autoregression, and show that a model trained with this method combined with a curriculum method can even increase the performance of the underlying model. We then present the additional properties of σ-GPTs, summarized in Table 4.1, in particular for estimating conditional probabilities and we present our token-based rejection sampling scheme which allows for generating the sequence per burst and its theoretical properties. We evaluate our model and our scheme on three main tasks, which are open text generation, path-solving, and aircraft vertical rate prediction.

Contributions:

- Introduce σ-GPT, a novel architecture, with two positional encodings related respectively to the input and output order, that allows a causal transformer to generate sequences in any order which can be modulated on the fly for any pass through the model.
- Demonstrate that our method can reach similar performance as left-to-right trained autoregressive models when trained with a curriculum scheme.
- Demonstrate that our method can be used to generate samples in any order, allowing for the generation of samples conditioned on any part of the sequence.
- Introduce a novel token-based rejection sampling scheme that leads to the generation of samples per burst.

4.2 Methodology

4.2.1 σ-GPTs: Shuffled Autoregression

We propose a novel approach for training autoregressive models, which involves doing next-token prediction on a shuffled input sequence. We present σ-GPT, where σ denotes the permutation used to shuffle the sequence, and by GPT we mean a standard transformer decoder without cross-attention (or causal transformer encoder) such as (Radford et al.,

2019). To train such a model, each sequence is shuffled randomly during training. The model is then tasked to predict the next token in the shuffled sequence conditioned on all the tokens it has seen before. This training is done as usual with a standard cross-entropy loss. Besides the randomization of the order of the sequence and the addition of a double positional encoding, no other changes are needed to the model or training pipelines. For the rest of the chapter, we use 'left-to-right order' to mention the usual order in which models are trained, even in the case of 2D data which are usually mapped to a sequence using a raster-scan order. And we use 'random order' to mean that the input has been shuffled.

4.2.2 Double Positional Encodings

To be able to model sequences in any order, each token needs to have information about its position and the one of the next token in the shuffled sequence. Specifically, when handling a sequence of tokens alongside a given permutation σ, every token contains three distinct pieces of information: its value $x_{\sigma(t)}$, its current position $\sigma(t)$, and the position $\sigma(t+1)$ of the subsequent token in the shuffled sequence, that are all concatenated. The necessity for double positional encoding arises from the intrinsic characteristics of transformers. Given that each token attends to every previous token in a position-invariant manner, each token needs to contain information about its position in the original sequence, so other tokens can know where they are located. And each token needs to know the position of the next token in the shuffled sequence as it is the target of the prediction. The double positional encoding is the only architectural change needed to train autoregressive models in random order. In this work, we used the standard sinusoidal positional encoding (Vaswani et al., 2017) for both the input and output positional encodings.

4.2.3 Conditional Probabilities and Infilling

Our method allows making conditional density estimation of the rest of the sequence. It is capable of making predictions all over the task space conditioned on any known subpart of the task. This can be done by prompting the model with the known part of the sequence and then decoding, in parallel and in one pass, the remaining tokens. Such evaluations are not possible with autoregressive models trained in a left-to-right order, as they need to follow the specific order they've been trained in. Examples, showing that the model usually has good estimates of the unconditioned distribution, can be seen in Figures 4.2 and 4.3a.

Directly related to conditional density estimation, is that our method naturally supports infilling, as it is straightforward to prompt the model with the known part of a signal and

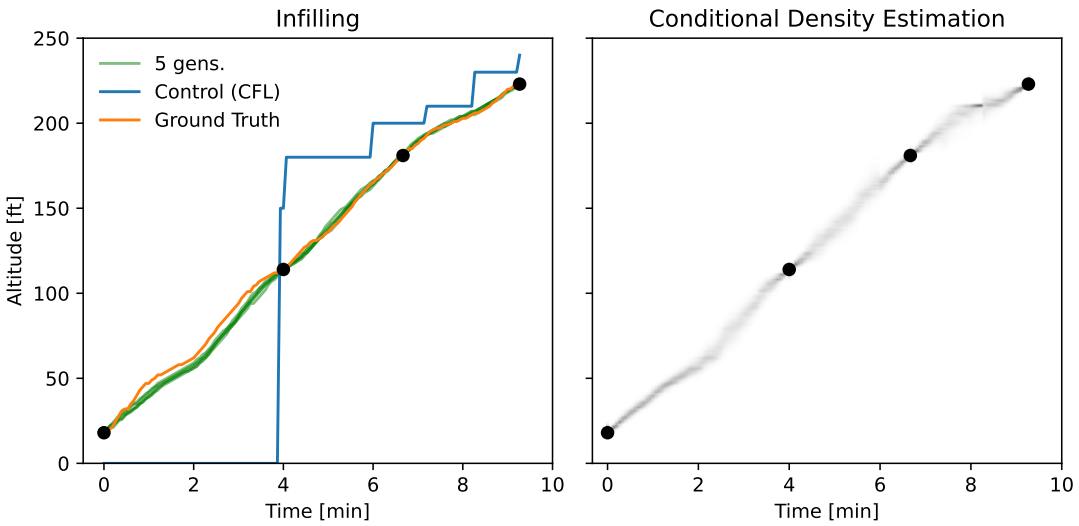
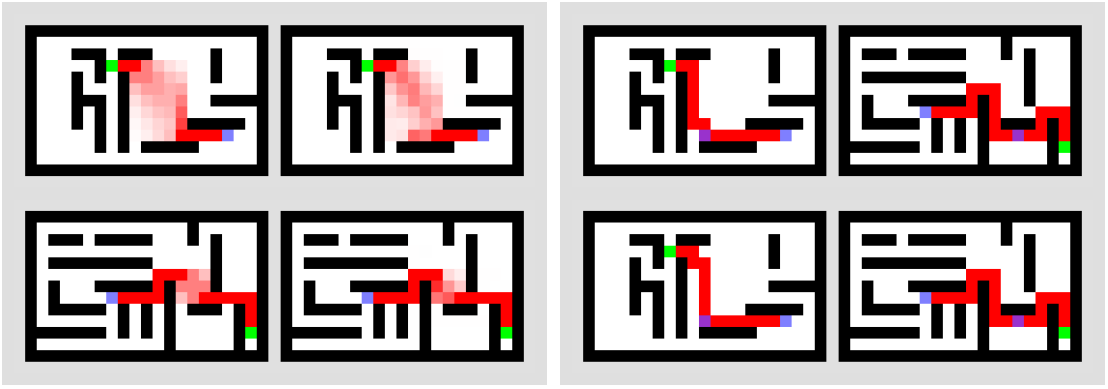


Figure 4.2: (Left.) We can infill the sequence by conditioning on the known part (black points). (Right.) We can also have estimates of the density at any point of the sequence.



(a) (Left.) The theoretical density of the optimal path in the maze. (Right.) The estimated probability of the class 'path' at every position before starting autoregression. We see that the model has good estimates of the true density.

(b) Two different conditional samplings for each maze. The known part of the path (purple) is prompted first, and the rest of the sequence can be completed coherently.

Figure 4.3: Conditional density estimation and infilling on the maze path-solving task.

to decode auto-regressively or by burst the rest of the signal. Figures 4.2 and 4.3a shows example of such samplings.

4.2.4 Token-based Rejection Sampling

Algorithm 3 Token-based rejection sampling, following notation of (Chen et al., 2023)

Given minimum target length T, y trained σ-GPT, and number of orders N_o
Given a prompt $x_i \in \mathbb{X}$ of length t_0 of initial tokens. ($\mathbb{X}$ can be the empty set)
Set $t = t_0$
while $t < T$ **do**
  In parallel, compute distribution conditioned on prompt $p(x_i|\mathbb{X}), \forall i \in t, \dots, T$
  In parallel, sample at every position $\tilde{x}_i \sim p(x_i|\mathbb{X}), \forall i \in t, \dots, T$
  Draw N_o random order σ and in parallel, compute all logits $q(x_i|\mathbb{X}, \tilde{x}_{\sigma_{<i}}), \forall i \in t, \dots, T$
  In parallel sample $T-t$ variables $u_i \sim U[0,1], \forall i \in t, \dots, T$ from a uniform distribution.

  In parallel, compute the acceptance decision $a_i = u_i < \min\left(1, \frac{q(\tilde{x}_i|\mathbb{X}, \tilde{x}_{\sigma_{<i}})}{p(\tilde{x}_i|\mathbb{X})}\right)$ for every order.
  Select the order that accepts the most tokens before seeing a first rejection.
  Keep that order and add the a accepted tokens before the first rejection to the prompt.

  Set $t = t + a$
end while

Autoregressive generation is a slow process as each token has to be generated sequentially. Even with caching strategies, this still scales linearly with the sequence length and it becomes prohibitively expensive for long sequences (Villegas et al., 2023). As our model allows for the generation of tokens in any order, we can leverage that fact and sample tokens in parallel at every position of the sequence. We can then evaluate the candidate sequence under different orders and accept multiple tokens in one pass. This algorithm runs efficiently on GPU as both the sampling at every position and the evaluation under different orders can be made in parallel, in a forward pass, and using an adapted KV-caching mechanism. We describe this caching mechanism more in detail in Appendix D.3. When conditioned on partially completed sequences the model outputs distributions that are compatible with different possible outcomes, and when evaluating under different orders for generation, the distribution of tokens is constrained to tokens that are compatible with the previous tokens seen in one given order. As both the sampling and evaluation can be done in parallel, we can compute the acceptance decision efficiently for every token.

This strategy outputs a decision for each remaining token, but the decisions made by models become sometimes nonsensical when two mutually exclusive tokens are part of the prompt. Once a rejection is seen, all subsequent accepted tokens in the order of evaluation should be discarded. Indeed, the scheme rejects tokens that are incoherent

with the ones already seen, and asking a model to make predictions based on incoherent tokens might lead to incoherent decisions. Using multiple orders allows keeping the one that accepts the most tokens in its evaluation. Even if it is dynamic, this algorithm can still easily generate multiple samples at once, by accepting the same amount of tokens for each sequence in the batch. Our rejection sampling algorithm is given in pseudo-code in Algorithm 3.

Other models such as Mask Git (Chang et al., 2022) or diffusion models (Ho et al., 2020; Austin et al., 2021) are doing generation by burst. However, these models usually require fixing the number of steps or a masking schedule beforehand. Our method on the other hand adapts dynamically to the underlying statistics of the data and thus does not require this extra hyper-parameter. We evaluate it on three synthetic cases to showcase this dynamic capability, we present the results in Section 4.3.8.

4.2.5 Other Orders

Our double positional encoding scheme allows for training and evaluating models in any order. Using a randomized order during training allows conditional density estimation, infilling, and burst-sampling at inference time. However the double positional encoding scheme allows any order to be used, and it can be used to train models in a deterministic order that is not left-to-right. As an example, we use a deterministic 'fractal' order to see how it compares to a random or left-to-right order. This order starts in the middle of the sequence then recursively goes to the first quarter and three-quarters of the sequence, and goes on recursively until all the positions have been visited. Such an order is fully deterministic, yet we make the hypothesis that this order leads to more difficult training for the model as it cannot rely on the locality of the information. We present the results in Section 4.3.5. Note that under perfect models, the order of modeling and decoding should not matter because of the chain rules of probability. We give more details about it in Appendix D.1.1.

4.2.6 Denoising Diffusion Models

Denoising diffusion models (Ho et al., 2020) is a family of generative models that can also be used to generate sequences in a few steps. They are trained to reverse a diffusion process that is applied to the data. Diffusion processes can be both continuous and discrete. In this work, we use as a baseline only the discrete diffusion case, in particular using a uniform diffusion process (Austin et al., 2021). To be able to compare the methods fairly, we use the same transformer architecture for both σ-GPT and the diffusion model, changing only the training objective. Compared to σ-GPT, diffusion models are not dynamic and require

a fixed number of steps to generate a sequence, independently of the underlying statistics of the data. They also don't natively support conditional density estimation and infilling.

4.3 Results

4.3.1 General performance

We tested our model across three main distinct tasks: language modeling, maze path solving, and aircraft vertical-rate prediction.

- Language Modeling: We used both the GPT-2 (123M) model on the Wikitext-103 data set (Merity et al., 2017) and GPT-2 (345M) on OpenWeb Text (Gokaslan and Cohen, 2019).
- Maze Path Solving: This task involves determining a valid path between a starting and ending point in 13 x 21 mazes featuring 15 barriers. Presented with an image of an empty maze with start and end points, the model is tasked with producing an image with a legitimate path.
- Aircraft Vertical-Rate Prediction: This task uses real aircraft trajectory data, with its aircraft type. The data represents trajectories conditioned by air traffic control directives. The model's objective is to predict the vertical trajectory from a plane's current altitude to a specified control level.

Additionally to these tasks, we created a synthetic benchmark for evaluating our burst-sampling algorithm.

- Product Data set: This toy example represents a pure product law case and is made of a sequence of length 100 with two classes (0,1) given by a Bernoulli law with p = 10%.
- Step Data set: This toy example comprises sequences of two classes (0,1) of length 100 which are 0 everywhere except on a step of length 10 placed randomly in the sequence
- Joint Law data set: This toy example represents a pure joint law and consists of a sequence of length 100 with 100 different classes, the model should predict a random generation of these different classes.

The general results of our models are presented in Table 4.2. These results indicate that training in a random order while requiring more compute-time as we describe

Table 4.2: General results. We report the validation perplexity for text generation, the test accuracy for the maze solver, and the mean squared error (MSE) for the vertical rate prediction. σ-GPT reaches a similar performance as GPT in text generation and maze solving and it outperforms GPT in the case of the vertical rate prediction. We report the validation perplexity for the text generation. For the path solver, we report the test accuracy on 1000 novel mazes. For the vertical prediction task, we report the mean squared error on the test set. We report the mean and standard deviation for the path-solving and the vertical rate prediction task. We do not report the validation error for the text generation for the discrete diffusion (Dis. Diff) as the training objective is different.

	Text-generation		**Path Solving**	**Vertical Rate**
	OWT Val Perp. (↓)	Wiki-103 Val Perp. (↓)	Accuracy (↑)	MSE (↓)
GPT	18.14	20.30	99.60 ± 0.70	274.8 ± 70.7
σ-GPT	18.64	16.69	98.30 ± 0.67	141.4 ± 4.1
Dis. Diff.	-	-	99.20 ± 0.67	105.94 ± 1.3

in Section 4.3.2, reaches similar performances to left-to-right trained models. For the text modeling, to have a fair comparison during training, we monitor the validation perplexity of the sequence evaluated in a left-to-right order. Training in random order for text modeling was plateauing at a higher left-to-right validation perplexity, but using a curriculum scheme allows reaching the same performances, as presented in Section 4.3.3. For the path solving and the vertical rate prediction, the models were able to reach the same left-to-right validation loss during training. In inference, we noticed a one percent drop in accuracy compared to diffusion models and left-to-right trained GPT. For the vertical rate prediction task, the data set that we used is limited to around 23.000 different sequences, we noticed that the standard left-to-right GPT was sometimes stuck repeating the same altitude, we think this is a modeling issue due to the small data regime. σ-GPT does not seem to suffer as much from this problem and offers a decrease in MSE. We hypothesize that this behavior comes from using a random order in inference which forces the model to fix some tokens over the whole sequence early in the generation. By doing so, the model gains the advantage of having a sketch of the whole sample and then concentrates on completing a coherent sample.

4.3.2 Training Efficiency

Modeling sequences in a random order is a more challenging task than modeling in left-to-right order. We think this is due to two main factors, at the beginning of the sequences models cannot rely on adjacent tokens to make educated guesses for the next token. Second some tasks are harder to learn in one direction than another and by modeling the data in any direction, we are always in the harder scenario. We give an example of one task that is harder to learn in one direction in Appendix D.1.

Table 4.3: Training efficiency. Number of steps/epochs required to reach the same performance and comparison with as GPT trained causally. As learning to predict in any order is a more challenging task, it is expected to need more computing time to reach the same accuracy. We don't report the standard deviation for text generation as we limited the training to one run.

Order	Text-generation	Maze Solver	Climbing Rate
σ-GPT	32500	78.0 ± 6.5	110.7 ± 4.5
GPT	16500	19.3 ± 4.9	25.0 ± 3.6

This implies that we expect and see an increase in the number of steps or epochs required to learn a task. As previously mentioned, we don't see experimentally a drop in the validation performance of our model in the case of the path-finding algorithm or the vertical rate forecasting, but the time to reach the same performance increased. In the case of text modeling, the models plateaued before reaching the same accuracy when trained in random order. We treat that case in the following section. We report in table Table 4.3, the increase in training steps or epoch to reach the same accuracy. We see that most of the time, the number of epochs or steps needed to reach the same performance drastically increases. We think again that this is due to the increased complexity of modeling the sequence without having to rely on local information.

4.3.3 Curriculum Learning

For text modeling, we found a gap in validation perplexity in the left-to-right order between models trained purely in a random order and models trained in a left-to-right order. We see in Table 4.4 that σ-GPT is stuck at larger perplexity in both Open Web Text and WikiText-103 (30.43 vs 18.14 and 39.85 vs 20.30). We found that training for longer and using larger model didn't help in reducing that gap. To solve that problem, we introduced a curriculum learning scheme where the model is shown first more sequences in left-to-right order and progressively learns to model the sequence randomly. Surprisingly using this scheme helped drastically the model which managed to get even better performance than left-to-right trained transformers in the Wikitext-103 case and reduce drastically the gap for models trained on OpenWebText.

4.3.4 Open Text Generation: t-SNE of Generated Sequences

To get a qualitative sense of the generated text by the different methods, We generate 3000 sequences of 1024 tokens with each method, embed each sequence using an embedding model, and then project the embeddings to 2D using t-SNE. We present the results in Figure 4.4. We used Open-AI `text-embedding-3-small` (OpenAI, 2024) to embed

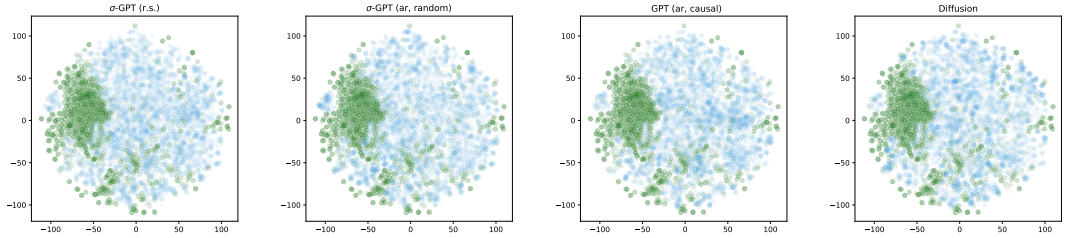
Table 4.4: Curriculum learning. We monitor the Validation Perplexity using a left-to-right order during training to have a comparable evaluation. We see that there is a gap between the model trained purely in a left-to-right fashion (GPT) and others trained in a random order (σ-GPT). Training for longer and larger models didn't help in removing that gap. We introduce a curriculum learning scheme that starts presenting the model with some percentage of the data (written in the corresponding label) in a left-to-right order at the beginning of the training and goes linearly to 100% of sequence in a random order at the end of the training. We see that training with this scheme removes the gap between σ-GPT and regular GPT and it reaches even better than left-to-right performance in the WikiText-103 case.

	Text-generation Val Perp. (↓) Min. Left-to-Right
Openweb Text - GPT (345 M)	
GPT	18.14
σ-GPT curr. 50%	18.64
σ-GPT no curr.	30.43
WikiText 103 - GPT (128M)	
σ-GPT curr. 50%	16.69
σ-GPT curr. 100%	19.38
σ-GPT curr. 10%	19.45
GPT	20.30
σ-GPT no curr.	39.85

the generated sequences into a single 1536 vector embedding. We represent as green embeddings of sequences of the validation set, used as reference. We compute the t-SNE using the whole 15'000 embeddings and then plot each method (blue) and the other considered method (small gray dots). We first see that embeddings of GPT, σ-GPT, σ-GPT with burst-sampling, and diffusion are spread over the whole space, showing that the model can generate sequences that are coherent with the validation set.

4.3.5 Training and Generating in Fractal Order

We describe here the results that we get when training a GPT using a deterministic, but not left-to-right order. We described the order in Section 4.2.5. We train a GPT using this specific order for the different tasks and present the results in Table 4.5. We found that training in that order was as difficult for the model as training in a random order, and we noticed a small drop in performance compared to σ-GPT. We suspect that this is due to the high discontinuity of the order of the sequence, which is such that two consecutive tokens are seen far away in the sequences. When predicting the first tokens, the model therefore cannot rely on information contained in neighboring tokens to make its prediction. As



(a) Rej. Sampling (b) σ-GPT (c) GPT (d) Diffusion

Figure 4.4: 2D t-SNE of `text-small-3-embeddings` in arbitrary units of 3000 sequences generated by each method. We compute the t-SNE of all the embeddings together, and then we display in each graph the embeddings of the validation set (green), the embeddings of the corresponding method (blue), and the embeddings of the other methods (gray). We see that the embeddings of the generated sequences have the same overall distribution compared to validation sets, which seems to indicate that *GPT*, σ-GPT, σ-GPT with burst-sampling, and diffusion models can generate sequences of similar quality.

	Text-generation Val Perp. (↓), Rand. ord.	**Path Solver** Test Acc (↑)	**Vertical Rate** MSE (↓)
σ-GPT	24.46	98.30 ± 0.67	141.4 ± 4.1
Fractal GPT	27.79	98.00 ± 1.94	145.7 ± 2.6

Table 4.5: Results for GPT trained in a fractal order compared to a standard left-to-right (GPT) and random (σ-GPT) order. We found that training models in this highly non-continuous order is as hard as training them in a random order, and additionally, models trained in that order cannot be used for conditional density estimation, infilling, or rejection sampling. For text modeling, we report the model perplexity on the validation set, in a random order for σ-GPT and in fractal order for the fractal GPT, as left-to-right validation perplexity is meaningless for fractal GPT which did not see sequences in that order during training.

the training behavior seen in models trained in random and fractal order is similar, we think that the drop in training efficiency comes more from the fact that the model cannot exploit this neighboring information than changing the order at every batch.

Additionally, models trained in a fractal cannot be used as such for infilling and conditional density estimation and therefore cannot be used with our rejection sampling scheme. As the order is fixed for every batch, it might not even need to have a double positional encoding.

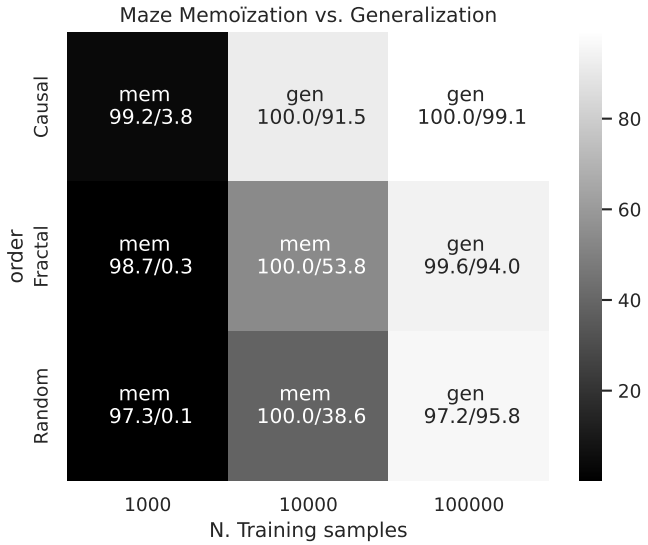


Figure 4.5: Number of examples needed to switch from memorization to generalization. The model is trained on a restricted data set size in the path-finding task. We see that the model trained in a random order needs more examples to switch from memorization to generalization. At 1k samples both models are fully in a memorization regime, at 100k both generalize but in between, at 10k, the model trained in a random order is still in a memorization regime.

4.3.6 Memorizing

As learning sequences in any direction is harder than modeling them under a predefined order, we also expect that the critical data set size when the model switches from memorization to generalization will increase. We follow the same hypotheses than (Varma et al., 2023), namely that the model has two mechanisms, one generalizing and one memorizing the data. As the mechanism of generalizing is more efficient as the data set grows it will be selected by gradient descent once the size of the data set gets beyond a critical size. As learning in a random order is a more difficult task, we expect that generalization is more difficult in that setup as well, hence the memorization regime should hold for bigger data set sizes. We reduce drastically the training data set size in the case of the path-finding task and we present the results in Figure 4.5. Once it gets to 1000 examples both models trained in left-to-right, fractal, and random order are in a memorizing regime, getting perfect accuracy on the training data but very low on the validation data. Conversely, once the data set gets bigger than 100k examples models trained in all the different orders are in a generalization regime. The transition happens in between and we find that it happens faster in the left-to-right order: at around 10k samples, the models trained left-to-right can generalize, while models trained in a random order are still in a memorization regime. We see also that models trained in a fractal order start generalizing faster than models trained in a random order, suggesting that the model can rely on seeing always the same

order to generalize more rapidly.

4.3.7 Infilling and Conditional Density Estimation

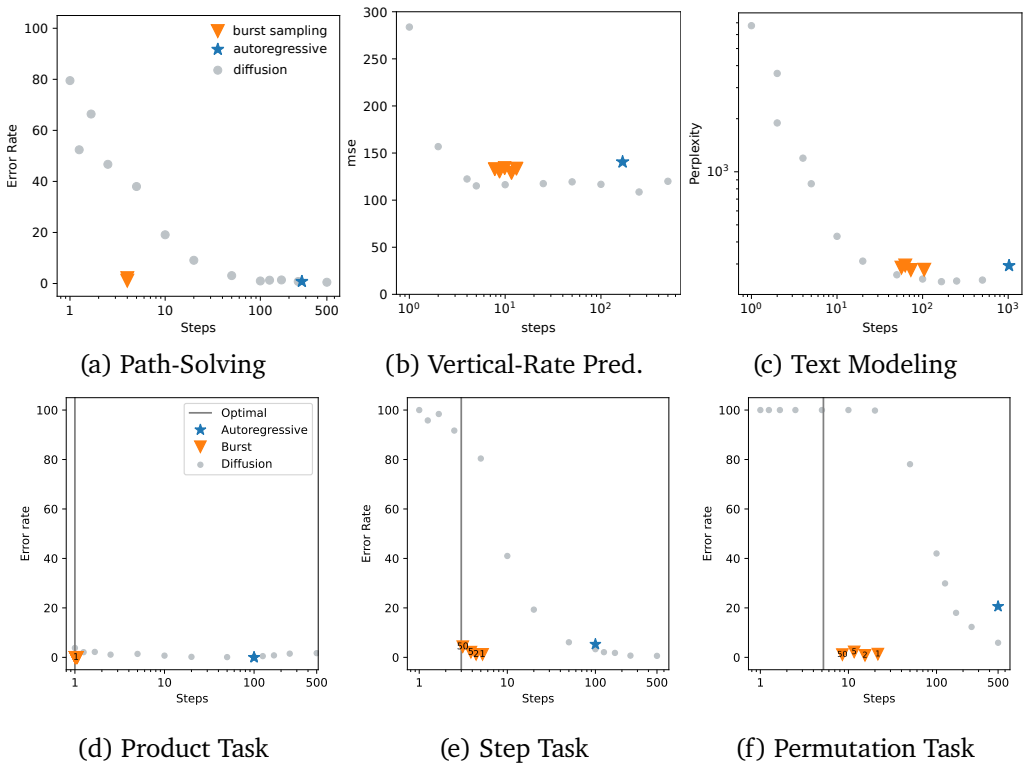
We show in Figure 4.2 that our model can be used to infill the sequence by conditioning on the known part of the sequence. In this figure, the larger points are part of the prompt and one can see the generated sequence complete sequence that matches the prompt. This figure also shows that our model has good estimates of the density at any point of the sequence. We represent at each point in the sequence the probability of the next sample given the known part of the sequence as shades of gray, the darker the more probable. We see that during generation, the model sees multiple possible outcomes that are coherent with the known part of the sequence. They are then constrained to a single sequence during the sampling. For left-to-right trained models, we can only have estimates of the density for the next tokens and we can't know what the model estimates for the rest of the sequence. In the case of the path-finding task, we show in Figure 4.3a that the model conditioned only on an empty maze has good estimates of the true density of the optimal paths, highlighting that the model has already partially solved the problem before starting generation. During sampling, the order of the generation and the sampling procedure influence which path is selected from this joint law. We show as well in Section 4.2.3, that we can constrain the generation of mazes and that the model can generate coherent samples based on some prompted tokens that can be chosen on the fly.

We also give some interactive examples of text generation in the supplementary material.

4.3.8 Token-based Rejection Sampling Scheme

We applied our token-based rejection sampling scheme to the problem of text generation, path-finding, and vertical rate forecasting Figures 4.6a to 4.6c. We found that in the three cases, our method was able to generate samples of comparable quality with an order of magnitude fewer steps than the autoregressive method. We compared to discrete diffusion models as well, and we can see that our method always outputs coherent samples, while the samples generated by the diffusion model are sometimes incoherent if the number of steps is not high enough. In the case of path-solving and in the three synthetic tasks, we see that at a comparable number of steps for both methods, our model shows better generation quality.

We tested our token-based rejection sampling scheme on three synthetic cases. We estimate the number of steps required to generate the sequence using a perfect model in Appendix D.2. We found that our scheme was close to the optimal heuristics in all three cases. In the case of the product data set, requiring only one step to accept the sequence.



(a) Path-Solving (b) Vertical-Rate Pred. (c) Text Modeling

(d) Product Task (e) Step Task (f) Permutation Task

Figure 4.6: We plot the performance vs steps of our σ-GPT used for autoregression in random order (blue), to σ-GPT with rejection sampling per burst (orange) against diffusion models (gray). We denote in the text the predefined number of steps chosen for generation in the diffusion models. For rejection sampling, we note the number of orders used for the evaluation. We see that increasing the number of orders leads to a decreased number of steps. For the synthetic tasks, we also represent heuristics for the optimal number of steps needed to generate the sequence (gray line), as described in Appendix D.2, and we see that our scheme is close to this heuristics.

In the case of the step data set, our scheme required four steps to accept the sequence. In the case of the joint law data set, our scheme required a few more steps, which is expected as the task is more complicated. We see that it manages to generate valid samples with a number of steps close to the optimal heuristics and with a large increase in performance compared to diffusion models at the same step.

4.4 Related works

Shuffling in Language Models: The objective of σ-GPT is to model shuffled sequences. XLNet (Yang et al., 2019) uses a similar objective and shuffling of the sequence order as a pretraining task for sentence encoding in the context of natural language understanding. While both approaches are modeling shuffled sequences, they still differ in their imple-

mentations: we use a double positional encoding and a regular causal mask instead of masking based on two streams and the modification of the attention matrix. The two models differ as well in their applications, XLNet is used to encode information similarly to BERT while our approach is generative. A recent approach by (Golovneva et al., 2024), trains a transformer both on a left-to-right and right-to-left order to solve a classic problem of the transformer model to understand tokens relations in both directions. In a similar setting, it has been shown that text corpora have a preferred left-to-right order of generation (Papadopoulos et al., 2024) and that training in reverse order can lead to a decrease in performance. This is a possible explanation of why training σ-GPT on text generation needed extra care compared to other tasks. In vision, the raster-scan order is mainly used to unfold 2D patches into sequences. However, a recent work by (Kakogeorgiou et al., 2024) showed that using a set of predefined orders of unfolding the images can improve performance. They pass the order information not by using a double positional encoding, but by a fixed number of beginning-of-sequences tokens.

Burst-Sampling Scheme: Other works are trying to solve the problem of the linear time required by autoregression using burst-sampling. Maskgit (Chang et al., 2022), for example, uses a BERT-like Masked Language Model (MLM) and a custom decoding scheme, which samples multiple tokens at the same time to generate an output. The number of tokens generated at each pass is fixed by a masking schedule and a confidence-driven decoding scheme is used to choose which tokens to predict next. Another approach, (Lezama et al., 2022) relies on an auxiliary model to guide the generation process. Alternatively, the approach of (Lee et al., 2022) focuses on generating preliminary drafts of an image, which are then iteratively improved. Current Video generation methods (Chang et al., 2023; Villegas et al., 2023) are leveraging a MaskGit-like approach (Chang et al., 2022) for generation because autoregressive generation of video frames would be too costly. Our rejection sampling scheme allows the generation of sequences by burst but in contrast to other schemes, the number of tokens accepted is dynamic and depends on the data being modeled. This allows for faster generation when the underlying data distribution is simple.

Discrete Diffusion Models: Diffusion models are also able to generate sequences in a few steps. We compared our approach with a discrete diffusion baseline (Austin et al., 2021). In the original work, discrete diffusion was also used to generate text, however without the possibility of conditional density estimation and infilling. Most of the diffusion approaches do not support conditional density estimation and infilling by default. Consistency Models (Song et al., 2023), which be adapted from continuous diffusion models, can be used to infill images but in a continuous case. However, there have been recent discrete text-diffusion models that allow infilling (Lou et al., 2024; Gulrajani and Hashimoto, 2023).

4.5 Conclusion

In this chapter, we introduced σ-GPT which adds novel capabilities of traditional left-to-right GPTs (Radford et al., 2018).

Training GPT-like models in different orders offer different desirable properties. It allows for the conditional prediction based on any subset of the tokens of the sequence, it naturally can be used for infilling, and as the model can do partial prediction, we can leverage them to do rejection sampling and accept multiple tokens at the same time during generation. Our findings indicate that conditional prediction learned by the models matches the theoretical partial distribution showing that the model is indeed able to understand and reconstruct the signal in any order. As the training objective of modeling sequences in any order is harder than training in a fixed order, it has an impact on the training efficiency, and in a small data set size, we show that it leads to more memoïzation. Finally, we showed that our model was able to generate sequences by burst using a novel per-token rejection sampling scheme, reaching optimal heuristics in some cases and decreasing the number of steps needed for generation by an order of magnitude.

Conclusion

In this thesis, we have extended the capabilities of attention-based models in two different ways. First, we added the capacity for these models to handle irregularly sampled data. Second, we allowed the sequences to be generated in any order chosen dynamically at inference time. These added capabilities extended the use of attention-based models to new real-world and domain-specific tasks, such as wind speed nowcasting and ascent and descent forecasting. We reached this goal by modifying a simple, robust, and scalable architecture: the Transformer (Vaswani et al., 2017).

This thesis was split into three parts: the first one focused on alternatives to the attention mechanism. We presented first a baseline and then an MLP-based alternative to the self-attention mechanism that scales linearly instead of quadratically with the sequence length. The second part presented our solution to the problem of wind nowcasting using a transformer encoder pipeline. In the last part of the thesis, we focused on the generation of complex joint signals with transformers.

In Chapter 1, we presented an averaging baseline for wind nowcasting, which can be interpreted as a proto-attention mechanism where the attention weights are not learned but a function of a modulated distance from the query to the context points. We showed that, despite its simplicity and some inherent limitations, this model was able to give satisfactory performances.

In Chapter 2, we introduced an efficient MLP-based alternative to the self-attention mechanism called Hypermixer. It extends the MLPMixer architecture to different sequence lengths by generating the weights of the first MLP dynamically based on the input data using hypernetworks.

In Chapter 3, we presented our solution to the wind nowcasting task using an attention network. The wind-nowcasting problem has the following properties: data came as sets of points sparsely placed in space, with no clear underlying structure, and predictions had to be possible at any place, conditioned on a context of past measurements. In attention networks, tokens attend to each other in a position-invariant way, leading to a position-equivariant architecture. This was the correct inductive bias for working with sets of

measurements. Furthermore, the generated forecasts should only depend on the context and the queried position and are not directly dependent on other generated samples. This allowed us to use a full-attention mechanism to make all predictions in one forward pass.

In Chapter 4, we proposed a new way of generating signals where we do not have this independence property. The traditional autoregressive approach generates sequences one token at a time in a fixed order, which has two drawbacks: first, it is a slow process, as it requires a forward pass per token and cannot be parallelized. Second, fixing the order of generation can lead to artifacts in the generated sequences such as the repetition problem.

We proposed to generate sequences in any order and we showed that models are usually capable of learning this harder task. We showed as well that models, given a partial sequence, have good estimates of the rest of the sequence. This fact was intuitively known for left-to-right models, but training in a random order highlighted that property directly. In this chapter, we also found that training models in that way increased the memoïzation problem, which was not expected. However, it reduced the repetition problem of autoregressive models and allowed us to devise a rejection sampling scheme leading to a sublinear generation time.

Future Works

In this thesis, we focused on a pure deep-learning solution to real-world problems. We offered a relatively small-scale solution that is promising for the task of wind nowcasting and ascent and descent forecasting. There are of course many other ways of approaching the problem. This section outlines different ways which could have been considered, or which could be complementary to our pure deep learning approach.

Completing the approach

Instead of having a purely data-driven approach to the modeled problem, one could rely on prior knowledge of the phenomenon. This could be achieved either by including physical knowledge about the phenomena or by enhancing the output of a more traditional simulator.

Physics-Informed Deep Learning The simplest strategy to guarantee that the output of a model is consistent with some known physical process is to hard-code the physical rules as a constraint or to give them as an additional loss.

In the case of wind nowcasting, generally, the wind field can be considered as divergence-

free (Kabanov et al., 2021). When using a data-driven approach, this is a property that the model has to learn and which might not be guaranteed. This is a constraint that can be added to the model, either by adding it as a loss or by designing the model in such a way that it respects this property. In the case of ascent and descent forecasting, a similar approach could have been used to ensure that the solution that the model outputs is consistent with the properties of the physical model.

This is the main strategy of Thuerey et al. (2021). This is a possible direction of improvement, which is straightforward to include in the model. However, it requires precise domain knowledge and should be carefully designed and adapted for each new task.

There is a way of learning these constraints as well (Alet et al., 2021). Learning the hidden physical constraints of the data and being able to inform the practitioner can be a powerful tool to understand the data better and to improve the model. It could also be used to hard-code these constraints in the model, which would lead to a more interpretable and robust model, and better guarantees on the output, which is crucial for the regulation of safety-critical applications.

Enhancing classical simulators Another approach, which is often used in weather forecasting systems, is to rely on deep learning to correct the output of a classical simulator. This approach offers, for example, smart data-driven inpainting of a too-large output.

As weather simulators are costly, running them with a fine resolution is computationally intensive. Large-scale weather forecasting system usually works with a resolution of a few kilometers (Météosuisse, 2014). Even at this reasonably large scale, the computational needs of traditional PDE solvers are reserved for national entities or large companies. In this thesis, we did not follow this approach, as we lacked access to simulator results. However, we have seen many times in this thesis that attention-based models are capable of handling different types of data as input, without needing much adaptation. It would be interesting to see how much the forecasting capabilities of the models can be improved by integrating coarse forecasts coming from national weather agencies.

For vertical-rate forecasting, it is mostly the same. In this thesis, we focus on a purely data-driven approach. However, there is an existing physical model of ascent and descent called BADA. The problem is that most of its parameters are unknown. Still, it could be used with averaged parameters and fed as an additional input to the transformer.

Scaling up

In this research, we work on data sets limited to a few weeks of data. Scaling up larger models and years-spanning data sets seems to be the logical next step to offer more robust

models and better predictions. These small-scale data sets were, in our opinion, the reason why artifacts appeared during generations, which motivated our need for a novel autoregressive scheme.

Repetitions in generation are still a problem for decoder models. But, experimentally, scaling up data sets and models seems to have solved this problem at least partly (Li et al., 2023a), explained by the fact that better modeling capabilities lead to models that better perceive that these repetitions are unnatural. However, even with multi-billion parameter architectures, these modeling problems can still occur. The main strategies for mitigating these issues are to introduce frequency-penalities, ad-hoc interventions, and to remove repetitions in the training corpora (Li et al., 2023a).

Final Remarks

In July 2020, a few months after I began my thesis, GPT-3 was released (Brown et al., 2020). This was a groundbreaking development, marking the first time a model appeared to truly understand text prompts and produce coherent completions. Previous models, by contrast, frequently strayed out of context.

Since the introduction of GPT-3 (Brown et al., 2020), the pace of advancements in the deep learning field has only accelerated, showing no signs of slowing down. Conducting research in such a rapidly evolving environment is incredibly motivating. Problems I anticipated would take years to solve are now being addressed in mere weeks. Being in a PhD program provides a unique vantage point to witness this extraordinary progress, often leaving me in a state of wonder.

A minor downside of this rapid progress is the challenge of keeping up with the latest improvements and novel methods. But, observing the increasing computational budgets of companies and universities, along with the expansion of deep learning into vastly different domains, is encouraging. A good example of this development is the transition for the Swiss National Weather Agency (Météosuisse) from a purely numerical solver approach (Météosuisse, 2014) to a GPU-based hybrid approach between deep learning and traditional methds (Météosuisse, 2024a,b). This access to larger GPU clusters is likely to lead to even more impressive model capabilities and the speed of innovation is likely to continue to increase.

In this fast-paced regime, identifying robust architectures that are domain-agnostic and withstand the test of time is crucial. Transformers, with their ability to scale reliably and achieve state-of-the-art performance across various domains, seem to be a key component in this ongoing search.

Appendices

Appendix A

Appendix - Chapter 1

A.1 More details on the KDTree implementation

We made extensive benchmarks with KDTrees using algorithms of the Scikit-Learn library (Pedregosa et al., 2011). We made ablation studies by deleting the masking and the scaling of the metric to see its impact on the runtime. We tried to vary the different leaf-size parameters. The idea is that it might be more efficient to use the linear search in a small number of bigger cells than to have plenty of small cells to check. Our benchmark shows that for the standard data set, the sweet spot is around a leaf size of 100 elements. Tables A.1 to A.3 lists all results for the different KDTrees.

A.2 Computing the distance to all the points of the batch to all segments

The algorithm to compute the distance from a batch of points to all the segments is described in Listing A.1. It is important to have a good representation of the segments to compute the distance efficiently. For example, when considering the formula for the distance in the publication [Eq. 6,8],we see that we need to know A, t_A, B, t_B for each point. We can precompute these quantities and group them in two $T \times 3$ matrix $\mathbf{A}$, $\mathbf{B}$, two $T \times 1$ matrix $\mathbf{t_A}$ and $\mathbf{t_B}$. We can precompute the unit vector $\vec{u} = \frac{\overrightarrow{AB}}{\|\overrightarrow{AB}\|_2^2}$ and store it in a $T \times 3$ matrix $\mathbf{U}$. We can compute the error efficiently by storing the maximal Euclidean error by segment in a $T \times 1$ matrix $\mathbf{E}$. During the search, all the batch points P can have a different scaling $\vec{\sigma}$. We group these scaling factors in a $P \times 3$ matrix. We define two matrices σ_{xyz} and σ_t to make the notation easier. The first one is a $M \times 3$ matrix, where the first two columns are repeated because the scaling on x and y are assumed to be the same. The

Table A.1: Comparison with KDTrees in different setups on the original data set.

Leaf-size	Method	Creation [ms]	Query [ms]	Total
10	KDTree	3546	6.97	1:56:06
	cKDTree	30	0.12	2:00
	scaled KDTree	3822	6.98	1:56:26
	scaled masked KDTree	3822	97.76	27:07:14
100	KDTree	492	3.13	52:08
	cKDTree	30	0.11	1:53
	scaled KDTree	500	3.61	1:00:12
	scaled masked KDTree	500	13.34	3:43:32
1000	KDTree	167	3.26	54:22
	cKDTree	30	0.13	2:18
	scaled KDTree	168	3.98	1:06:19
	scaled masked KDTree	168	4.89	1:21:34
	TNN CPU	1000	1.03	17:08
	TNN GPU	9000	0.16	2:43
	Linear search CPU	–	9.03	2:30:32
	Linear search GPU	–	2.55	42:28

Table A.2: Comparison with KDTrees in different setups on the Smoothed Random Walk data set.

Leaf-size	Method	Creation [ms]	Query [ms]	Total
10	KDTree	9057	16.60	4:36:37
	cKDTree	37	0.19	03:09
	scaled KDTree	9456	18.13	5:02:07
	scaled masked KDTree	9456	–	–
100	KDTree	785	6.02	1:40:23
	cKDTree	37	0.17	2:53
	scaled KDTree	762	14.95	4:09:14
	scaled masked KDTree	762	13.79	3:49:50
1000	KDTree	225	6.07	1:41:06
	cKDTree	36	0.25	4:11
	scaled KDTree	230	7.18	1:59:36
	scaled masked KDTree	230	6.61	1:50:07
	TNN CPU	1000	1.60	26:35
	TNN GPU	5000	0.39	6:28
	Linear search CPU	–	11.95	3:19:09
	Linear search GPU	–	2.51	41:48

Table A.3: Comparison with KDTrees in different setups on Random Points.

Leaf-size	Method	Creation [ms]	Query [ms]	Total
10	KDTree	3853	15.39	4:16:26
	cKDTree	36	0.25	4:05
	scaled KDTree	4070	17.27	4:47:49
	scaled masked KDTree	4070	17.07	4:44:35
100	KDTree	742	6.61	1:50:12
	cKDTree	36	0.26	4:17
	scaled KDTree	908	9.14	2:32:19
	scaled masked KDTree	908	8.25	2:17:33
1000	KDTree	367	7.03	1:57:09
	cKDTree	36	0.50	8:18
	scaled KDTree	373	9.14	2:32:25
	scaled masked KDTree	373	4.89	1:21:34
	TNN CPU	1000	15.51	4:18:34
	TNN GPU	5000	1:36	22:40
	Linear search CPU	–	7.43	2:03:51
	Linear search GPU	–	1.72	28:42

second one is a $M \times 1$ matrix σ_t that corresponds to the temporal part's scaling.

```
def distance_to_segments(batch, t, t_w, A, t_A, B, t_b,sigma_xyz,
    sigma_t U, E):
    # Shape: [P, T, 3]
    AP = -(A.unsqueeze(0)-batch.unsqueeze(1)
    # Shape: [P, T]
    delta = clamp(AP * U).sum(2), 0, 1)
    # Shape: [P, T, 3]
    P_AB = delta.unsqueeze(2)*(B-A) + A
    # Shape: [P, T, 3]
    D = ((P_AB-batch)**2) * sigma_xyz.unsqueeze(1)
    # Shape: [P, T]
    D = D.sum(2)

    # Shape: [P, 1]
    tw = (t-time_window).unsqueeze(1)
    # Shape: [P, T] += [P, 1]
    D += clamp((t_B-tw), 0)**2*sigma_t.unsqueeze(1)
    # Shape: [P, T]
    D[t_A > t] = float("inf")

    # Shape [T]
    error = sigma_xyz.max(1).unsqueeze(1).matmul(E)
    # Shape [P, T]
    return clamp(D - error, 0)
```

Listing A.1: Pseudocode for computing the distance from all points in a batch to all segments.

Appendix B

Appendix - Chapter 2

B.1 Extended Related Work

In this section, we provide an extended version of the related work.

B.1.1 Green AI

Schwartz et al. (2020) challenges the current pursuit for higher accuracy at the cost of larger computation with the notion of "Green AI". Moreover, Strubell et al. (2019) estimated the monetary and environmental cost of large model pretraining. Apart from being problematic environmentally, they argue that the monetary cost of pretraining is too high to be widely accessible for most researchers. In a research community that focuses on task performance, low-resourced researchers would be disadvantaged. Therefore, metrics that take the cost of reaching a result are important to consider (Schwartz et al., 2020). The metric $Cost(R) \propto E \cdot D \cdot H$, is proposed and discussed in Section 2.2. However, reporting a single metric $Cost(R)$ is often ambiguous. Therefore, in our experiments, we consider the factors E, D, and H.

To measure the computational cost per example E, Schwartz et al. (2020) propose a count of the floating point operations (FPOs) required. In our experiments, we adopt this metric and further include wall-clock time for a practical application. The component D evaluates the quantity of training data needed to reach a given accuracy or the performance of a model in a low-resource scenario (Hedderich et al., 2021; Chen et al., 2021). Finally, the component H measures the cost associated with hyperparameter tuning. This is reported using *expected validation performance* introduced by Dodge et al. (2019, 2021), which computes the validation performance one would yield in expectation after k hyperparameter trials of random search (Bergstra and Bengio, 2012).

Current literature does not focus on all facets of Green AI as formalized as $Cost(R)$. Typically, improving efficiency involves making existing models more accessible. For example, improving accessibility through model distillation (Sanh et al., 2019) or adapter modules (Houlsby et al., 2019). Another avenue involves reducing the computational complexity, with examples: prompt-tuning (Schick and Schütze, 2021), self-attention in Transformers (Child et al., 2019; Beltagy et al., 2020; Katharopoulos et al., 2020, et cetera). The latter approach is similar to our work. However, they focus on the processing time of a single example E and do not consider the other facets of Green AI. In chapter 2, we focus on MLP-based approaches, which we argue will have improvements in all facets of Green AI due to their simplicity.

B.1.2 MLP-based Models

The vision domain has seen promising results with purely MLP-based models (Tolstikhin et al., 2021), however, they lack the desired inductive biases for NLP. Some desirable properties for modeling language include: **i**) *position invariance*, which is important for generalization, **ii**) *adaptive size* for variable-length inputs, **iii**) a *global receptive field*, which allows interactions to not be limited to small token neighborhoods, **iv**) *learnabilty* allowing for universal applicability to various tasks, and **v**) *dynamicity* which implies that output is conditioned on the input. MLP-based models are typically not used for NLP as the inductive biases of position invariance, adaptive size, and global receptive field are non-trivial for MLPs.

Several methods try to overcome the lack of adaptivity to size by introducing shifting operations and local windows. Yu et al. (2022a) and Lian et al. (2022) use spatial shifting to pass the information of adjacent tokens through an MLP. (Tang et al., 2022) uses a circular shifting operator. However, the position invariance is violated because positional information is required in the decision of which tokens are included in the neighborhood. The aggregation of local information itself is done via a (relative) position-specific MLP. Global interactions are modeled only through the inclusion of enough layers or through a hierarchical layout (Yu et al., 2022a; Guo et al., 2022).

For vision tasks, it can be useful to exploit the fact that 2D images consist of two axes. Tatsunami and Taki (2022) make use of this fact by integrating a respective inductive bias. (Tu et al., 2022) achieve linear complexity by applying a gMLP (Liu et al., 2021) to only a single axis.

A global receptive field in MLP-based models is achieved through token mixing and a weighted summation of the inputs, similar to self-attention. This allows for interaction between tokens. Liu et al. (2021) propose the model gMLP, where the mixing weights are determined by a fixed learnable interaction matrix between positions. However, this comes at the cost of violating position-invariance, size adaptivity, and dynamicity. Dy-

naMixer (Wang et al., 2022) enables dynamicity by estimating the mixing weights from the concatenation of the inputs via a linear layer. This is efficient due to a dimensionality reduction step, but the concatenation still implies position dependence and fixed-sized inputs. (Lee-Thorp et al., 2022) proposes the model FNet to use static Fourier transformations to model token interactions. This model made significant improvements in computation cost, although the functions lack learnability and are position-dependent.

B.1.3 Hypernetworks

A hypernetwork uses a network to generate the weights for another, often larger, network (Ha et al., 2017).

Tay et al. (2021) leveraged task-conditioned hypernetworks for the GLUE benchmark. They achieved paralleled performance to the state-of-the-art at the time, whilst being more parameter efficient. Karimi Mahabadi et al. (2021) applied hypernetworks to Transformers to allow for parameter sharing in multitask learning. Their results showed parameter efficiencies and improved out-of-domain generation. Zhmoginov et al. (2022) combine hypernetworks and transformers in the vision domain for a few shot generalization. LambdaNets are strongly related to our work, as they generate linear functions from context, in a similar capacity to a Hypernetwork (Bello, 2021). Their model is similar to the standard attention mechanism where the weights of three matrices Q, K, V are learned. In contrast, HyperMixer uses the inputs to create non-linear transformations by generating an MLP. Features are combined based on their locations - a comparison can be found in Appendix B.5.

Combining MLPMixer and hypernetworks allows for an efficient and simple MLP-based model to have all the necessary inductive biases for NLP. The MLPMixer provides a simple token interaction backbone. By deploying hypernetworks to build the weights of the token mixing MLP, the missing inductive biases of position invariance and size adaptation are obtained.

B.2 Experimental Details

B.2.1 General Information

Implementation We implemented all models within the same general framework based on PyTorch (Paszke et al., 2019)[1]. For tokenization, we use the pre-trained tokenizer from BERT-Base (Devlin et al., 2019). Data sets are downloaded directly from HuggingFace

[1]An implementation of the layer is available at https://github.com/idiap/hypermixing

Data sets (Lhoest et al., 2021). As such, they are directly downloaded by our training script. We apply no further preprocessing.

For computing expected validation performance, we use the public implementation by Dodge et al. (2019).

We run our experiments on single-GPU servers available to us as part of a computation grid, ranging between GeForce GTX Titan X and RTX 3090. Apart from Transformers on SNLI and MNLI, which take about 4 hours on slower GPUs, all experiments finished within 3 hours.

Hyperparameters We provide CSV files detailing all parameters of every run alongside their results in the supplementary material, ensuring the reproducibility of our study. Note that the computation environment (e.g., type of GPU) might lead to small differences.

B.2.2 Peak Performance

To ensure a fair comparison, we aim to compare models of approximately the same number of parameters (≈11 M parameters). All models have 6 layers with token embedding size $d = 256$ and hidden size $d' = 512$. For MLPMixer and gMLP we set the size of the token mixing modules to $N = 250$ and $N = 100$, respectively. These lengths are chosen to match the number of parameters of the other models (11 M). The hidden layer size is set to 512 in all models. We use dropout at the input to each layer with a probability of 0.1. For all models, including the ablations, we first tune the learning rate of Adam (Kingma and Ba, 2015) using a logarithmically spaced grid of 7 values $\alpha \in \{0.001, 0.0005, 0.0002, 0.0001, 0.00005, 0.00002, 0.00001\}$ on the validation set. For our baselines, we then evaluate 10 different seeds and report the mean accuracy and standard deviation on the validation set. On the test set, we only report the results of the model yielding the best results on the validation set, as the GLUE benchmark (Wang et al., 2018) has a hidden test set with limited access. Ablations are evaluated on the validation set with a single seed.

B.2.3 Time per Example

Due to the lack of reliable software to measure FOPs in PyTorch, we calculate these numbers manually. Our process is described in Appendix B.4. For the measurement of wallclock time, we measured the time of 1,000 batches through a single layer of each token mixing module with $d = 256, d' = 512$ (as used in our experiments).

B.2.4 Visualizing Attention Patterns

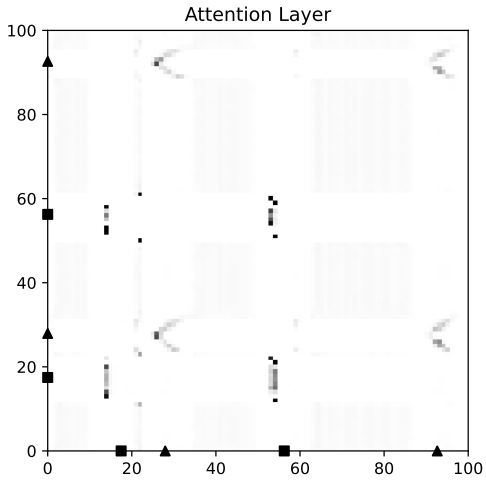
This section gives more detail about how we set up the synthetic example (Fleuret, 2019) for evaluating whether the different models were able to learn some attention-like transformation. We have a data set made of 1D sequences that contain two rectangular and two triangular shapes. Each of these shapes has a different height taken at random in the input sequence. The output sequence has the same shapes in the same positions, but the heights of triangular shapes should be the mean of the two triangular shapes in the input sequence. Similarly, the height of the rectangular shapes in the output sequence is the mean of the height of the two rectangular shapes in the input sequence.

The model should be able to see across the sequence and compute the mean of the two different shapes to succeed at the task. All the models considered for this task have a similar structure: they consist of a particular layer (MLPMixer, HyperMixer, or Attention) surrounded by two pairs of 1D-convolutional layers with kernels of size five and a symmetric zero-padding of size two ensuring that the output shape is constant. We made an ablation to ensure that this layer was mandatory by changing it with another similar 1D convolutional layer, which corresponds to `None` in the Figure 2.4b.

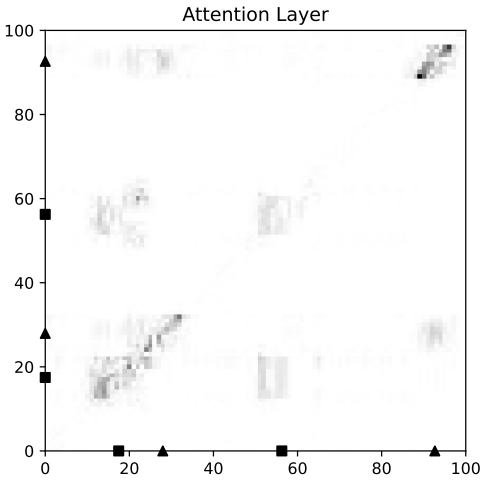
Before visualizing the pseudo-attention maps, all models were trained on 25,000 training examples. We use input-gradients (Simonyan et al., 2014) to evaluate whether models could « attend » to the different shapes. This method computes the gradient of the output sequence relative to the input sequence, giving the corresponding saliency map, which can then be recombined into a pseudo-attention matrix where the i-th column corresponds to the saliency maps of the i-th output token. A large value in the (i, j) entries of the pseudo-attention matrix means that the output token i strongly depends on the input j, making it comparable to an attention matrix. A representation of these pseudo-attention matrices is shown in Figure B.1a.

Figure B.1 shows the pseudo-attention of all models (except 'None') alongside the true attention weights of attention. First, it should be noted that pseudo-attention weights are a relatively blurry version of true attention weights, where high weights occur at positions that correspond to the same shape (comparing Figure B.1a to Figure B.1b). Second, we observe that the pseudo-attention weights of HyperMixer and attention (comparing Figure B.1d to Figure B.1b) are similar. This indicates that HyperMixer indeed learns an attention-like function. Third, MLPMixer also shows a similar pattern, but the relevant positions have weak connections (Figure B.1c). This confirms our finding that MLPMixer requires substantially more training data to learn strong connections.

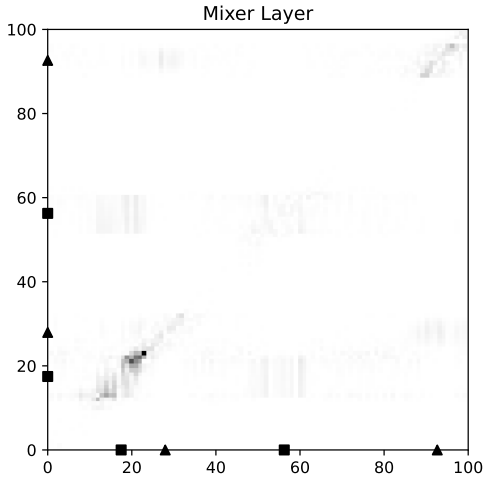
Figure B.1 represents the pseudo-attention matrices for the different models. We can notice that it indeed approximates the true attention matrix, shown in Figure B.1a, and that the model with no special layer cannot attend to the correct part of the sequence, as expected. Finally, we can see that the pseudo-attention of the Mixer layer is not as peaked



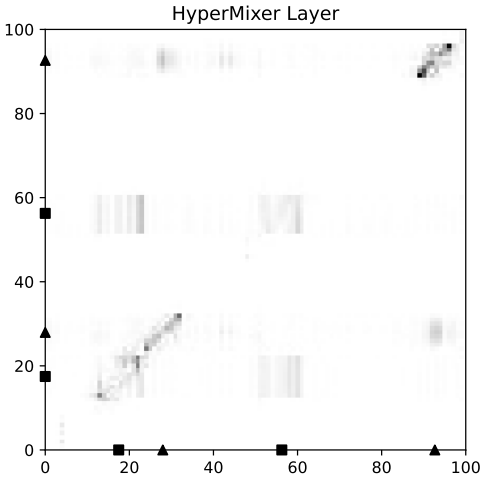
(a) True attention map of Attention



(b) Pseudo-attention map of Attention



(c) Pseudo-attention map of MLPMixer



(d) Pseudo-attention map of HyperMixer

Figure B.1: Results and (pseudo-)attention maps on the synthetic task (Fleuret, 2019).

	MNLI	SNLI	QQP	QNLI	SST	Params
	Baselines (accuracy / learning rate)					
FNet	59.6 / 5e-4	75.1 (.001)	79.7 (.001)	59.2 (5e-4)	80.4 (.001)	9.5 M
Linear Transformer	66.2 /.001	82.2 /0.001	81.7 /5e-4	61.1 /1e-4	80.7 /2e-4	11 M
Transformer	66.0 /2e-4	81.2 /2e-4	82.9 /2e-4	65.4 /5e-4	78.9 /5e-4	11 M
MLPMixer	64.2 /.001	80.5 /.001	83.6 /.001	68.7 /5e-5	82.3 /.001	11 M
gMLP	61.5 /.001	80.9 /2e-4	83.0 /5e-4	61.1 /5e-5	79.2 /1e-4	11 M
HyperMixer (tied)	66.5 /1e-4	81.8 /2e-4	85.4 /1e-4	77.5 /5e-5	81.3 /5e-4	11 M
	Ablations (accuracy / learning rate)					
Feature Mixing only	54.4 /.001	67.2 /5e-4	75.9 /.001	61.0 /.001	81.8 /5e-4	9 M
Token Mixing only	59.5 /2e-4	73.6 /2e-4	81.7 /2e-4	61.8 /2e-4	80.1 /5e-4	9 M
Shared Weight-Vector	53.7 /5e-4	68.1 /.001	83.0 /.001	66.4 /5e-5	80.5 /.001	9.5 M
HyperMixer (untied)	66.0 /.001	82.3 /.001	84.6 /.001	72.2 /5e-5	81.3 /.001	12 M

Table B.1: Best validation set results on natural language understanding tasks after tuning the learning rate on a grid.

	MNLI	SNLI	QQP	QNLI	SST	Params
FNet	58.8	75.2	78.4	59.0	80.2	9.5 M
Lin. Transformer	67.0	81.9	82.3	61.0	82.5	11 M
Transformer	64.9	81.1	82.1	67.1	77.7	11 M
MLPMixer	62.6	79.7	83.2	69.1	80.8	11 M
gMLP	62.9	79.9	82.3	60.0	78.5	11 M
HyperMixer (tied)	64.9	81.0	83.9	76.8	80.9	11 M

Table B.2: Test set results on natural language understanding tasks, when using the median seed.

as the one corresponding to the Attention or HyperMixer layer.

B.3 Further Results

B.3.1 Validation Set Results

In Table B.1, we show the best scores on the validation set that we obtained from the grid search (using a fixed seed), alongside the learning rate that yielded that score.

In Section 2.5.3, we reported the test set results of all models when using the best-performing seed. In Table B.2, we show test set results when using the median seed.

B.3.2 Ablations

We first describe the ablation models before we discuss their results (Table B.3).

	MNLI	SNLI	QQP	QNLI	SST	Params
	Validation set results (average accuracy / standard deviation over 10 seeds)					
Feature Mixing only	54.5 (0.25)	67.0 (0.14)	75.9 (0.06)	60.8 (0.42)	79.7 (0.64)	9 M
Token Mixing only	59.0 (0.79)	74.5 (5.53)	79.5 (4.63)	61.8 (1.29)	76.3 (4.94)	10 M
Shared Weight-Vector	57.1 (2.38)	74.3 (1.96)	82.9 (0.10)	65.9 (0.42)	79.8 (0.52)	9.5 M
HyperMixer (untied)	65.8 (0.46)	81.7 (0.30)	84.8 (0.23)	73.3 (0.53)	80.3 (0.35)	12 M
HyperMixer (tied)	66.2 (0.21)	81.9 (0.27)	85.6 (0.20)	78.0 (0.19)	80.7 (0.84)	11 M

Table B.3: Mean and standard deviation of HyperMixer ablations on the validation set.

Feature Mixing Only The most simplistic MLP architecture is one that doesn't use token mixing, i.e., the token mixing module is set to the identity function. The outputs at the last layer are aggregated via average pooling before being plugged into the linear classifier. This allows a baseline where the token interactions are not modeled. Therefore, this architecture serves as a control for how important token mixing is in any given task.

Token Mixing Only A simplistic single-layer MLP architecture ablation. This model consists of a variable dimension MLP where the weights are generated using a hypernetwork which only allows for location interaction. This model is included to argue that the best simple model requires both location and feature mixing to efficiently model textual inputs.

Shared Weight-Vector A simple way to obtain a variable size location-mixing MLP is by weight-sharing. Concretely, we use a single learnable weight vector $w_1 \in \mathbb{R}^{d'}$, which we copy N times to create a weight matrix $W_1 \in \mathbb{R}^{N \times d'}$. Analogously, we create W_2 from a separate vector w_2. Note that this baseline does not support dynamicity, as the weight vector is independent of the inputs. This baseline thus shows the importance of dynamicity in our model.

B.4 Comparison of #FOP

We want to compute the number of floating-point operations needed in self-attention vs. HyperMixing for a single example. Let N be the sequence length, d be the embedding size of each token, and d' the hidden dimension.

For simplicity, we will assume basic mathematical operators like exp, tanh, $\sqrt{x}$ and division to be equal to one floating operation. However, their actual cost is higher but depends on implementation and hardware.

B.4.1 Basic Building Blocks

We first compute the number of operations infrequently occurring in basic building blocks of neural networks.

Matrix Multiplication Multiplying matrix $A \in \mathbb{R}^{N \times d}$ $A \in \mathbb{R}^{d \times M}$ takes $2d(NM)$ operations, as $2d$ operations are needed for a single dot-product and there are NM entries in the resulting matrix.

Linear Layer Passing a single vector of size d through a linear layer without bias of size (d, d') is the multiplication of a single vector with a matrix, i.e., incurs $2dd'$ operations in total.

GELU GELU is usually approximated as

$$\text{GELU}(x) = 0.5x \left[1 + \tanh\left(\sqrt{2/\pi}(x + cx^3)\right)\right]$$

So in total, GELU is computed for every of the d features and every of the N vectors, meaning the GELU activation layer takes $9dN$ operations.

MLP (input = output size) Given hidden size d' and input/output size d, we have two linear layers of size (d, d') and (d', d), respectively, plus a GELU layer on d' dimensions, incurring $4dd' + 9d'$.

MLP (input /= output size) Given hidden size d', input size d and output size d'', we have two linear layers of sizes (d, d') and (d', d''), incurring $2dd' + 2d'd'' + 9d'$.

Softmax Softmax is applied over N values, each of which goes through an exp and a division by the normalization value. The normalization value requires N additions. So in total, the number of operations is $3N$.

B.4.2 HyperMixer

HyperNetwork (tied case) In the tied case, we have one MLP that generates an output for each vector, so the number of operations needed for an MLP of input and hidden size d and output sizes d': $N(2d^2 + 2dd' + 9d)$

Mixing MLP The mixing MLP has input and output size N and hidden size d', which is applied to each of the d embedding dimensions (i.e., after transposition), incurring $d(4d'N + 9')$ operations in total.

Total: The total number of operations in HyperMixer is $d(4Nd' + 9d') + N(2d^2 + 2d'd + 9d)$

B.4.3 Self-attention

Multi-head self-attention with h heads applies self-attention independently to each head consisting of vectors of size d/h, respectively.

Self-attention consists of

- 3 linear layers to transform queries, keys, and values: $6h(d/h)^2$
- h matrix multiplications with sizes $N(d/h)$, totalling $2h(d/h)N^2$ operations
- softmax: $3N$
- a weighted average for each of the inputs, consisting of $(2dN^2)$ operations.

In total: $6h(d/h)^2 + hN^2 2(d/h) + 3N + (2dN^2)$

B.5 Connection with Lambda Layers and Linear Transformer

We saw in Section 2.5.7 that HyperMixer was able to allow a form of attention without computing an attention matrix directly and thus scaling only linearly with the input length. In that regard, this method is similar to other methods such as (Bello, 2021) or (Katharopoulos et al., 2020). We will describe here the difference between these approaches and our method. Let us write the standard attention formula and the HyperMixer layer under the following form:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^T)\boldsymbol{V} \tag{B.1}$$

$$\text{HyperMixer}(\boldsymbol{X}) = \boldsymbol{W}_1 \sigma(\boldsymbol{W}_2^T \boldsymbol{X}) \tag{B.2}$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{N \times d'}$, $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ and $\boldsymbol{W}_1, \boldsymbol{W}_2$ are the weights generated by the hypernetwork.

We can notice that the two operations differ mainly in the non-linearity location and the uses of linear or non-linear projection of the input. Indeed, attention applies a non-linearity to $\boldsymbol{QK}^T$ and uses a linear projection of the input $(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ to construct the attention map. On the contrary, HyperMixer uses two linear mapping of the input $(\boldsymbol{W}_1, \boldsymbol{W}_2)$ and applies a non-linearity to $\boldsymbol{W}_2^T\boldsymbol{X}$, which is similar in a way to $\boldsymbol{K}^T\boldsymbol{V}$. The quadratic cost of the attention layer comes from the place of the non-linearity as it requires the explicit computation of $\boldsymbol{QK}^T \in \mathbb{R}^{N\times N}$ which is quadratic relative to the input size. Most of the strategies used to overcome this quadratic cost generally find a way of moving this non-linearity. This is the case of (Katharopoulos et al., 2020) which applies non-linearities ϕ independently to $\boldsymbol{Q}$ and $\boldsymbol{K}$ and (Bello, 2021) that applies softmax only to $\boldsymbol{K}$. In that regard, these two methods can be compared with HyperMixer as they all scale linearly with the input size due to the non-linearity location. Still, HyperMixer is conceptually different because it uses a non-linear transformation of the input and because it uses, in our opinion, a simpler and more understandable design entirely based on MLPs.

B.6 Ablations on Transformer Layout

While all Transformer layouts have a feature mixing and a token mixing component in each layer, the arrangement and connection of these components through skip connections and normalization layers remains an open question. The original Transformer (Vaswani et al., 2017) uses what is now known as the "post-norm" layout:

$$\mathbf{x}^1 = \text{LayerNorm}(\mathbf{x} + \text{token\_mixing}(\mathbf{x})) \tag{B.3}$$
$$\mathbf{x}^{out} = \text{LayerNorm}(\mathbf{x}^1 + \text{feature\_mixing}(\mathbf{x}^1)) \tag{B.4}$$

where $\mathbf{x} \in \mathbb{R}^{N\times d}$ is the input to the layer, and $\mathbf{x}^{out} \in \mathbb{R}^{N\times d}$ is the output of the layer.

(Wang et al., 2019) proposes the "pre-norm" layout:

$$\mathbf{x}^1 = \mathbf{x} + \text{token\_mixing}(\text{LayerNorm}(\mathbf{x})) \tag{B.5}$$
$$\mathbf{x}^{out} = \mathbf{x}^1 + \text{feature\_mixing}(\text{LayerNorm}(\mathbf{x}^1)) \tag{B.6}$$

(Bachlechner et al., 2021) proposes the "ReZero" normalization, which introduces a learnable scalar $\alpha \in \mathbb{R}$, initialized to zero:

$$\mathbf{x}^1 = \mathbf{x} + \alpha_1 \cdot \text{token\_mixing}(\mathbf{x}) \tag{B.7}$$
$$\mathbf{x}^{out} = \mathbf{x}^1 + \alpha_2 \cdot \text{feature\_mixing}(\mathbf{x}^1) \tag{B.8}$$

(Wang and Komatsuzaki, 2021) observe that a speed-up can be obtained by parallelizing the two components:

$$\mathbf{x}^{out} = \mathbf{x} + \text{token\_mixing}(\text{LayerNorm}(\mathbf{x})) + \text{feature\_mixing}(\text{LayerNorm}(\mathbf{x})) \tag{B.9}$$

.

Finally, (Chowdhery et al., 2023) call the following the "standard serialized" formulation:

$$\mathbf{x}^{1} = \mathbf{x} + \text{token\_mixing}(\text{LayerNorm}(\mathbf{x})) \tag{B.10}$$
$$\mathbf{x}^{out} = \mathbf{x} + \text{feature\_mixing}(\text{LayerNorm}(\mathbf{x}^{\mathbf{1}})). \tag{B.11}$$

As Figure 2.1 shows this is the model we have fixed for all previous experiments.

In the following, we combine each of the presented layouts with self-attention and HyperMixing, respectively. Since we noticed early that the training with HyperMixing is not stable with some of the layouts, we also experimented with adding two different kinds of normalization to HyperMixer: layer normalization applied after TM-MLP, as shown in Listing 2.1, and length normalization. For the latter, we simply scale the generated weight matrices by $\frac{1}{M}$, where M is the number of keys. The intuition is that this keeps the magnitude of activations in the hidden layer of TM-MLP approximately the same across different input lengths.

Results Table B.4 shows the best validation set results after tuning the learning rate using a logarithmically spaced grid of 7 values $\alpha \in \{0.001, 0.0005, 0.0002, 0.0001, 0.00005, 0.00002, 0.00001\}$.

The results show that self-attention is nearly insensitive to the type of layout, as all models except for ReZero attain an accuracy of 76-77% on average. In contrast, HyperMixer without normalization performs substantially worse with prenorm, ReZero, and the parallel layout. Length normalization mitigates this problem to some degree, but the addition of layer normalization yields the overall best results, where all models achieve between 77 and 78% of accuracy on average. We, therefore, recommend adding layer normalization by default when using HyperMixing in a new context.

	MNLI	SNLI	QQP	QNLI	SST	Average
			Multi-head self-attention			
serialized	65.71	80.88	82.99	69.67	79.70	75.79
post-norm	66.13	81.70	84.31	71.54	79.70	76.68
pre-norm	66.60	80.59	82.96	73.13	80.73	76.80
ReZero	56.83	70.85	77.72	63.44	78.10	69.39
parallel	66.30	81.46	83.12	71.55	79.70	76.43
			HyperMixing (no normalization)			
serialized	66.18	81.63	85.59	78.4	81.65	78.69
post-norm	62.59	79.49	82.37	76.75	80.39	76.32
pre-norm	56.62	78.49	82.88	64.18	81.08	72.65
ReZero	35.45	33.82	63.18	49.46	49.08	46.20
parallel	60.37	79.71	83.62	65.24	80.16	73.82
			HyperMixing (length normalization)			
serialized	65.91	81.27	85.27	77.80	81.88	78.43
post-norm	62.67	79.46	82.61	76.53	80.39	76.33
pre-norm	64.83	80.71	84.41	76.31	81.65	77.58
ReZero	35.45	33.82	63.18	70.31	54.13	51.38
parallel	65.37	81.12	84.44	76.77	80.28	77.60
			HyperMixing (layer normalization)			
serialized	66.47	81.36	85.74	77.72	80.50	78.36
post-norm	64.26	80.05	83.81	76.62	80.85	77.12
pre-norm	64.72	81.05	83.81	76.11	81.54	77.45
ReZero	65.64	80.74	84.45	74.41	81.08	77.26
parallel	65.49	80.59	84.43	76.53	81.65	77.74

Table B.4: Best validation set results on natural language understanding tasks after tuning the learning rate on a grid.

Appendix C

Appendix - Chapter 3

C.1 Data set descriptions

C.1.1 Wind nowcasting

The wind nowcasting experiment uses the same data set as (Pannatier et al., 2022). It is available at: https:/zenodo.org/record/5074237. It is an important task for ATC as it involves the crucial prediction of high-altitude winds using real-time data transmitted by airplanes. This prediction is essential for ensuring efficient airspace management the airspace. It is important to note that in the vertical direction (z), the majority of wind measurements are typically taken at elevated altitudes, specifically between 4,000 and 12,000 meters. Regarding the horizontal dimensions, the airspace extends over 600 km from north to south and approximately 500 km from east to west. As ground-based measurements are not accessible at these heights, our dependence lies on the measurements gathered directly from airplanes. As planes do not record the wind in the z direction, the input space corresponds to wind speed in the x, y plane $\mathbb{I} \subseteq \mathbb{R}^2$, and the output space is the wind speed measured later, $\mathbb{O} \subseteq \mathbb{R}^2$. Here the underlying space is European airspace represented as $\mathbb{X} \subseteq \mathbb{R}^3$ In this particular setup, the models need to extrapolate in time, based on a set of the last measurements.

Airplanes measure wind speed with a sampling frequency of four seconds. We split the data set into time slices, where each slice contains one minute of data, such that each time slice contains between 50 and 1500 data points. We tried different time intervals but noticed that having a longer time slice did not improve the quality of the forecasts. The objective for the different models is to output a prediction of the wind at different query points 30 minutes later. We evaluate performance using the Root Mean Square Error (RMSE) metric, as in previous work.

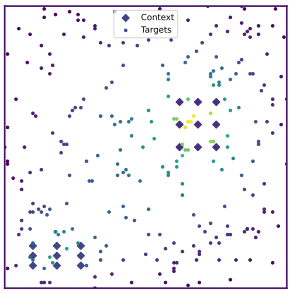


Figure C.1: A sample of the Poisson equation data set (Alet et al., 2019). • represents the targets, The context values are comprised of points on the boundaries, and points on the sources and sink are represented by ♦ which corresponds to their thermal coefficient.

C.1.2 Poisson Equation

This experiment uses the same data set as (Alet et al., 2019). It is available at https:/github.com/FerranAlet/graph_element_networks/tree/master/data. The Poisson equation models the heat diffusions over the unit square with sources represented by $\phi(x, y) \in \mathbb{R}$ and fixed boundary conditions $\omega \in \mathbb{R}$. The equation is given by:

$$\begin{cases} \Delta\phi(x, y) = \psi(x, y) & \text{if } (x, y) \in (0, 1)^2 \\ \phi(x, y) = \omega & \text{if } (x, y) \in \partial[0, 1]^2 \end{cases} \tag{C.1}$$

It should be noted that the boundary constant and sources function $\phi(x, y)$ conditions can change for each sample.

The data set used in (Alet et al., 2019) uses three dimensions for the context values $\mathbf{y}_c \in \mathbb{I} \subseteq \mathbb{R}^3$: either sources values $\phi(x, y) = \mu$ inside the domain encoded as $\mathbf{x}_c, \mathbf{y}_c = (x, y), (\mu, 0, 0)$ or boundary conditions $\phi(x, y) = \omega$ on the boundaries, encoded as $\mathbf{x}_c, \mathbf{y}_c = (x, y), (0, \omega, 1)$. The target space is one-dimensional $\mathbf{y}_t \in \mathbb{O} \subseteq \mathbb{R}$ and corresponds to the solution of the Poisson equation at that point. A sample is depicted in Figure C.1.

We evaluate all models for three different sizes on this particular setup and find that MSA with the novel encoding scheme outperforms other models by a significant margin as can be seen in Table 3.2. We initialized GEN with a 7×7 regularly initialized grid on the $[0, 1]^2$ as in the original work (Alet et al., 2019).

C.1.3 Navier-Stokes Equation

Similarly to the darcy flow task, we adapted the Navier-Stokes equation as described in (Li et al., 2021) to an irregular setup.

The Navier-Stokes equation describes a real fluid and is described with the following PDE :

$$\begin{cases} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) = \nu \Delta w(x,t) + f(x), x \in (0,1)^2 & \text{if } t \in (0,T] \\ \nabla \cdot u(x,t) = 0, x \in (0,1)^2 & \text{if } t \in [0,T] \\ w(x,0) = w_0(x) & \text{if } x \in (0,1)^2 \end{cases} \tag{C.2}$$

For more detailed information regarding the notation, please refer to the original work by Li et al. (Li et al., 2021), specifically section 5.3.

In this study, our objective is to forecast the future state of the vorticity quantity, specifically 50 timesteps ahead, based on measurements of vorticity at different spatial locations. This approach differs from the original work, where the model was provided with ten initial vorticity grids and required to predict the complete system evolution.

To create our data set, we subsampled the complete data set, which consisted of the evolution of the Navier-Stokes equation solved by a numerical solver. The full data set had dimensions of $1000 \times 1024 \times 1024$.

For our purposes, we selected pairs of slices that were separated by 50 timesteps and performed spatial subsampling. We took 64 context measurements and 256 targets as our subsampled data points. This experiment adapts the data set available in (Li et al., 2021). It is available at https:/github.com/neural-operator/fourier_neural_operator and can be processed with the code given to rearrange it in the irregular setup. We present a sample of the data set in Figure C.3.

C.1.4 Darcy Flow

We evaluated the performance of various models in solving the Darcy Flow equation on the unit grid with null boundary conditions, as described in (Li et al., 2021). Specifically, we aimed to predict the value of the function u, given the diffusion function a, with the two related implicitly through the PDE given by:

$$\begin{cases} -\nabla \cdot (a(x) \nabla u(x)) = 1 & \text{if } x \in (0,1)^2 \\ u(x) = 0 & \text{if } x \in \partial [0,1]^2. \end{cases} \tag{C.3}$$

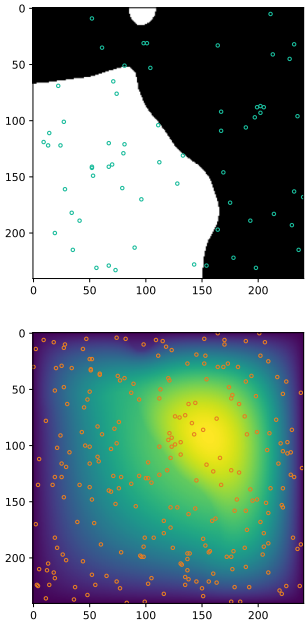


Figure C.2: A sample of the Darcy Flow data set (Li et al., 2021). Blue dots correspond to the context set and orange dots to the targets.

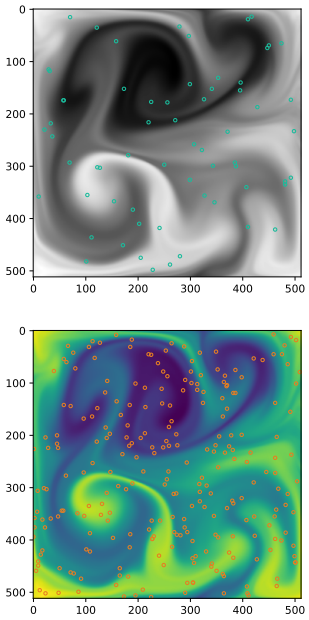


Figure C.3: A sample of the Navier-Stokes data set (Li et al., 2021). Blue dots correspond to the context set and orange dots to the targets.

The data set used in our study is adapted from that used in (Li et al., 2021), and originally generated by a traditional high-resolution PDE solver. The data set consists of a 1024 $\times$ 1024 grid, which was subsampled uniformly at random and arranged into context-target pairs. The context is comprised of the evaluations of the diffusion coefficient: $(\mathbf{x}_c, \mathbf{y}_c) = ((x, y), a(x, y))$, and the target is the solution of the Darcy Flow equation at a position, $(\mathbf{x}_t, \mathbf{y}_t) = ((x, y), u(x, y))$.

The results are presented in Table 3.2, they are coherent with the rest of the experiment and show that the MSA and TFS models outperform all competing models. We used the same initialization for GEN as for the Poisson Equation.

This experiment adapts the data set available in (Li et al., 2021). It is available at https:/github.com/neural-operator/fourier_neural_operator and can be processed with the code provided to rearrange it in the irregular setup.

C.1.5 Two-Days Weather Forecasting

In this task, we want to evaluate our models on the task of two-day weather forecasting based on irregularly sampled data in space. The requested data collection description focuses on climate reanalysis from the ERA5 data set, which is available through the

Copernicus Climate Data Store (CDS). The data set contains various parameters related to wind, surface pressure, temperature, and cloud cover.

To access the data, please visit the following URL: https:/cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels?tab=form.

We selected the following variables:

- 10m u-component of wind: This refers to the eastward wind component at a height of 10 meters above the surface.
- 10m v-component of wind: This represents the northward wind component at a height of 10 meters above the surface.
- 100m u-component of wind: This denotes the eastward wind component at a height of 100 meters above the surface.
- 100m v-component of wind: This signifies the northward wind component at a height of 100 meters above the surface.
- Surface pressure: This represents the atmospheric pressure at the Earth's surface.
- 2m temperature: This indicates the air temperature at a height of 2 meters above the surface.
- Total cloud cover: This refers to the fraction of the sky covered by clouds. The data collection includes measurements for all times of the day and all days available in the data set. However, for training purposes, we selected the data from the year 2000. For validation, we use the data from the year 2010, specifically the months of January and September.

Next, we proceeded to extract the corresponding grib files. To create our data set, we paired time slices that had a time difference of two days, equivalent to 48 timesteps since ERA5 data has a one-hour resolution over the whole world.

For the context slice, we performed subsampling at 1024 different positions and retained all seven variables mentioned earlier. As for the target slice, we subsampled it at 1024 different positions and kept only the two components of the wind at 100m.

C.2 Detailed results

Here, we present a breakdown of the detailed results for the different models with and without shared positional encoding and presented in terms of the specific metrics reported

Data set	dim(𝕏)	dim(𝕀)	dim(𝕆)	# Train. points	# Val. points	URL
Wind Nowcasting	3	2	2	26.956.857	927.906	(Pannatier et al., 2022) Data set page
Poisson Equation	2	3	1	409.600	102.400	(Alet et al., 2019) Github Repository
Navier-Stokes Equation	2	1	1	48.883.20	11.673.600	(Li et al., 2021) Github Repository
Darcy Flow Equation	2	1	1	409.600	102.400	(Li et al., 2021) Github Repository
ERA5	2	7	2	8.648.704	2.107.392	(Hersbach et al., 2023) Data set Store
Random	2	1	1	640.000	64.000	Randomly generated
Sine	2	1	1	640.000	64.000	Randomly generated

Table C.1: Description of the different data sets used in this study.

in their original study. In Table 3.2 of the main paper, we provide the results for GEN and CNP based on their original implementations, which do not involve sharing the position encoding and we translated all metrics to RMSE. For TFS and MSA, we include the results obtained with shared position encoding. To ensure comprehensiveness and to differentiate the enhanced performance attributed to the latent representation from that resulting from the improved data encoding approach, we also apply the shared encoding technique to GEN and CNP.

In most cases, we observe that employing a shared encoding for position tends to improve the performance of all models, except for CNP where it seems to decrease the overall performance in some cases. Additionally, it seems that MSA surpasses its competitors in the majority of instances, consistently exhibiting superior performance across various data sizes. Moreover, when the shared encoding for positions fails to achieve the optimal performance, the model's performance is generally comparable to that without shared encoding, often falling within a standard deviation range. We also give in Table C.3 a detailed breakdown of the results for the training set.

C.3 Specific Aspects of Our Approach Compared to Existing Works

In our research, we address a distinct challenge that sets our work apart from conventional mesh-based simulators (Pfaff et al., 2020; Janny et al., 2023; Li et al., 2021). Our focus is on extrapolating sets of sparse and variably-positioned measurements, a context markedly different from the consistent mesh structures employed in traditional approaches. Unlike mesh-based models where values are accessible at fixed points on an irregular mesh, our method tackles scenarios where the quantity and locations of measurements can change dynamically from one data set to another. This unique aspect of our work necessitates a

Table C.2: Results of the High-Altitude Wind Nowcasting, Poisson, Navier-Stokes, and Darcy Flow equation and the weather forecasting task. Each model ran for respectively 10, 2000, 1000, 100, and 100 epochs on an NVIDIA GeForce GTX 1080 Ti. The low number of epochs for wind nowcasting is due to the amount of data which is considerably larger than in the other experiments. The standard deviation is computed over 3 runs. We choose the configuration of the models so that every model has a comparable number of parameters. We underlying the best models for each size and we put in bold the best model overall. In this table, we kept the results in the same metric as originally reported. The first half of the table corresponds to the case without the novel encoding scheme, as opposed to the second one. The last part of the table reports the results of GeoFNO another interesting baseline, which we discussed in detail in Appendix C.5. In Table 3.2 of the main paper, we recomputed all metrics in terms of Root Mean Square Error (RMSE).

	Size	Wind Nowcasting RMSE (↓)	Poisson Equation MSE (↓)	Navier-Stokes Equation MSE (↓)	Darcy Flow Equation RMSE (↓) E−4	ERA 5 MSE (↓)
Default encoding for positions						
CNP	5k	10.19 ± 0.21	.130 ± .0134	.492 ± .0033	9.65 ± 0.47	4.48 ± 0.01
	20k	10.11 ± 0.20	.105 ± .0012	.452 ± .0015	8.71 ± 0.11	4.44 ± 0.00
	100k	10.20 ± 0.26	.105 ± .0024	.430 ± .0010	8.17 ± 0.06	4.43 ± 0.01
GEN	5k	9.84 ± 2.92	.017 ± .0011	.365 ± .0012	9.25 ± 0.18	4.47 ± 0.02
	20k	9.24 ± 0.35	.020 ± .0054	.359 ± .0007	8.74 ± 0.09	4.44 ± 0.00
	100k	9.23 ± 0.44	.048 ± .0389	.355 ± .0006	8.66 ± 0.09	4.46 ± 0.00
TFS (Ours)	5k	8.75 ± 0.14	.055 ± .0248	.367 ± .0033	7.46 ± 0.55	4.46 ± 0.00
	20k	8.70 ± 0.06	.017 ± .0014	.357 ± .0019	6.69 ± 0.14	4.38 ± 0.01
	100k	8.67 ± 0.07	.016 ± .0045	.350 ± .0011	7.47 ± 0.24	4.42 ± 0.01
MSA (Ours)	5k	8.40 ± 0.10	.048 ± .0186	.361 ± .0054	7.74 ± 0.48	4.44 ± 0.02
	20k	8.47 ± 0.12	.047 ± .0107	**.346 ± .0042**	6.76 ± 0.10	4.39 ± 0.01
	100k	8.98 ± 0.22	.030 ± .0081	.350 ± .0015	7.33 ± 0.22	4.41 ± 0.01
Sharing encoding for positions						
CNP	5k	10.33 ± 0.19	.165 ± .0023	.706 ± 0.0003	17.26 ± 0.04	4.53 ± 0.01
	20k	10.12 ± 0.21	.160 ± .0011	.706 ± 0.0001	17.06 ± 0.06	4.47 ± 0.00
	100k	10.26 ± 0.24	.158 ± .0006	.705 ± 0.0001	16.87 ± 0.02	4.43 ± 0.01
GEN	5k	8.79 ± 0.23	.017 ± .0023	.372 ± 0.0008	6.83 ± 0.17	4.54 ± 0.01
	20k	9.08 ± 0.47	.018 ± .0036	.366 ± 0.0000	6.75 ± 0.05	4.50 ± 0.01
	100k	9.07 ± 0.00	.019 ± .0008	.361 ± 0.0000	7.19 ± 0.12	4.50 ± 0.01
TFS (Ours)	5k	8.30 ± 0.03	.025 ± .0121	.365 ± 0.0026	7.58 ± 0.84	4.53 ± 0.01
	20k	8.20 ± 0.04	.010 ± .0013	.355 ± 0.0009	**6.64 ± 0.14**	4.45 ± 0.01
	100k	8.38 ± 0.13	.035 ± .0054	.349 ± 0.0015	7.25 ± 0.19	4.41 ± 0.00
MSA (Ours)	5k	8.07 ± 0.11	.014 ± .0014	.357 ± 0.0013	7.51 ± 0.61	4.52 ± 0.03
	20k	**7.98 ± 0.03**	**.007 ± .0004**	.347 ± 0.0016	6.74 ± 0.38	4.44 ± 0.01
	100k	8.18 ± 0.14	.010 ± .0017	.347 ± 0.0008	6.97 ± 0.24	4.40 ± 0.01
Different encoding scheme						
GeoFNO	5k	11.38 ± 0.34	.030 ± .0086	.532 ± .0064	8.46 ± 0.11	4.43 ± 0.02
	20k	11.31 ± 0.40	.035 ± .0073	.473 ± .0103	8.16 ± 0.21	4.41 ± 0.01
	100k	11.11 ± 0.47	.035 ± .0226	.429 ± .0060	7.92 ± 0.10	4.41 ± 0.01
	1.5M	11.28 ± 0.40	.014 ± .0035	.402 ± .0029	8.22 ± 0.02	4.42 ± 0.01

Table C.3: Corresponding training metrics

	Size	Wind Nowcasting RMSE (↓)	Poisson Equation MSE (↓)	Navier-Stokes Equation MSE (↓)	Darcy Flow Equation RMSE (↓) E−4	ERA 5 MSE (↓)
			Default encoding for positions			
CNP	5k	11.94 ± 0.78	.127 ± .0179	.485 ± .0033	9.02 ± 0.54	4.33 ± 0.01
	20k	10.19 ± 1.83	.084 ± .0057	.438 ± .0015	7.88 ± 0.16	4.25 ± 0.01
	100k	10.17 ± 1.24	.021 ± .0040	.398 ± .0010	6.93 ± 0.08	4.17 ± 0.01
GEN	5k	11.02 ± 3.19	.011 ± .0006	.362 ± .0012	8.49 ± 0.17	4.32 ± 0.02
	20k	9.98 ± 0.76	.006 ± .0006	.354 ± .0007	7.76 ± 0.11	4.24 ± 0.01
	100k	9.56 ± 0.21	.011 ± .0122	.339 ± .0006	7.01 ± 0.26	4.10 ± 0.01
TFS (Ours)	5k	9.86 ± 0.21	.022 ± .0021	.365 ± .0033	6.50 ± 0.60	4.29 ± 0.02
	20k	9.69 ± 0.38	.005 ± .0008	.353 ± .0019	5.48 ± 0.10	4.08 ± 0.03
	100k	9.55 ± 0.19	.001 ± .0001	.314 ± .0011	4.45 ± 0.17	3.78 ± 0.01
MSA (Ours)	5k	8.86 ± 0.01	.022 ± .0062	.359 ± .0057	6.91 ± 0.50	4.28 ± 0.02
	20k	7.94 ± 0.03	.005 ± .0012	.341 ± .0042	5.72 ± 0.11	4.13 ± 0.01
	100k	6.67 ± 0.02	.000 ± .0001	.310 ± .0015	4.91 ± 0.17	4.19 ± 0.01
			Sharing encoding for positions			
CNP	5k	10.41 ± 0.03	.154 ± .0010	.703 ± .0001	16.94 ± 0.06	4.38 ± 0.00
	20k	9.48 ± 0.02	.133 ± .0004	.700 ± .0001	16.68 ± 0.10	4.30 ± 0.01
	100k	8.60 ± 0.05	.107 ± .0008	.696 ± .0001	16.36 ± 0.02	4.23 ± 0.01
GEN	5k	10.04 ± 0.13	.005 ± .0004	.369 ± .0007	5.93 ± 0.15	4.42 ± 0.01
	20k	9.75 ± 0.09	.003 ± .0007	.362 ± .0002	5.65 ± 0.06	4.36 ± 0.01
	100k	9.84 ± 0.02	.001 ± .0000	.345 ± .0005	4.97 ± 0.22	4.34 ± 0.02
TFS (Ours)	5k	8.78 ± 0.02	.009 ± .0015	.364 ± .0024	6.74 ± 0.93	4.41 ± 0.01
	20k	7.94 ± 0.03	.002 ± .0002	.352 ± .0012	5.56 ± 0.10	4.26 ± 0.01
	100k	6.57 ± 0.02	.000 ± .0000	.312 ± .0006	4.51 ± 0.24	4.09 ± 0.01
MSA (Ours)	5k	8.54 ± 0.03	.008 ± .0024	.355 ± .0013	6.74 ± 0.61	4.38 ± 0.03
	20k	7.73 ± 0.02	.002 ± .0003	.342 ± .0014	5.79 ± 0.34	4.25 ± 0.01
	100k	6.58 ± 0.05	.000 ± .0000	.310 ± .0008	4.76 ± 0.12	4.14 ± 0.02
			Different encoding scheme			
GeoFNO	5k	11.68 ± 0.30	.008 ± .0010	.526 ± .0070	7.56 ± 0.13	4.18 ± 0.02
	20k	11.56 ± 0.20	.003 ± .0005	.467 ± .0100	7.02 ± 0.32	4.07 ± 0.03
	100k	10.72 ± 0.27	.001 ± .0001	.421 ± .0068	5.87 ± 0.45	3.75 ± 0.07
	1.5M	10.30 ± 0.93	.000 ± .0002	.363 ± .0192	2.56 ± 0.51	3.00 ± 0.26

departure from the conventional mesh framework. Mesh-based simulators, while effective in their domains, are not equipped to handle the sporadic and non-uniform nature of our data. For instance, in applications such as airspace monitoring, data points represent measurements taken only where planes are present, leaving vast areas without data. Our methodology, therefore, diverges fundamentally from mesh-based techniques, as it requires innovative handling of sparse and irregular data inputs, rather than relying on a fixed, uniform mesh structure.

Furthermore, our approach markedly differs from trajectory prediction models (Yuan et al., 2021; Nayakanti et al., 2023; Girgis et al., 2022). While these models excel in forecasting future points along established trajectories, our work diverges in both intent and application. Our primary concern is not the sequential prediction of trajectories but rather the interpretation and forecasting of phenomena in a spatially and temporally sparse environment. In the context of wind nowcasting, for example, our model excels at processing limited and scattered data points from multiple trajectories to generate a comprehensive forecast. This capability to assimilate sparse measurements from varied locations and synthesize them into a coherent prediction sets our work apart from traditional trajectory-focused models. These models, such as Wayformer (Nayakanti et al., 2023) and Agentformer (Yuan et al., 2021), are primarily designed for time series forecasting and trajectory completion.

C.4 Comparison with Hypermixer

In Chapter 2, we presented the hypermixer layer, which is a drop-in, linear-cost replacement for the self-attention layer in transformers. It has the advantage of handling variable-sized inputs and outputs, and its low cost makes it particularly appealing when the number of input measurements is large.

We evaluated hypermixer on the wind nowcasting task and present the results in table C.4. In all cases, MSA outperforms Hypermixer. We hypothesize that the difference in performance arises from the same issue faced by other baselines: the additional mixing of information in the hypermixer layer creates a bottleneck that hinders training.

In this application, we have seen that increasing the size of the context did not have a significant impact on the performance of the model. This is likely due to the fact that the only measurements that are relevant are the ones that are sampled the latest, and older measurements are not useful for the prediction. In this case, the number of context points does not go beyond 2000, which is perfectly manageable by the MSA layer.

If the number of measurements were to increase, we could use the hypermixer layer to handle the additionnal load. We present such a scenario in Figure C.4. We see that the

Table C.4: Comparison between Hypermixer and MSA on the wind nowcasting task. In that task, MSA outperforms Hypermixer in all cases. We hypothesize that the reason is that the additional mixing of information in the hypermixer layer is detrimental in this case.

	Size	RMSE (↓)
MSA (Ours)	5k	8.07 ± 0.11
	20k	**7.98 ± 0.03**
	100k	8.18 ± 0.14
Hypermixer	5k	9.49 ± 0.09
	20k	9.50 ± 3.15
	100k	7.74 ± 0.08

MSA layer is able to handle up to 5000 context points, and even up to 50k context points with the memory-efficient kernel, but the time per step starts to increase significantly. Hypermixer is able to handle 50k context points in a reasonable amount of time. We show a linear and a quadratic curve to compare the scaling of the different methods. At the time of writing, the flash attention kernel for a full-attention forward and backward pass is not available in the PyTorch library; such a kernel might be able to close the gap between the two methods.

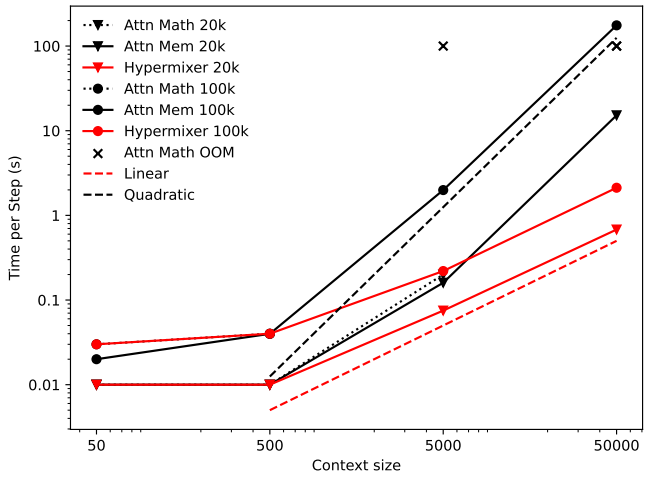


Figure C.4: Comparison between MSA and Hypermixer for different numbers of context points. Attn Math represents the Math PyTorch kernel; it becomes prohibitively expensive for a model with 100k parameters when the context has more than 5000 measurements. Attn Mem represents the Memory-Efficient PyTorch kernel; it manages to handle even 50k context points, but its time per step starts to increase significantly. Hypermixer is able to handle 50k context points in a reasonable amount of time. We show a linear and a quadratic curve to compare the scaling of the different methods.

C.5 Comparison with GeoFNO

The Fourier Neural Operator (FNO) family (Li et al., 2021) and its variant GeoFNO (Li et al., 2023b), have emerged as a significant tool in resolving Partial Differential Equations (PDEs) across various benchmarks. Even though the GeoFNO model was specifically designed for irregular meshes, setting it apart from other models evaluated, it is recognized for its adept handling of irregular geometries and flexibility in adapting to different types of Partial Differential Equations (PDEs), making it a strong baseline for our experiments.

We evaluated GeoFNO on our specific data set and problem. We tried both the original model configuration, consisting of 1.5 million parameters, and its scaled-down versions with 5k, 20k, and 100k parameters, as we found the 1.5 million parameters to be over-parametrized. The outcomes of these tests can be found in Table C.2.

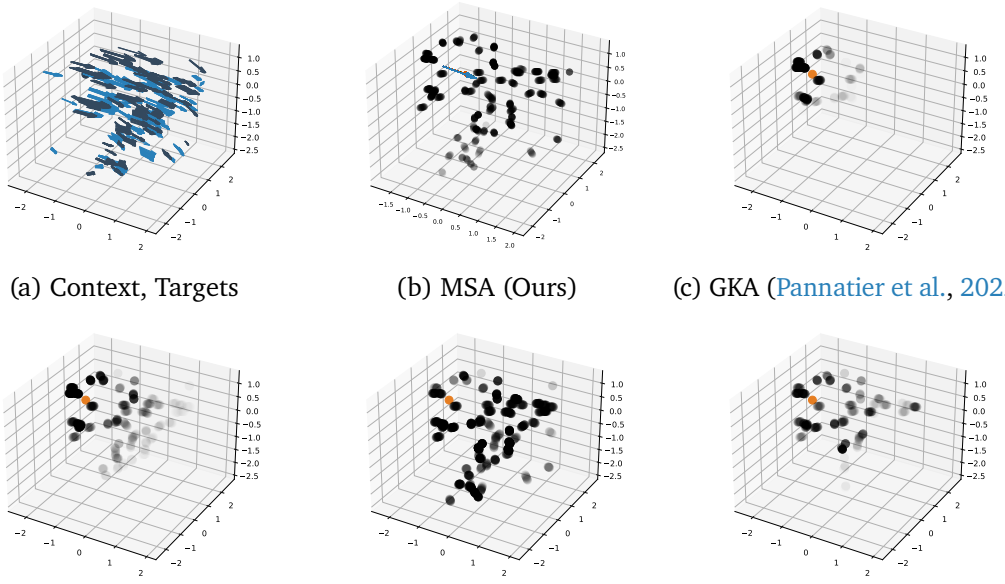
Despite GeoFNO's overall robustness, we found that it encounters challenges in this particular setup. We hypothesize that the model struggled with sparse irregular data, as it needs to learn to map a set of data to a grid before applying a regular FNO layer. Sparsity, in this context, might lead to several parts of the grid being empty thus leading to information loss or dispersion. Secondly, in this experiment, models need to handle variable measurement position, which changes from batch to batch, which is more complicated than learning a mapping from an irregular but fixed mesh. These factors potentially contributed to the performance inconsistencies observed in our experiments. For instance, in the Poisson setup, where data points were more evenly distributed, GeoFNO's performance occasionally matched that of the MSA model in some randomized trial runs, as indicated in [Figure C.7e]. However, in more demanding scenarios like wind nowcasting, GeoFNO consistently underperformed in comparison to both MSA and GEN models.

C.6 Attention matrix for Wind Nowcasting

For all models, we can plot the norm of the input gradients to see how changing a given value would impact the prediction [Figure C.5]. We see that Transformers seem to find a trade-off between taking into account the neighboring nodes and the global context.

C.7 Comparison with Graph Kernel Averaging

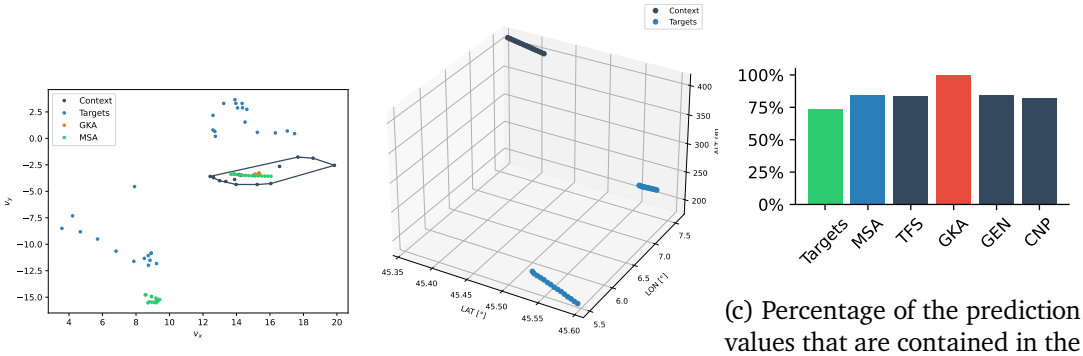
Smart averaging strategies such as GKA are limited by design to the range of the values in their context. However, these baselines are useful as they are not prone to overfitting and



(a) Context, Targets (b) MSA (Ours) (c) GKA (Pannatier et al., 2022)

(d) GEN (Alet et al., 2019) (e) CNP (Garnelo et al., 2018) (f) TFS (Ours)

Figure C.5: Displaying the importance given to points in the context to do the prediction for the different models for a given query point (orange). We use the norm of the input gradients for that purpose and highlight the context values that have the largest gradient with respect to the output. The opacity of the dots corresponds to the relative magnitude of the input gradients compared to other points in the context.



(a) Context, targets, and forecast in wind speed space.

(b) Context and targets in the 3D space.

(c) Percentage of the prediction values that are contained in the convex hull of the measurement in the context in percent.

Figure C.6: Context and targets represented in wind speed and 3D space. Both the context and targets correspond to 1-minute slices of data. The targets capture wind measurements recorded 30 minutes following the context data. Additionally, the forecasts generated by the MSA and GKA models are depicted. Notably, the GKA model is confined to predicting within the convex hull of the context values, which is visually highlighted in grey. Conversely, the MSA model is unrestricted by this limitation and can make predictions beyond the convex hull of the context values.

in the case of time series forecasting (if the underlying variable has some persistence), the last values would in general be the best predictor of the next value. We hypothesize that over sufficiently short horizons, these methods would achieve performance similar to that from more complicated models, but when the prediction horizon rises, this design limitation would become more pronounced.

Since, for extrinsic forecasting, target points need not lie in the convex hull of context points, the greater flexibility of MSA and TFS can make correct forecasts that are impossible with GKA. Figure C.6a demonstrates this fact, with a plot of the target variable, the context points, and the context convex hull in wind speed space.

But greater capacity means also more failure modes. As we saw in the metrics of Table 3.3 of the main paper, MSA and TFS are the only models to outperform GKA whereas other models, even if they are more flexible, fail to beat this baseline. On average 27% of the predictions were outside the convex hull of the context. In Figure C.6c we analyze the percentage of measurements outside the convex hull produced by the different models. We can see that GKA is limited as 100% of the measurements lie within the convex hull whereas the other models can compensate and predict values outside it.

C.8 Decoder and conditioning function

In the main paper, we presented a comparison of the key distinctions between our attention-based architectures. However, it could be that the performance differences observed could be attributed to minor variations in the model architecture design. For the sake of completeness, in this section, we provide experimental results to address this aspect by comparing the remaining differences between GEN and TFS. Specifically, we focus on analyzing the impact of the number of decoder layers and the conditioning function. It is important to note that this analysis does not apply to MSA, as it does not possess an encoder-decoder structure.

GEN and TFS differ not only in their latent representation but also on two other points: (1) the way that they access the encoded information in the decoder and (2) how much computing power is used in the decoder stack.

In all three models, we tried using either a distance-based function or cross-attention to combine the target position with the encoded latents. CNP concatenates the same context vector to all target queries, which we model as equivalent to averaging the latents based on the distance in the case that there is only one latent.

Specifically, each latent is associated with a position $\mathbf{x}_l$ in the underlying space with corresponding value $\mathbf{l}_y$. Conditioning is made by averaging the latent features based on their distance with the query $\mathbf{x}_t$. Its formula is given by:

$$z = \sum_{\mathbf{l} \in \text{latents}} r(\text{dist}(\mathbf{x}_l, \mathbf{x}_t))\mathbf{l}_y \tag{C.4}$$

Where $r : \mathbb{R} \to [0, 1]$ is a function that maps distances to weights, and $\mathbf{x}_l$ represents the position associated with the latent nodes with value $\mathbf{l}_y$. This vector z is then concatenated to the target position $\mathbf{x}_t$ and fed to the decoder which is an MLP in this case.

GEN uses this distance-based aggregation function by default whereas TFS uses cross-attention. We tried using cross-attention for CNP and GEN and using the same distance-based aggregating function for TFS using the distance between context position and target position as a reference for this scheme. For GEN and CNP using cross-attention gives approximately the same results, but using distance-based conditioning for TFS hurts the performance significantly.

We also ran ablations that created hybrid cases for both transformers and GEN and concluded that adding decoding layers helped TFS reach better performance. Adding layers to the GEN decoder drastically reduces model performance, which seems to be coherent with the fact that GEN was shown to be more prone to overfitting in the results

Table C.5: Results of the ablation of the conditioning function used to combine the latents with the target position. We adapted the architecture so that they used both a distance-based conditioning function, which combines the query position with a weighted average of the nearest latents and standard cross-attention. We show in italics hybrid architectures where we had to switch the default conditioning method. For GEN and CNP replacing the conditioning method does not impact the performance, but for TFS switching to a distance-based conditioning function impacts the performance drastically.

	Distance-based	**Cross-attention**
CNP	.115 ± .015	*.119 ± .014*
GEN	.024 ± .004	*.022 ± .006*
TFS	*.042 ± .023*	**.020 ± .011**

Table C.6: Results of the decoder-layer ablation. One difference between TFS and GEN is that by default TFS uses multiple layers in its decoder whereas GEN uses one, and delegates all processing to the encoder. We see that having multiple decoder layers helps the transformer whereas it impacts the performance of the GEN.

	# Decoder layers		
	1	**2**	**4**
GEN	.021 ± .002	.040 ± .012	.106 ± .004
TFS	.020 ± .011	.019 ± .005	**.013 ± .001**

section.

Finally, we want to highlight that MSA outperforms all the models presented in this section by a large margin Table 3.2.

C.9 Wind nowcasting metrics

This section defines the metrics used for wind nowcasting.

Root Mean Square Error (RMSE) It takes the square root of the square distance between the prediction and the output. It has the advantage of having the same units as the forecasted values.

$$\text{RMSE}(\hat{t}, t) = \sqrt{\frac{\sum_k^N (\hat{t}_k - t_k)^2}{N}} \tag{C.5}$$

Mean Absolute Error for angle (angle MAE) It is interesting and sometimes more insightful to decompose the errors made by the models in their angular and norm components. This is the role of this metric and the following.

$$\text{angle MAE}(\hat{t}, t) = ||(\alpha(\hat{t}) - \alpha(t))||_{L_1} \tag{C.6}$$

where $\alpha\left(\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}\right) = \arctan(y, x) * \frac{360}{2\pi}$

Mean Absolute Error for norm (norm MAE) Following the explanation of the last paragraph:

$$\text{norm MAE}(\hat{t}, t) = \sum_k | \, || \, \hat{t}_k \, ||_2 - || \, t_k \, ||_2 \, | \tag{C.7}$$

Relative Bias (rel BIAS) x,y Additionally, we used weather metrics as in (Ghiggi et al., 2019). The relative bias measures if the considered model under or overestimates one quantity. It is defined as:

$$\text{rel BIAS}(\hat{t}, t) = \frac{\text{mean}(\hat{t}_k - t_k)}{\text{mean}(\hat{t}_k)} \tag{C.8}$$

Ratio of standard deviation (rSTD) The ratios of standard deviation indicate whether the dispersion of the output of the model matches the target distribution. It has an optimal value of 1.

$$\text{rSTD}(\hat{t}, t) = \frac{\text{std}(\hat{t})}{\text{std}(t)} \tag{C.9}$$

NSE The last domain metric used is the Nash-Sutcliffe efficiency (NSE), which compares the error of the model with the average of the target data. A score of 1 is ideal and a negative score indicates that the model was worse than the average prediction on average.

$$\text{NSE}(\hat{t}, t) = 1 - \frac{\text{MSE}(\hat{t}, t)}{\text{MSE}(t, \text{mean(t)})} \tag{C.10}$$

C.10 Influence of the tightness of the measurements on the performances

To evaluate the influence of the tightness of the measurements on the results we designed the following experiment: We start with a pair of context $(\mathbf{x}_c, \mathbf{y}_c)$ and target points $(\mathbf{x}_t, \mathbf{y}_t)$. We pick one point from the context and restrict the context to a small number of measurements close to this point. Then, we calculate the error made by the model when it is given only this restricted context and rank them by their distance to the context's center. As the dimensions (longitude, latitude, altitude) differ, we first normalized the data along each axis. We repeat this process and average the results.

Table C.7: Influence of the tightness of the measurements on the performance. We restrict the context to measurements that are close to the query point and then measure the error as a function of the distance from this query point averaged over the whole data set. We estimate the standard deviation using three random seeds.

Normalized Dist	MSA	TFS
0.5–1.0	5.56 ± 0.49	7.73 ± 0.37
1.0–1.5	6.62 ± 0.35	7.35 ± 0.53
1.5–2.0	7.57 ± 0.43	7.87 ± 0.36
2.0–2.5	9.52 ± 0.47	9.48 ± 0.66
2.5–end	10.35 ± 0.49	10.53 ± 0.61

We see that the error increases with the distance to the context. Now in practice, the context is not restricted and contains points all over the space (so the mean minimum normalized distance to the target points is usually often below 1.0)

We assessed general uncertainty by training the model using various random seeds, resulting in an ensemble of models from which we can determine the standard deviation and add it to the results table of the previous experiment.

C.11 Influence of the target positions

In this section, we assess the impact of the target position on the training of the MSA model.

The context data for the Poisson equation is heavily prescribed (where context points belonged to the source or sink or boundary conditions), while for the Darcy Flow equation, we constructed a similarly-shaped data set by subsampling the data set from (Alet et al., 2019) to create the context and targets. Thus, the Darcy flow test problem gives an ideal testbed for examining the influence of target positions on the training. To examine the

impact of the context position, we designed an additional experiment where we sampled the context from the bottom left quadrant of the unit cube and the targets from the upper right quadrants of the unit cube. We conduct our analysis for small (5k parameters) and large (100k parameters) models. We report the test loss as a function of the percentage of the context from the upper-right quadrants (where all targets are located, the remaining from the bottom-left quadrant).

Table C.8: Influence of the target position on the training. We report the test error for the MSA model at two different sizes. We devised an experiment in which we extracted the context from the bottom left quadrant of the unit cube, while the targets were sampled from the upper right quadrants of the same unit cube.

	0%	**2%**	**25%**	**50%**	**100%**
MSA 5k	0.14	0.15	0.14	0.09	0.03
MSA 100k	0.14	0.14	0.13	0.09	0.03

In the first column (0% of the data from the first quadrant), we see that the MSA model struggled to learn when the context and targets are disjoint. The error improves with greater overlaps between context and target, with p = 100% corresponding to the case in the paper (albeit on one-quarter of the data). Similar dynamics hold for both model sizes. We believe that this is due to the Darcy Flow simulation being highly location-dependent, as can be seen in some examples from the data set Figure C.2.

Regarding the training performance, we noticed no substantial difference in the total time to a solution, they all took approximately 1000 epochs in all cases. Larger models were always able to overfit the data.

Table C.9: Influence of the target position on the training. We report the train error for the MSA model at two different sizes. Both models were able to overfit the data.

	0%	**2%**	**25%**	**50%**	**100%**
MSA 5k	0.07	0.07	0.07	0.05	0.02
MSA 100k	0.00	0.00	0.00	0.00	0.00

C.12 Different Message Passing Scheme

We try here different message-passing schemes even one using attention, to help us demonstrate the fact that the whole family of models that encodes the space as a single graph suffers from the same bottlenecking effect. Here are the results of the wind nowcasting task:

Although certain message-passing schemes enhance the performance of the GEN model,

Table C.10: Performance of GEN with different message passing schemes for the wind nowcasting task. Various message-passing schemes, including those with attention, were explored using PyTorch Geometric (Fey and Lenssen, 2019). The best-performing GEN model with the default message-passing scheme is indicated with a reference line. While certain schemes showed improvements over GEN's performance, the overall performance remained relatively consistent and was still outperformed by MSA. For additional information on the different message-passing schemes and related references, please refer to the PyTorch Geometric documentation.

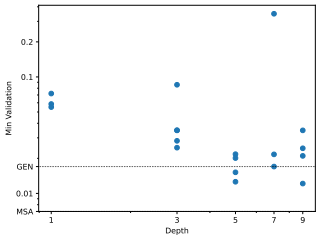
	Score
TransformerConv	9.28
GATv2Conv	9.34
GeneralConv	9.37
GATConv	9.77
SAGEConv	9.78
ARMAConv	9.84
TAGConv	9.87
SGConv	9.95
SuperGATConv	10.10
GCNConv	10.13
LEConv	10.61
GENConv	23.98

it still falls short of the performance achieved by MSA. We attribute this difference to the bottlenecking effect caused by the latent graph. For additional information on the various message-passing schemes and relevant references, please consult the PyTorch Geometric documentation available at https:/pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers.

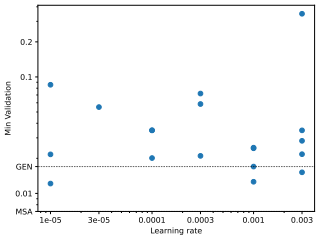
C.13 Hyperparameter sensitivity analysis

In this section, we analyze the robustness of MSA to hyperparameter changes. We focus on the number of layers [Figure C.7a] and the hidden dimension [Figure C.7c] in the case of the Poisson equation. In the Table C.2, we report three different sizes of the different models, and we tried to keep them as balanced as possible. The exact configurations of the models can be found in the config folder of the code, however, for the sake of completeness, we grid-searched over the mentioned hyperparameters while randomly sampling the learning rate.

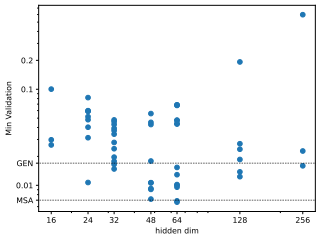
We also did it for the width hyperparameter of GeoFNO, in the case of the Poisson equation [Figure C.7e] and the wind nowcasting experiment [Figure C.7g]. In the case of the Poisson



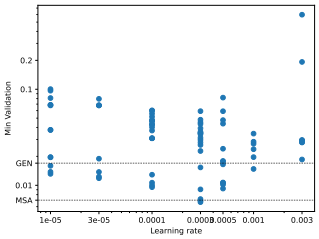
(a) Impact of the number of layers on the performance of MSA in the Poisson Experiment.



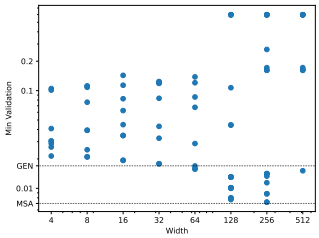
(b) Validation loss in function of the learning rate when varying the number of layers for MSA in the Poisson Experiment.



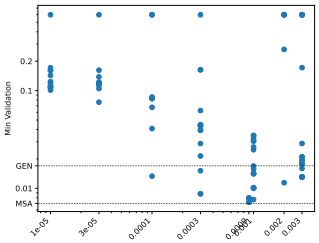
(c) Impact of the hidden dimension on the performance of MSA in the Poisson Experiment.



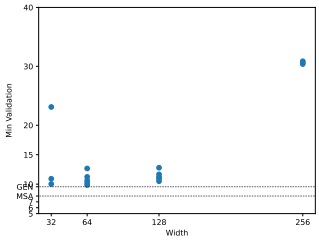
(d) Validation loss in function of the learning rate when varying the hidden dimension for MSA in the Poisson Experiment.



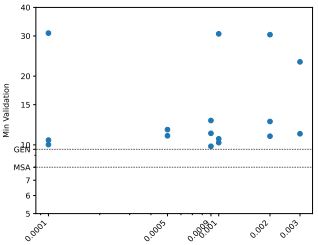
(e) Impact of the width hyperparameter on the performance of GeoFNO for the Poisson equation.



(f) Validation loss in function of the learning rate when varying the width hyperparameter for the Poisson equation.



(g) Impact of the width hyperparameter on the performance of GeoFNO for the wind nowcasting task.



(h) Validation loss in function of the learning rate when varying the width hyperparameter for the wind nowcasting task.

Figure C.7: Different hyperparameter sensitivity analysis for the Poisson equation and the wind nowcasting task.

experiment, the randomized trial helps in finding the best hyperparameter, leading to a model that is on par with MSA, we suspect that in this setup, the context positions are more evenly distributed in space, which might lead to a setup that does not suffer as much from the flaws listed in Appendix C.5, however in the case of the wind experiment, the randomized experiment did not help as much and the model is still lacking behind MSA and GEN. However, it might be possible to GeoFNO to this special case of data set irregularly sampled in space is an interesting avenue for future work.

Appendix D

Appendix - Chapter 4

D.1 Shuffled sequences are harder to learn

Using a random order in the training of a transformer often leads to a harder task to learn. To showcase this property, we designed the following task. The model is asked to model a lazy random walk that starts from a multinomial distribution made of four Dirac distributions (at 100, 120, 130, 140), with equal probabilities. The random walk evolves with the following law:

$$p(X_t|X_{t-1}) = \begin{cases} \frac{2}{5} & \text{if } |X_t - X_{t-1}| = 1 \\ \frac{1}{5} & \text{if } X_t = X_{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{D.1}$$

With $p(X_0 = 100) = p(X_0 = 120) = p(X_0 = 130) = p(X_0 = 140) = 1/4$.

In this setup predicting future tokens conditioned on past tokens is easy and is given by:

$$X_{t_3} \sim \mathcal{N}(X_{t_2}, (t_3 - t)\sqrt{0.8}) \tag{D.2}$$

with $t_1 < t_2 < t_3$

note: $\mathbb{E}(X^2) = 2 * 0.4 * 1 + 0.2 * 0 = 0.8$

The exact probability mass function (pmf) can also be computed:

$$p(X_{t_3} = x|X_{t_2}) = \sum_{o=0}^{n} \binom{n}{o} \binom{n-o}{u} p_u^u p_d^d p_o^o \tag{D.3}$$

with $n = t_3 - t_2$, $u = n - o + x - X_{t_2}$, $d = n - o - u$.

Predicting tokens conditioned on future information is much harder in that setup as the model has to take into account the initial multinomial distribution and keep track of all the different possibilities.

In such cases the exact pmf is given by :

$$p(X_{t_1} = x | X_{t_2}) = \frac{(\sum_{i \in \mathcal{S}} P(i, x, t_1)) P(x, X_{t_2}, t_2 - t_1)}{\sum_{i \in S} P(i, X_{t_2}, t_2)} \tag{D.4}$$

Where $P(a, b, n) = \#\text{paths } a \to b \text{ in } n \text{ steps } = \sum_{o=0}^{n} \binom{n}{o} \binom{n-o}{u}$. with $u = n - o + b - a$.

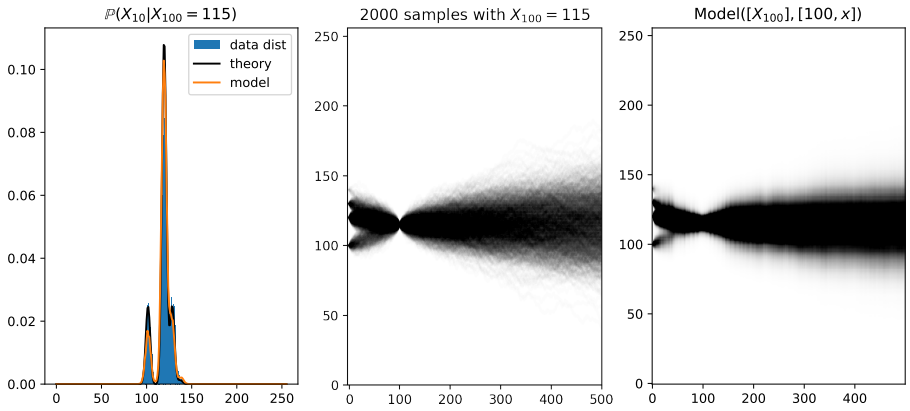


Figure D.1: Random walks at different altitudes results. (**Left**) Comparison with the theoretical density at position 10, is hard to model as the model needs to take into account the number of possible paths from each starting point. (**Middle**). Empirical density (**Right**). Learned density. The model is capable of modeling the density in two directions when conditioned on one position even if one order is more complicated than the other.

When trained with a left-to-right order, the model quickly learns the lazy random walk distribution but is never asked to learn more than that. However, when trained with a random order, the model has to learn to compute the more complex multinomial distribution, which is a much harder task as the model has to learn the global statistics of the sequence.

D.1.1 Learning a harder problem: Chain rule of probability

Shuffling the order of the tokens in a sequence has the effect of creating a harder problem for the model. But if the model has access to enough data and to learn the distribution perfectly then shuffling should not change the results. If the model learned the final

distribution perfectly, the decoding order should not matter in theory as stated by the chain rule, indeed

$$\begin{aligned} p(X_1 = x_1, X_2 = x_2, \ldots, X_T = x_T) = \\ & p(X_1 = x_1) p(X_2 = x_2 | X_1 = x_1) \ldots \\ & p(X_T = x_t | X_1 = x_1, \ldots X_{T-1} = x_{t-1}) \\ = p(X_{\sigma_1} = x_{\sigma_1}) & p(X_{\sigma_2} = x_{\sigma_2} | X_{\sigma_1} = x_{\sigma_1}) \ldots \\ & p(X_{\sigma_T} = x_{\sigma_t} | X_{\sigma_1} = x_{\sigma_1}, \ldots X_{\sigma_{T-1}} = x_{\sigma_{t-1}}) \end{aligned} \quad \text{(D.5)}$$

For any permutation $\sigma = \begin{pmatrix} 1 & 2 & \cdots & T \\ \sigma_1 & \sigma_2 & \cdots & \sigma_T \end{pmatrix}$.

This is not what we see in practice, as models are not perfect and data sets are finite, so the impact of the order has its importance.

Another demonstration of this effect is the chain of thought step-by-step prompting strategy (Wei et al., 2022; Kojima et al., 2022) of large language models. Where adding a few tokens between the question and the answer increases the working memory of the transformer, which seems to help the models make fewer mistakes.

D.2 Estimation of number of steps for burst sampling

D.2.1 Deterministic case and lucky samplings

In this section, we describe the fully deterministic case and the lucky samplings. In the deterministic case, and assuming a perfect model, the model will accept the sequence in one step. Indeed, if the model already knows the sequence, the probability distribution won't change when it is seen under a different order, therefore the rejection sampling ratio will always be one and the model will accept the sequence in one step.

In is the same for lucky samplings, again assuming a perfect model and a valid sample, the probability of a token given a specific order can only be higher than the probability of the token conditioned only on the prompt, as the tokens seen can only remove valid possibilities. Hence $q(x) > p(x)$ and the model will accept the sequence in one step.

D.2.2 Product Data set: Number of passes needed to generate a valid sequence

In this section, we give the number of steps required to generate a valid sequence in the case of the product data set using our rejection sampling scheme under a perfect model. The product data set is made of a sequence of length 100 with two classes (0,1) given by a Bernoulli law with p = 10%. At the beginning of the generation, a perfect model will predict $p(x_t = 1|\{\}) = 0.1, \forall t \in \{1, \dots, 100\}$. As all tokens are independent, the conditioning of the model will not change the probability of the next token. Therefore, the model will accept all tokens in the first step of the rejection sampling scheme. Our scheme should generate a valid sample for this distribution in at most 1 step.

D.2.3 Step Data set: Number of passes needed to generate a valid sequence

In this section, we give the number of steps required to generate a valid sequence in the case of the step data set using our rejection sampling scheme under a perfect model. The step data set is made of a sequence of length 100 of classes $\{0, 1\}$ The sequence is 0 everywhere except on a step of ones of length 10 placed randomly in the sequence. At the beginning of the generation, a perfect model cannot do better than predicting $p(x_t = 1|\{\}) = 0.1, \forall t \in \{1, \dots, 100\}$.

Sampling with this law will lead to a sequence with 10 ones on average. The number of accepted tokens in the first round of rejection sampling will then depend on the random order sampled. The first token given by the order will always be accepted as $q(x_t|\{\}) = p(x_t|\{\})$. Now the next tokens are either compatible with the tokens already accepted or not. If the next token is compatible with the already accepted tokens, then it will be accepted with probability 1 as $p(x_t|\{\}) = q(x_t|x_{\text{accepted}})$. If it is not compatible, then it will be rejected as $q(x_t|\{x_{\text{accepted}}\}) = 0$ and the rejection sampling stops rejecting all the remaining tokens.

Now the reason for non-compatibility in the step data set is either that

a. the model has already seen a 1, and sees a second 1 that is too far away,

b. that the model sees a one and then a zero nearby which would mean the end of the sequence which is not long enough, or

c. that it has seen only 0s, and no places are remaining for 1s. Therefore it has to reject the zero that would lead to an empty sequence. 1. In case, a-b. the model accepts a one, in the first step, and in case c. the start of the step is confined between the two last zeros where there are still enough places. 2. Once a token from the step is accepted, or there is only one place for a step, the signal is deterministic and a perfect model will 3. sample a

complete step, Our scheme should generate a valid sample for this distribution in at most 3 steps.

D.2.4 Permutation

When wanting to generate a valid permutation per burst assuming a perfect model, when given a partially completed permutation, a perfect model cannot do better than predicting a uniform law on the remaining tokens.

There is a well-known formula for the number of different classes that one gets when sampling with a uniform law, it is given by:

$$\mathbb{E}[\#\text{ of incoherent tokens}] \tag{D.6}$$
$$= T - \mathbb{E}[\#\text{ classes being sampled}] \tag{D.7}$$
$$= (T-1)^T / T^{T-1} \tag{D.8}$$

More generally, the probability of sampling exactly k classes is given by

$$p(\text{sampling exactly } k \text{ classes}) = \binom{T}{k} k! \left\{ {T \atop k} \right\} / T^T \tag{D.9}$$
$$\tag{D.10}$$

Where $\left\{ {T \atop k} \right\}$ is the Stirling number of the second kind.

Having a closed formula for the expected number of steps is hard, but we can estimate it numerically using Equation (D.9). For a sequence of length 100, we find that the expected number of steps is 5.2 with a standard deviation of 0.6. Which gives us a lower bound for the number of steps required to generate a valid permutation.

Note that in this specific case, for our rejection sampling scheme to be able to generate a valid permutation in this number of steps, the model needs to validate the partial sequence under an order which is such that each unique classes are seen first because once a class is repeated the model will have to reject the rest of the sequence.

D.3 Caching Scheme for Burst rejection sampling

Autoregressive transformers usually rely on a KV-caching mechanism to go from a $O(N^3)$ to a $O(N^2)$ cost of generation. this KV cache can be adapted to generate a sample by burst. If we already have accepted a set of tokens T_1 and we have a KV cache K_1, V_1 that stores the keys and values for this set of tokens, and we want to generate a prediction for all the remaining set of T_2 tokens in parallel.

The output of the transformer corresponding to the remaining set of tokens is

$$V_2' = \text{softmax}\left(\; Q_{T_2} K_1^T \;\middle|\; D \; \right) \begin{pmatrix} V_1 \\ \hline V_2 \end{pmatrix} \tag{D.11}$$

With $D = \text{diag}(q_i \cdot v_i)$ This formula parallelizes easily. We give the corresponding code in pytorch.

```
def burst(self, kv_cache, q, k, v):
    k1, v1 = kv_cache.get()

    att1 = q @ k1.transpose(-2, -1)
    # no masking on KV cache, tokens can see them all

    att2 = (q * k).sum(-1, keepdim=True)
    att = torch.cat((att1, att2), dim=-1)
    att *= (1.0 / math.sqrt(k.size(-1)))
    att = F.softmax(att, dim=-1)
    att = self.attn_drop(att)

    att1 = att[..., :-1]
    att2 = att[..., [-1]]

    # Works even if the cache is empty
    y = att1 @ v1 + att2 * v
    y = rearrange(y, "b h t e -> b t (h e)")

    y = self.resid_drop(self.proj(y))

    return y
```

Listing D.1: KV Caching scheme of burst sampling

D.4 Additional experiments for the vertical rate forecasting

We present here additional results concerning the vertical rate modeling experiment Table D.1. We report the Mean Square Error (MSE) compared to the ground truth when prompted by partially completed trajectories. The idea is the see the effect of partial left-to-right conditioning on the quality of the generated sequences. 0%, 10%, and 50% denote the percentage of the actual flight that is given as a prompt to the model. In this setup, we see that σ-GPT outperforms models trained causally for a generation. As the data set size is small, we noticed that models trained in a left-to-right manner were suffering from a repetition problem that arises when the modeling capabilities of the models are insufficient. As the prompting is given to the model in left-to-right order, we see that causal models can outperform random models when prompted with half of the actual sequence.

We see as well that diffusion models usually outperform autoregressive models in this task. However, they need to be retrained to be able to generate sequences conditioned on a partial sequence. We see that the autoregressive models can be used in a more flexible way as they can be used to generate sequences conditioned on a partial sequence without retraining.

Table D.1: This table presents the Mean Square Error (MSE) results for the climbing rate forecasting task. The MSE is calculated on the entire generated sequence, conditioned on different points during the climb: the start (0%), early stage (10% of the climb), midway (50% of the climb) and in the middle of the first climb. For each validation sequence, we generated 20 sequences autoregressively using the model, following three different schemes: causal scheme, random scheme, and binary search tree order. We also report the performance of diffusion models, for comparison, but we only report their performance for the entire sequence as they need to be retrained to be able to generate sequences conditioned on a partial sequence.

Size	Method	0%	10%	50%	Mid Climb
Small	**Fractal**	145.7 ± 2.6	1278.9 ± 161.9	1466.6 ± 172.2	623.6 ± 84.4
	Causal	274.8 ± 70.7	179.9 ± 48.0	**52.9 ± 12.7**	46.1 ± 10.0
	Random	141.4 ± 4.1	123.8 ± 6.0	63.8 ± 1.2	40.1 ± 2.7
	Diffusion	**105.94 ± 1.3**	-	-	-
Base	**Fractal**	158.1 ± 5.9	2523.5 ± 13.9	1623.5 ± 283.5	884.4 ± 15.7
	Causal	424.7 ± 10.9	287.7 ± 0.5	70.0 ± 5.7	61.5 ± 6.8
	Random	146.3 ± 6.4	119.3 ± 7.8	65.3 ± 5.1	41.6 ± 0.7
	Diffusion	107.96 ± 2.4	-	-	-
Tiny	**Fractal**	152.3 ± 8.7	2553.6 ± 168.3	1508.8 ± 220.3	972.8 ± 44.9
	Causal	290.9 ± 13.0	224.0 ± 1.8	58.9 ± 1.6	53.9 ± 5.7
	Random	129.5 ± 23.7	**108.1 ± 6.1**	72.6 ± 4.7	**36.7 ± 0.8**
	Diffusion	123.28 ± 6.9	-	-	-

Bibliography

M. Abadi, P. Barham, J. Chen, et al. Tensorflow: A system for large-scale machine learning, 2016. (url). 12, 18

AI@Meta. Meta Llama 3. 2024. (url). (code). 3

S. Albanie. Euclidean Distance Matrix Trick. *Visual Geometry Group*, 2019. (pdf). 20

F. Alet, A. K. Jeewajee, M. B. Villalonga, et al. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, pages 212–222. PMLR, 2019. (pdf). (code). 47, 48, 49, 51, 54, 108, 112, 118, 123

F. Alet, D. Doblar, A. Zhou, et al. Noether Networks: meta-learning useful conserved quantities. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. (pdf). (url). (code). 85

J. Austin, D. D. Johnson, J. Ho, et al. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021. (pdf). 71, 80

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. (pdf). 31, 32

T. Bachlechner, B. P. Majumder, H. Mao, et al. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pages 1352–1361. PMLR, 2021. (pdf). 31, 103

D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. (pdf). 3, 4

I. Bakhtourine, T. Bannerman, S. Bourton, et al. Leveraging Generative AI in Europe: The Opportunities and Challenges. *McKinsey & Company*, 2023. (url). Accessed: 2024-07-16. 1

P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction.

Nature, 525(7567):47–55, 2015. ISSN 0028-0836. doi: 10.1038/nature14956. (pdf). 11

I. Bello. LambdaNetworks: Modeling long-range Interactions without Attention. In *International Conference on Learning Representations*, 2021. (pdf). (url). 34, 95, 102, 103

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. (pdf). 94

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. (pdf). 12, 16

J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. (pdf). (url). 93

J. Betker, G. Goh, L. Jing, et al. Improving image generation with better captions, 2023. (pdf). (url). 1

R. Bommasani, D. A. Hudson, E. Adeli, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. (pdf). 28

S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, sep 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. (pdf). (url). 35

T. Brown, B. Mann, N. Ryder, et al. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, et al., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. (pdf). (url). 1, 4, 86

H. Chang, H. Zhang, L. Jiang, et al. MaskGIT: Masked Generative Image Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11315–11325, June 2022. (pdf). (code). 71, 80

H. Chang, H. Zhang, J. Barber, et al. Muse: Text-To-Image Generation via Masked Generative Transformers. In A. Krause, E. Brunskill, K. Cho, et al., editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 4055–4075. PMLR, 23–29 Jul 2023. (pdf). (url). 80

C. Chen, S. Borgeaud, G. Irving, et al. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. (pdf). 70

G. Chen, Y. Ding, and X. Shen. Sweet knn: An efficient knn on gpu through reconciliation between redundancy removal and regularity. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 621–632. IEEE, 2017. (pdf). 13

J. Chen, D. Tam, C. Raffel, et al. An empirical survey of data augmentation for limited data learning in nlp. *arXiv preprint arXiv:2106.07499*, 2021. (pdf). 93

R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. (pdf). 94

A. Chowdhery, S. Narang, J. Devlin, et al. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023. (pdf). (url). 104

G. Czibula, A. Mihai, and E. Mihuleţ. NowDeepN: An Ensemble of Deep Learning Models for Weather Nowcasting Based on Radar Products' Values Prediction. *Applied Sciences*, 11(1):125, 2021. (pdf). 13

T. Dao, D. Y. Fu, S. Ermon, et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. (pdf). (url). 60

T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. (pdf). (url). 60

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, jun 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. (pdf). (url). 1, 3, 28, 60, 95

J. Dodge, S. Gururangan, D. Card, et al. Show Your Work: Improved Reporting of Experimental Results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2185–2194, Hong Kong, China, nov 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1224. (pdf). (url). 38, 93, 96

J. Dodge, S. Gururangan, D. Card, et al. Expected Validation Performance and Estimation of a Random Variable's Maximum. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4066–4073, Punta Cana, Dominican Republic, nov 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.342. (pdf). (url). 38, 93

A. Dosovitskiy, L. Beyer, A. Kolesnikov, et al. An image is worth 16x16 words: Transformers

for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. (pdf). 1, 4, 50, 60

D. Drakulic, S. Michel, F. Mai, et al. BQ-NCO: Bisimulation Quotienting for Generalizable Neural Combinatorial Optimization. *arXiv preprint arXiv:2301.03313*, 2023. (pdf). 42

J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. (pdf). 30

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 (CVPR)*, pages 12873–12883, June 2021. (pdf). 5

C. Esteves, J. E. Slotine, and A. Makadia. Scaling Spherical CNNs. In A. Krause, E. Brunskill, K. Cho, et al., editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 9396–9411. PMLR, 2023. (pdf). (url). 48

M. Fey and J. E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. (pdf). 125

F. Fleuret. Attention Mechanisms. Deep Learning Course - Chapter 13.2, 2019. (pdf). (url). 39, 40, 97, 98

J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988. (pdf). 31

J. Friedman, T. Hastie, R. Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001. (pdf). 14

F. Fusco, D. Pascual, P. Staar, and D. Antognini. pNLP-Mixer: an Efficient all-MLP Architecture for Language. In S. Sitaram, B. Beigman Klebanov, and J. D. Williams, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 53–60, Toronto, Canada, jul 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-industry.6. (pdf). (url). 34

V. Garcia, E. Debreuve, F. Nielsen, and M. Barlaud. k-nearest neighbor search: fast gpu-based implementations and application to high-dimensional feature matching. *2010 IEEE International Conference on Image Processing*, pages 3757–3760, 2010. doi: 10.1109/icip.2010.5654017. (pdf). 13

M. Garnelo, D. Rosenbaum, C. Maddison, et al. Conditional Neural Processes. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713. PMLR, 10–15 Jul 2018. (pdf). (url). 47, 48, 49, 52, 56, 118

N. Geneva and N. Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022. (pdf). 48

G. Ghiggi, V. Humphrey, S. I. Seneviratne, and L. Gudmundsson. GRUN: an observation-based global gridded runoff dataset from 1902 to 2014. *Earth System Science Data*, 11 (4):1655–1674, 2019. doi: 10.5194/essd-11-1655-2019. (url). 122

A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, page 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606157. (pdf). 12, 16

R. Girgis, F. Golemo, F. Codevilla, et al. Latent Variable Sequential Set Transformers for Joint Multi-Agent Motion Prediction. In *International Conference on Learning Representations*, 2022. (pdf). (url). 48, 115

A. Gokaslan and V. Cohen. OpenWebText Corpus, 2019. (url). 72

O. Golovneva, Z. Allen-Zhu, J. Weston, and S. Sukhbaatar. Reverse Training to Nurse the Reversal Curse, 2024. (pdf). 80

A. Gu and T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2024. (pdf). (code). 3

I. Gulrajani and T. Hashimoto. Likelihood-Based Diffusion Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. (pdf). (url). 80

J. Guo, Y. Tang, K. Han, et al. Hire-MLP: Vision MLP via Hierarchical Rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 826–836, June 2022. (pdf). 94

J. K. Gupta and J. Brandstetter. Towards Multi-spatiotemporal-scale Generalized PDE Modeling. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. (pdf). (url). 48

D. Ha, A. M. Dai, and Q. V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2017. (pdf). (url). 28, 95

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (pdf). 31

M. A. Hedderich, L. Lange, H. Adel, et al. A Survey on Recent Approaches for Natural Language Processing in Low-Resource Scenarios. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, et al., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2545–2568, Online, jun 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.201. (pdf). (url). 93

J. Henderson. The Unstoppable Rise of Computational Linguistics in Deep Learning. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6294–6306, Online, jul 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.561. (pdf). (url). 28, 30, 41

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. (pdf). 31

H. Hersbach, B. Bell, P. Berrisford, et al. ERA5 hourly data on single levels from 1940 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2023. (url). Accessed on 2023-05-17. 54, 112

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. (pdf). 71

N. Houlsby, A. Giurgiu, S. Jastrzebski, et al. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019. (pdf). 94

S. Iyer, N. Dandekar, and K. Csernai. First Quora Dataset Release: Question Pairs, 2017. (url). 35

A. Jaegle, S. Borgeaud, J.-B. Alayrac, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021. (pdf). 57

S. Janny, A. Bénéteau, M. Nadri, et al. EAGLE: Large-scale Learning of Turbulent Fluid Dynamics with Mesh Transformers. In *The Eleventh International Conference on Learning Representations*, 2023. (pdf). (url). 112

D. I. Kabanov, L. Espath, J. Kiessling, and R. F. Tempone. Estimating divergence-free flows via neural networks. *PAMM*, 21(1):e202100173, 2021. doi: https://doi.org/10.1002/pamm.202100173. (pdf). (url). 85

I. Kakogeorgiou, S. Gidaris, K. Karantzalos, and N. Komodakis. SPOT: Self-Training with Patch-Order Permutation for Object-Centric Learning with Autoregressive Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22776–22786, June 2024. (pdf). 80

J. Kaplan, S. McCandlish, T. Henighan, et al. Scaling laws for neural language models, 2020. (pdf). 3, 4

R. Karimi Mahabadi, S. Ruder, M. Dehghani, and J. Henderson. Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576, Online, aug 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.47. (pdf). (url). 95

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 13–18 Jul 2020. (pdf). (url). 34, 36, 60, 94, 102, 103

R. Kikuchi, T. Misaka, S. Obayashi, et al. Nowcasting algorithm for wind fields using ensemble forecasting and aircraft flight data. *Meteorological Applications*, 25(3):365–375, 2018. (url). 13

H. Kim, A. Mnih, J. Schwarz, et al. Attentive Neural Processes. In *International Conference on Learning Representations*, 2019. (pdf). (url). 48, 52, 56

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. (pdf). (url). 96

T. Kojima, S. S. Gu, M. Reid, et al. Large Language Models are Zero-Shot Reasoners. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. (pdf). 131

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. (pdf). (url). 1

R. Lam, A. Sanchez-Gonzalez, M. Willson, et al. GraphCast: Learning skillful medium-range global weather forecasting, 2022. (pdf). (url). 46, 47, 48

M. Le, A. Vyas, B. Shi, et al. Voicebox: Text-Guided Multilingual Universal Speech Generation at Scale. In A. Oh, T. Naumann, A. Globerson, et al., editors, *Advances in Neural Information Processing Systems*, volume 36, pages 14005–14034. Curran Associates, Inc., 2023. (pdf). (url). 1

D. Lee, C. Kim, S. Kim, et al. Draft-and-Revise: Effective Image Generation with Contextual RQ-Transformer. In S. Koyejo, S. Mohamed, A. Agarwal, et al., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30127–30138. Curran Associates, Inc., 2022. (pdf). 80

J. Lee, Y. Lee, J. Kim, et al. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019. (pdf). (url). 48

J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon. FNet: Mixing Tokens with Fourier Transforms. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Proceedings*

of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4296–4313, Seattle, United States, jul 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.319. (pdf). (url). 34, 42, 95

J. Lezama, H. Chang, L. Jiang, and I. Essa. Improved Masked Image Generation with Token-Critic. In S. Avidan, G. Brostow, M. Cissé, et al., editors, *Computer Vision – ECCV 2022*, pages 70–86, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-20050-2. (pdf). 80

Q. Lhoest, A. V. del Moral, Y. Jernite, et al. Datasets: A community library for natural language processing. In H. Adel and S. Shi, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, nov 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-demo.21. (pdf). (url). 96

H. Li, T. Lan, Z. Fu, et al. Repetition In Repetition Out: Towards Understanding Neural Text Degeneration from the Data Perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. (pdf). (url). 4, 7, 86

S. Li, X. Jin, Y. Xuan, et al. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019. (pdf). 48

Z. Li, N. B. Kovachki, K. Azizzadenesheli, et al. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021. (pdf). (url). (code). 48, 55, 109, 110, 112, 117

Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier Neural Operator with Learned Deformations for PDEs on General Geometries, 2023b. (pdf). (url). (code). 117

D. Lian, Z. Yu, X. Sun, and S. Gao. AS-MLP: An Axial Shifted MLP Architecture for Vision. In *International Conference on Learning Representations*, 2022. (pdf). (url). 94

H. Liu, Z. Dai, D. So, and Q. V. Le. Pay Attention to MLPs. In M. Ranzato, A. Beygelzimer, Y. Dauphin, et al., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9204–9215. Curran Associates, Inc., 2021. (pdf). (url). 34, 35, 42, 94

A. Lou, C. Meng, and S. Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution, 2024. (pdf). 80

F. Mai, A. Pannatier, F. Fehr, et al. HyperMixer: An MLP-based Low Cost Alternative to Transformers. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15632–15654, Toronto, Canada, jul 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.871. (pdf). (url). (code). 5, 27

F. Mai, J. Zuluaga-Gomez, T. Parcollet, and P. Motlicek. HyperConformer: Multi-head HyperMixer for Efficient Speech Recognition. In *Interspeech 2023*. ISCA, 2023b. (pdf). (code). 42

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*, 2017. (pdf). 72

Météosuisse. Learn More About the COSMO Model, June 2014. (pdf). (url). Federal Department of Home Affairs FDHA, Federal Office of Meteorology and Climatology MeteoSwiss. 85, 86

Météosuisse. Prévisions météorologique avec GPU, Factsheet 1, 2024a. (pdf). (url). Federal Department of Home Affairs FDHA, Federal Office of Meteorology and Climatology MeteoSwiss. 86

Météosuisse. Le modèle de prévision ICON, Factsheet 2, 2024b. (pdf). (url). Federal Department of Home Affairs FDHA, Federal Office of Meteorology and Climatology MeteoSwiss. 86

N. Nayakanti, R. Al-Rfou, A. Zhou, et al. Wayformer: Motion Forecasting via Simple & Efficient Attention Networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2980–2987, 2023. doi: 10.1109/ICRA48891.2023.10160609. (pdf). 48, 115

NVIDIA. CUBLAS Library, 2007. (url). 12

NVIDIA, P. Vingelmann, and F. H. Fitzek. CUDA, release: 10.2.89, 2020. (url). 13

S. M. Omohundro. Five Balltree Construction Algorithms. Technical Report TR-89-063, International Computer Science Institute, December 1989. (pdf). 16

OpenAI. New embedding models and API updates, 2024. (url). Accessed: [01.03.2024]. 74

A. Pannatier, R. Picatoste, and F. Fleuret. *Efficient Wind Speed Nowcasting with GPU-Accelerated Nearest Neighbors Algorithm*, pages 325–333. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2022. doi: 10.1137/1.9781611977172.37. (pdf). (url). (code). 5, 48, 54, 107, 112, 118

A. Pannatier, E. Courdier, and F. Fleuret. σ-GPTs: A New Approach to Autoregressive Models. In *Machine Learning and Knowledge Discovery in Databases: Research Track*. Springer Nature Switzerland, 2024a. ISBN 978-3-031-70358-4. (pdf). (code). 5

A. Pannatier, K. Matoba, and F. Fleuret. Inference from Real-World Sparse Measurements. *Transactions on Machine Learning Research*, 2024b. ISSN 2835-8856. (pdf). (url). (code). 5

V. Papadopoulos, J. Wenger, and C. Hongler. Arrows of Time for Large Language Models, 2024. (pdf). 80

A. Paszke, S. Gross, F. Massa, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, et al., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. (pdf). (url). 12, 13, 18, 26, 95

F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (url). 25, 89

W. Peebles and S. Xie. Scalable Diffusion Models with Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4195–4205, October 2023. (pdf). 5

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia. Learning Mesh-Based Simulation with Graph Networks. In *International Conference on Learning Representations*, 2020. (pdf). 48, 112

S. Pulkkinen, D. Nerini, A. A. P. Hortal, et al. Pysteps: an open-source Python library for probabilistic precipitation nowcasting (v1.0). *Geoscientific Model Development Discussions*, pages 1–68, 2019. ISSN 1991-959X. doi: 10.5194/gmd-2019-94. (url). (code). 11

A. Radford, K. Narasimhan, T. Salimans, et al. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018. (url). 1, 3, 28, 60, 81

A. Radford, J. Wu, R. Child, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. (pdf). 1, 67

A. Radford, J. W. Kim, T. Xu, et al. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*, 2022. (pdf). (url). 1, 60

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016. (pdf). 35

R. Rombach, A. Blattmann, D. Lorenz, et al. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022. (pdf). (code). 1

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. (pdf). 94

T. Schick and H. Schütze. It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, et al., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, pages 2339–2352, Online, jun 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.185. (pdf). (url). 94

R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020. (pdf). 28, 37, 38, 93

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*, 2017. (pdf). 13, 46, 47

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, 2014. (pdf). (url). 97

R. Socher, A. Perelygin, J. Wu, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013. (pdf). 30, 35

Y. Song, P. Dhariwal, M. Chen, and I. Sutskever. Consistency Models. In A. Krause, E. Brunskill, K. Cho, et al., editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32211–32252. PMLR, 23–29 Jul 2023. (pdf). (url). 80

E. Strubell, A. Ganesh, and A. McCallum. Energy and Policy Considerations for Deep Learning in NLP. In A. Korhonen, D. Traum, and L. Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, jul 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355. (pdf). (url). 28, 93

R. Suman, L. Karel, W. Matthew, et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677, 09 2021a. ISSN 1476-4687. doi: 10.1038/s41586-021-03854-z. (pdf). (url). 11, 13

R. Suman, L. Karel, W. Matthew, et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677, 09 2021b. ISSN 1476-4687. doi: 10.1038/s41586-021-03854-z. (pdf). (url). 46, 47

J. Sun, H. Vû, J. Ellerbroek, and J. Hoekstra. Ground-based Wind Field Construction from Mode-S and ADS-B Data with a Novel Gas Particle Model. In *Proceedings of the Seventh SESAR Innovation Days*, 2017. (pdf). (url). (code). 7th SESAR Innovation Days, SIDs. 12, 13, 21, 22

P. Sun, Y. Jiang, S. Chen, et al. Autoregressive Model Beats Diffusion: Llama for Scalable Image Generation, 2024. (pdf). 5

C. Tang, Y. Zhao, G. Wang, et al. Sparse MLP for Image Recognition: Is Self-Attention Really Necessary? *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(2):2344–2351, Jun. 2022. doi: 10.1609/aaai.v36i2.20133. (pdf). (url). (code). 94

Y. Tatsunami and M. Taki. RaftMLP: How Much Can Be Done Without Attention and with Less Spatial Locality?, December 2022. (pdf). 94

Y. Tay, Z. Zhao, D. Bahri, et al. HyperGrid Transformers: Towards A Single Model for Multiple Tasks. In *ICLR 2021*, 2021. (pdf). 95

N. Thuerey, P. Holl, M. Mueller, et al. *Physics-based Deep Learning*. WWW, 2021. (url). 85

I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, et al. MLP-Mixer: An all-MLP Architecture for Vision. In M. Ranzato, A. Beygelzimer, Y. Dauphin, et al., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24261–24272. Curran Associates, Inc., 2021. (pdf). (url). 6, 27, 28, 34, 35, 94

Z. Tu, H. Talebi, H. Zhang, et al. MAXIM: Multi-Axis MLP for Image Processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5769–5780, June 2022. (pdf). 34, 94

V. Varma, R. Shah, Z. Kenton, et al. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023. (pdf). 77

A. Vaswani, N. Shazeer, N. Parmar, et al. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, et al., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. (pdf). 3, 28, 32, 36, 48, 50, 60, 68, 83, 103

R. Villegas, M. Babaeizadeh, P.-J. Kindermans, et al. Phenaki: Variable Length Video Generation from Open Domain Textual Descriptions. In *International Conference on Learning Representations*, 2023. (pdf). 70, 80

A. Wang, A. Singh, J. Michael, et al. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In T. Linzen, G. Chrupała, and A. Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, nov 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. (pdf). (url). 35, 96

B. Wang and A. Komatsuzaki. GPT-J-6B: A 6 billion parameter autoregressive language model, 2021. (url). 104

Q. Wang, B. Li, T. Xiao, et al. Learning Deep Transformer Models for Machine Translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, 2019. (pdf). 31, 103

Z. Wang, W. Jiang, Y. M. Zhu, et al. DynaMixer: A Vision MLP Architecture with Dynamic Mixing, 17–23 Jul 2022. (pdf). (url). 34, 95

J. Wei, X. Wang, D. Schuurmans, et al. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. (pdf). 131

Wikipedia contributors. Secondary surveillance radar — Wikipedia, The Free Encyclopedia, 2021. (url). [Online; accessed 30-January-2021]. 13

A. Williams, N. Nangia, and S. Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, jun 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1101. (pdf). (url). 35

Z. Yang, Z. Dai, Y. Yang, et al. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, et al., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. (pdf). 79

P. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 02 1970. (pdf). 12, 16

T. Yu, X. Li, Y. Cai, et al. S2-MLP: Spatial-Shift MLP Architecture for Vision. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 297–306, January 2022a. (pdf). 34, 94

W. Yu, M. Luo, P. Zhou, et al. MetaFormer Is Actually What You Need for Vision, June 2022b. (pdf). 36

Y. Yuan, X. Weng, Y. Ou, and K. M. Kitani. AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9813–9823, October 2021. (pdf). 48, 115

M. Zaheer, S. Kottur, S. Ravanbakhsh, et al. Deep Sets. In I. Guyon, U. V. Luxburg, S. Bengio, et al., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. (pdf). 52

A. Zhmoginov, M. Sandler, and M. Vladymyrov. HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning. In K. Chaudhuri, S. Jegelka, L. Song, et al., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 27075–27098. PMLR, 17–23 Jul 2022. (pdf). (url). 95

ARNAUD PANNATIER

PhD Candidate in Attention-based models

PhD Candidate within Pr. François Fleuret's Machine Learning group (Idiap Research Institute, EPFL), primarily focusing on attention models and their adaptability to various domains, ranging from wind prediction, air traffic control, vision, to natural language processing. I collaborate closely with companies on practical problems using real-world data.

EDUCATION

PhD in Machine Learning
Innosuisse, with SkySoft ATM
Idiap Research Institute, EPFL
2020 - 2024

Master in Computer Sciences and Engineering
Math Faculty, EPFL
2017 - 2020

Bachelor in Physics
Physics Faculty, EPFL
2014 - 2017

LANGUES

French : Native speaker
English : Fluent
German: Basic knowledge

SKILLS

Expert : PyTorch, Python, Javascript
Advanced: Jax, C++, Go, Django, Matlab
Basic : Docker, SQL, PHP, Swift

INFORMATIONS

Rue des Longs-Prés 40
3960 Sierre
+41 77 439 30 16
arnaud.pannatier@idiap.ch
https://arnaudpannatier.ch

Born September 20, 1995, in Sion, Valais
Married, two children (born 2022, 2023)

REFERENCES

François Fleuret, francois.fleuret@unige.ch
Michael Liebling, mliebling@idiap.ch
J.-C. Chappelier, jean-cedric.chappelier@epfl.ch

PUBLICATIONS

A. Pannatier, E. Courdier, F. Fleuret **σ-GPTs: A New Approach to Autoregressive Models.** In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) 2024*

A. Pannatier, K. Matoba, F. Fleuret **Inference from Real-World Sparse Measurements** In *Transactions on Machine Learning Research (TMLR), 2024*

F. Mai, A. Pannatier, F. Fehr, H. Chen, F. Marelli, F. Fleuret, J. Henderson **HyperMixer: An MLP-based Low Cost Alternative to Transformers.** In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 2023.* DOI: 10.18653/v1/2023.acl-long.871

A. Pannatier, R. Picatoste, and F. Fleuret. **Efficient Wind Speed Nowcasting with GPU-Accelerated Nearest Neighbors Algorithm.** In *Proceedings of the SIAM International Conference on Data Mining (SDM), 2022.* DOI: 10.1137/1.9781611977172.37

A. Pannatier **A Control Plane in Time and Space for Locality-Preserving Blockchains** In *Master Thesis, Decentralized Distributed Systems Laboratory (DEDIS), EPFL, 2020*
Price: Kudelski Award

TEACHING EXPERIENCE

Teaching Assistant, EPFL
- Advanced Analysis 1-2 Pr. Stubbe, 2016, 2017, 2018
- Analysis 2 Pr. Buffoni, 2018
- Advanced Analysis 3 Pr. Krieger, 2018
- Introduction à l'apprentissage automatique Pr. Liebling, 2020, 2023
- Deep Learning Pr. Fleuret, 2021, 2022
- Information, Computation, Communication Pr. Chappelier, 2020, 2022, 2023

Substitute Math Teacher
Lycée Collège de la Planta
Mathematics, Advanced Mathematics
Mme Jordan, Mme Veuthey, Mr. Petit 1-5e, 2017 - 2019

WORK EXPERIENCE

Software engineer
Caelum Fintech SA, Technopôle, Sierre
Principal engineer, solution development and creation of a Predictive Power analysis tool.
2018 - 2020 (Part time)

INTERESTS

Hiking
Hiked from home (VS/CH) to Santiago de Compostella (ES)
2300 km in 83 days (April 2019 - July 2019)

Chess
Treasurer of the Union Valaisanne des Échecs (UVE-WSB)
Play with Valais 1 (League A) and Sion 1 (League B)
Ex-Captain of Chess Team Valais 2 (2nd league)