`PhD-FSTM-2025-113`
The Faculty of Sciences, Technology and Medecine

Dissertation

Defence held on September 12^{th}, 2025 in Luxembourg

to obtain the degree of

Docteur de l'Université du Luxembourg en Informatique

by

Fabien Bernier
Born on 8th January 1998 in Saint-Nazaire (France)

Scalable, Accurate and Context-Aware Energy Demand Forecasting for Smart Grids

Dissertation defence committee

Dr. Gilbert Fridgen, Chairman
Full Professor, Université du Luxembourg

Dr. Yves Le Traon, Dissertation Supervisor
Full Professor, Université du Luxembourg

Dr. Maxime Cordy
Assistant Professor, Université du Luxembourg

Dr. Miguel Couceiro
Full Professor, Instituto Superior Técnico, Lisbon, Portugal

Dr. Matthieu Jimenez
Data Scientist, Administration des Contributions Directes, Luxembourg

Full committee

Dr. Gilbert Fridgen, Chairman
Full Professor, Université du Luxembourg

Dr. Yves Le Traon, Dissertation Supervisor
Full Professor, Université du Luxembourg

Dr. Maxime Cordy
Assistant Professor, Université du Luxembourg

Dr. Miguel Couceiro
Full Professor, Instituto Superior Técnico, Lisbon, Portugal

Dr. Matthieu Jimenez
Data Scientist, Administration des Contributions Directes, Luxembourg

Dr. Sylvain Kubler, Expert in an advisory capacity
Research Scientist, Université du Luxembourg

Affidavit / Statement of originality

I declare that this thesis:

- is the result of my own work. Any contribution from any other party, and any use of generative artificial intelligence technologies have been duly cited and acknowledged;
- is not substantially the same as any other that I have submitted, and;
- is not being concurrently submitted for a degree, diploma or other qualification at the University of Luxembourg or any other University or similar institution except as specified in the text.

With my approval I furthermore confirm the following:

- I have adhered to the rules set out in the University of Luxembourg's Code of Conduct and the Doctoral Education Agreement (DEA)[1], in particular with regard to Research Integrity.
- I have documented all methods, data, and processes truthfully and fully.
- I have mentioned all the significant contributors to the work.
- I am aware that the work may be screened electronically for originality.

I acknowledge that if any issues are raised regarding good research practices based on the review of the thesis, the examination may be postponed pending the outcome of any investigation of such issues. If a degree was conferred, any such subsequently discovered issues may result in the cancellation of the degree.

Approved on 2025-08-21

---

[1] If applicable (DEA is compulsory since August 2020)

Abstract

Energy demand forecasting at large scale is a critical challenge for modern power grid operators, especially as the transition to all-electric systems, the integration of renewables, and evolving consumption behaviors introduce greater variability and complexity. In an industrial partnership with Creos Luxembourg S.A., this PhD thesis addresses the needs of grid management through a comprehensive exploration of scalable, accurate, and practical solutions to electricity consumption forecasting at the household level.

The first axis of this thesis targets scalability in forecasting solutions. Existing state-of-the-art machine learning models often suffer from significant computational costs, rendering them impractical for large-scale and frequent deployment. This manuscript introduces Transplit, a novel transformer-based forecasting architecture that exploits the inherent seasonality of consumption data by encoding entire seasonal cycles (e.g., days) as compact vectors. This approach dramatically reduces the sequence length processed by the model, resulting in a lightweight method capable of training on thousands of households with minimal hardware (including CPUs), while achieving competitive accuracy against deep learning baselines, paving the way for fine-grained, frequent grid state estimation at population scale.

The second research direction focuses on peak demand forecasting, essential for operational safety, economic dispatching, and load balancing, yet under-addressed by classical error metrics (MSE, MAE) that flatten out local maxima. To overcome this limitation, we propose DeDiPeak, a framework comprising new *Peak Prediction Performance* (P3) metrics — specifically designed to assess forecasted peaks on both timing and amplitude — with a deblurring diffusion model. This diffusion post-processor can be plugged into any forecasting model to selectively enhance peaks, leading to significant improvements in peak-specific accuracy without degrading the overall error. Empirical results show up to a 36% gain in peak forecasting quality over benchmarks, not only on electricity consumption, but also on a broad spectrum of real-world datasets.

The third axis investigates the leverage of external factors, such as weather, holidays, and special events, in global forecasting models. While traditional models often degrade when incorporating such exogenous variables, we show that a hypernetwork-based architecture can bridge this gap by dynamically generating consumer-specific model weights conditioned on compact embeddings that represent both household identity and contextual factors. Evaluated on multi-year, multi-household datasets containing both consumption records and aligned external signals, our hypernetwork approach consistently outperforms classical, global, and expert-mixture models, achieving individual-level accuracy without a costly per-consumer model maintainance, and enabling fast adaptation to changing consumer populations.

Finally, to a lesser extent, this thesis explores the emergent use of Large Language Models (LLMs) in power system optimization, and especially their ability to solve the Optimal Power Flow (OPF) problem by querying LLMs with graph- and table-based representations of realistic power grids. We show that LLMs can be adapted to reliably solve OPF instances, with their performance improving as model size and tailored training increase, thereby opening new perspectives for cross-domain modeling in the energy sector.

Together, these contributions advance the state of the art in scalable grid analytics, robust peak event prediction, multi-factor consumption modeling, and cross-disciplinary energy system optimization. The resulting frameworks are directly applicable within the industrial context of Creos Luxembourg S.A., encouraging the deployment of next-generation forecasting tools, which are critical for tomorrow's smart grids.

Acknowledgments

Throughout this journey, many people have shaped both my research and personal growth, and while words may not be enough in capturing my gratitude, I wish to acknowledge those who made this thesis possible.

My deepest appreciation first goes to Pr. Yves Le Traon, who opened the doors to this opportunity. Although his supervision capabilities decreased over time as he was transitioning to the role of director of the SnT, he still found time for monthly discussions on my PhD and keep track with the whole storyline.

I am profoundly grateful to Dr. Maxime Cordy, whose guidance transformed my understanding of academia. Beyond supervising my work, he became a mentor who challenged me to take a step back, think at a higher level, and question assumptions. His patience during my moments of doubt and his openness created an environment where I could effeciently evolve with minimal stress.

Special recognition goes to Dr. Matthieu Jimenez, my office companion and close collaborator. Our daily exchanges, heated discussions about research directions, and shared moments of discoveries made this journey far richer. His insights from the Serval team and friendship transformed what could have been solitary work into a collaborative adventure for some time.

I extend warm thanks to CREOS, our industrial partner, especially Yves Reckinger and Robert Graglia, whose real-world challenges provided the perfect testing ground for our theoretical contributions. Their openness to academic collaboration and willingness to share their expertise bridged the gap between research and practice in meaningful ways.

To the distinguished jury members: thank you for your time, constructive feedback, and genuine interest in this work. Your perspectives have undoubtedly strengthened the final outcome.

The entire SerVal team deserves recognition for creating such a vibrant research community. For fear of missing names because there are so many people I would like to thank, I prefer not to cite anyone here. Some even became friends, close friends, and they will recognize themselves :) From spontaneous coffee break conversations to intensive swimming sessions, each interaction contributed to the intellectual richness of this experience. You made the lab feel like a second home, which is the ultimate reason that encouraged me to come on site everyday. Remote days were extremely rare.

Finally, my heartfelt gratitude to Océane, whose love and support anchored me through every challenge — understand: deadlines. Her belief in me, especially during hard moments, reminded me why this journey mattered. Thank you for being my constant source of happiness.

Contents

1 Introduction 1
1.1 Context: towards an all-electric society . . . 2
1.2 Limitations of state-of-the-art forecasting pipelines . . . 2
1.3 Research objectives . . . 3
1.4 Methodological overview . . . 3
1.5 Industrial partnership and datasets . . . 4
1.6 Outline of the manuscript . . . 5
1.7 Contributions . . . 5

2 Background 7
2.1 From Statistical to Machine Learning Approaches . . . 8
2.2 Neural Network Architectures for Time Series . . . 8
2.3 The CREOS dataset . . . 9

3 Transplit: Faster and Cheaper Seasonal Forecasting at Scale 11
3.1 Context . . . 12
3.2 Background . . . 13
3.3 Proposed approach . . . 13
  3.3.1 Motivations . . . 13
  3.3.2 Transplit . . . 14
  3.3.3 Season tokenization . . . 15
3.4 Evaluation . . . 18
  3.4.1 Research Questions . . . 18
  3.4.2 Models . . . 18
  3.4.3 Data . . . 18
  3.4.4 Experimental settings . . . 19
3.5 Results . . . 20
  3.5.1 Speed and lightness . . . 20
  3.5.2 Accuracy . . . 22

3.5.3 Case study 22
3.6 Optimizing Transplit 24
3.6.1 The seasonality hyperparameter 24
3.6.2 Optimal sequence length 25
3.6.3 Loss trade-off 26
3.7 Discussion 26

4 Peak Forecasting Enhancement with Deblurring Diffusion Models 31
4.1 Context 32
4.1.1 Transplit limitations 32
4.1.2 MSE loss: an already-known culprit 33
4.1.3 Dedipeak 34
4.2 A small history with early experiments 35
4.2.1 Analogy with image generation 35
4.2.2 Using a GPT-like architecture 36
4.3 Background and Related Work 39
4.3.1 Time series and peak forecasting 39
4.3.2 Denoising Diffusion Probabilistic Models 41
4.3.3 Inverse Heat Dissipation 42
4.3.4 Parallel with DDPMs 44
4.3.5 Noise Scheduler 46
4.3.6 Adaptive loss 46
4.4 DeDiPeak 46
4.4.1 Peak Forecasting Metrics 46
4.4.2 Effectively Forecasting Peaks 48
4.5 Experiments 50
4.5.1 Empirical Setup 50
4.5.2 Implementation details 50
4.5.3 Results 51
4.5.4 Comparison Studies 53
4.6 Details and additional results 55
4.6.1 Implementation 55
4.6.2 Additional Visual Results 59
4.6.3 Effect of noise 62
4.6.4 Metric parameters 62
4.6.5 Inference time 64
4.6.6 Performance of a probabilistic model 66
4.6.7 Peak rate 68
4.7 Discussion 68

5 Leveraging External Factors in Household-Level Electrical Consumption Forecasting using Hypernetworks 71
5.1 Context . . . 72
5.2 Early attempts in Transplit adaptation . . . 74
5.3 Background . . . 76
5.4 Hypernetworks for Time Series Forecasting . . . 77
    5.4.1 Problem Formulation . . . 77
    5.4.2 Model Architecture . . . 78
    5.4.3 Experimental setup . . . 80
5.5 Results . . . 82
    5.5.1 Impact of External Factors . . . 82
    5.5.2 Performance Across Model Architectures. . . . 83
    5.5.3 Cost . . . 84
    5.5.4 Generalizing consumers embeddings . . . 84
    5.5.5 Ablation studies . . . 84
5.6 Discussion . . . 86
5.7 Conclusion . . . 87

6 PowerGraph-LLM: Novel Power Grid Graph Embedding and Optimization with Large Language Models 89
6.1 Context . . . 90
6.2 PowerGraph-LLM Framework . . . 91
6.3 Empirical Study . . . 93
    6.3.1 Experimental Protocol . . . 93
    6.3.2 Effectiveness of pre-trained LLMs . . . 95
    6.3.3 Impact of the size of the model . . . 95
    6.3.4 Impact of fine-tuning . . . 95
6.4 Conclusion . . . 96

7 Conclusion 99
7.1 Summary of Contributions . . . 100
7.2 Limitations . . . 101
7.3 Perspectives for Future Work . . . 101
7.4 Concluding Remarks . . . 102

8 Appendices 103
8.1 Time series token prediction with adaptive neural decision trees . . . 104

List of publications and tools i

List of figures i

List of tables **vi**

Bibliography **ix**

1

Introduction

With the energy sector transitioning from fossil fuels to renewable generation and all-electric appliances, power grid operators face new challenges including weather-dependent evolving consumption patterns and shorter planning horizons that make accurate load forecasting critical for grid stability. This chapter introduces these challenges through the presentation of three main contributions: Transplit (a lightweight transformer for seasonal time series), DeDiPeak (a diffusion-based approach for peak forecasting), and contextual hypernetworks for personalized forecasting, all developed in partnership with Luxembourg's grid operator Creos S.A.

Contents

1.1 Context: towards an all-electric society 2
1.2 Limitations of state-of-the-art forecasting pipelines 2
1.3 Research objectives 3
1.4 Methodological overview 3
1.5 Industrial partnership and datasets 4
1.6 Outline of the manuscript 5
1.7 Contributions 5

1.1 Context: towards an all-electric society

Over the last decade the energy ecosystem has entered an unprecedented transition: fossil-fuel based production is being progressively replaced by distributed renewable generation, while end-users switch to all-electric appliances for mobility, heating and industry. Although this paradigm shift is indispensable from a sustainability perspective, it also compromises the real-time balance that modern power systems are designed to maintain. In particular, distribution system operators must now cope with *(i)* larger forecast errors caused by weather-dependent generation, *(ii)* rapidly evolving consumption habits (*e.g.* electric vehicles, heat pumps), and *(iii)* the decreasing planning horizon imposed by regulatory and market rules. Precise and reliable short-term *load forecasting* has therefore become important for operational security, preventive maintenance and even optimal economic dispatch [NTN+20; HF16]. Luxembourg is no exception to this trend, where the national grid operator Creos Luxembourg S.A., our industrial partner, is tasked with ensuring the stability of the electricity network in the face of these challenges.

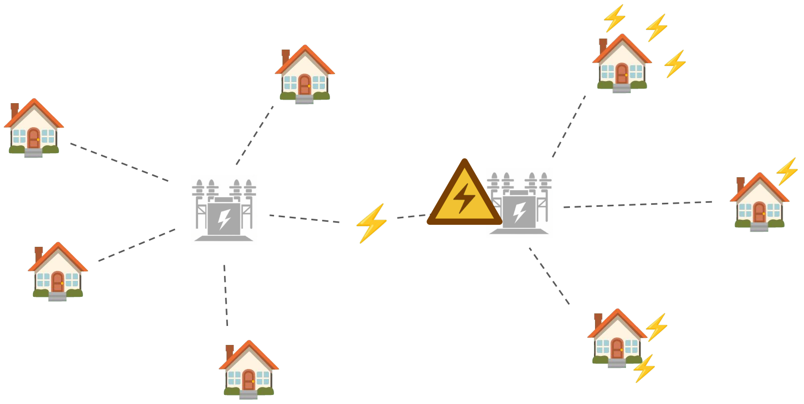


Figure 1.1: Important localized overconsumption can lead to network overloads, which threaten the stability of the power grid.

1.2 Limitations of state-of-the-art forecasting pipelines

The recent wave of deep-learning based time-series models — recurrent [ZM98; YSL+19], convolutional networks [MS22], transformers [VSP+17; whXW+21; ZZP+21] and, more recently, diffusion approaches [RSS+21] — raised the accuracy bar quite significantly. Yet their applicability at *population scale* remains limited:

1. **Scalability**. Training large models for every smart-meter individually or retraining a global model on *millions* of meter-hours quickly becomes exorbitant in CPU/GPU time and energy consumption. This tension gave rise to the field of *Frugal Machine Learning* [EVH+21; PG21], which calls for resource-aware forecasting solutions.

2. **Peak-centric accuracy**. Standard error metrics such as MAE/MSE smooth out local maxima, while distribution system operators are very interested in *peaks* that threaten network stability [DMD+21; ABC+05]. Models optimised for average behaviour often predict *shy* peaks (low amplitude, wrong timing) as illustrated in Fig. 4.2.

3. **Multi-factor interactions**. Electrical demand depends on exogenous variables—weather, calendar events, socio-economic context—whose impact is *customer-specific*. Global architectures struggle to exploit these heterogeneous signals, whereas individual models capture them but at an unrealistic operational cost.

1.3 Research objectives

The overarching goal of this thesis is to devise *scalable, accurate and practical* forecasting methodologies that fulfill industrial-grade requirements. Concretely, three intertwined questions are being addressed:

RQ1 How can the *computational footprint* of deep sequence-to-sequence forecasters be reduced without sacrificing predictive power?

RQ2 How can one *quantify* and subsequently *improve* the prediction of demand peaks that critically drive grid dimensioning?

RQ3 How can external factors be *integrated* into large-scale models while still accounting for household-level behaviors?

RQ4 Can emerging *Large Language Models* (LLMs) contribute to core power-system optimisation tasks such as the Optimal Power Flow (OPF) problem?

1.4 Methodological overview

To answer these questions, we propose a three-axes research programme:

1. Transplit: a Transformer tailored for Seasonal Time Series. We introduce Transplit, a lightweight encoder–only Transformer that exploits the strong daily/weekly periodicity of electricity load. A *Slice-to-Vector-to-Slice* (SVS) module converts entire seasons (*e.g.* days) into compact embeddings, so that the self-attention mechanism operates on 30 vectors instead of 720 raw points for a monthly context window. This architectural simplification divides both training time and GPU memory by up to three orders of magnitude while retaining accuracy on standard benchmarks such as the ECL, Weather, Traffic and ETTm2 time series datasets. Moreover, Transplit can be retrained on ~6000 households in <30 minutes on a single CPU, opening opportunities for frugal, population-scale forecasting.

2. DeDiPeak: Deblurring Diffusion for Peak Forecasting. To restore sharpness lost during point-wise optimisation, we introduce DeDiPeak, a plug-and-play post-processor that performs the last steps of an inverse heat-dissipation diffusion model [RHS22; HS22b]. Two dedicated metrics, the *Peak Prediction Performance* scores $P3_{\mathrm{E}}$ and $P3_{\mathrm{sw}}$, jointly evaluate timing and amplitude errors, overcoming the limitations of traditional indicators. Applied to the outputs of any base forecaster, the diffusion module selectively enhances local maxima and yields up to a 36% improvement in peak accuracy on electricity, traffic, weather and financial datasets, with negligible degradation of the global MSE.

3. Contextual Hypernetworks for Personalised Forecasting. Finally, we tackle the integration of exogenous variables through a *hypernetwork* that dynamically generates household-specific weights conditioned on both customer embeddings and contextual features (temperature, holidays, sport events, etc.). The resulting architecture reconciles the efficiency of a single global model with the precision of per-user fine-tuning, outperforming strong baselines (Informer, NHits, expert mixtures) by up to 16 % on a two-year, 6000-meter dataset provided by Creos.

Transplit and Dedipeak, in particular, are not only studied on electrical consumption time series, but also evaluated on more general time series datasets, allowing for broader usage outside of the scope of electrical load consumption forecasting.

1.5 Industrial partnership and datasets

All methods were developed and evaluated in close collaboration with Creos Luxembourg S.A., which granted access to an anonymised dataset of more than 6 000 smart meters, as mentioned above. Public benchmarks (ECL [Dat14], Weather, Traffic) were additionally used to ensure replicability and external validity.

1.6 Outline of the manuscript

The remainder of this manuscript is organised as follows. Chapter 2 reviews the literature on general time-series forecasting — more specific background will be added in respective chapters. Chapter 3 introduces Transplit and its frugal training procedure. Chapter 4 formalises the P3 metrics and details the DeDiPeak framework. Chapter 5 presents the hypernetwork architecture for context-aware forecasting. Chapter 6 investigates the use of LLMs for Optimal Power Flow. Finally, Chapter 7 summarises the findings and discusses future work.

1.7 Contributions

For the reader's convenience we recap below the main contributions delivered throughout the thesis:

- **Transplit** — a seasonal-aware transformer that achieves up to 940× faster training on CPU than competitive deep models, enabling nationwide deployment on commodity hardware.
- **P3 metrics** — first reproducible indicators ($P3_{sw}, P3_{\mathcal{E}}$) dedicated to peak demand evaluation, released as an open-source Python package.
- **DeDiPeak** — a diffusion-based post-processor that lifts peak forecasting accuracy by up to 36 % without degrading global MAE.
- **Hypernetwork architecture** — a compact, plug-and-play module that injects external factors and household embeddings into a single global model, outperforming state-of-the-art individual and mixture baselines by up to 16 %.
- **First evidence of LLM-based OPF solving** — demonstrating the feasibility of leveraging large language models for core power-system optimisation tasks.

Added together, these results advance the state of the art in frugal grid analytics, peak-centric evaluation, multi-factor forecasting and cross-disciplinary energy optimisation, made to specifically fit Creos Luxembourg's needs.

2

Background

This chapter introduces the common background on time series forecasting for the contributions in this thesis, as well as the dataset provided by Creos, regularly mentioned and used in the next chapters.

Contents

2.1 From Statistical to Machine Learning Approaches . . . 8
2.2 Neural Network Architectures for Time Series 8
2.3 The CREOS dataset 9

Time series forecasting has evolved significantly last decades, transitioning from classical statistical approaches to sophisticated deep learning architectures. This progression has been driven by increasing data availability, growing computational power, and the need to address complex forecasting challenges across various domains.

2.1 From Statistical to Machine Learning Approaches

Early forecasting methodologies were predominantly based on statistical time series models that leveraged temporal patterns such as seasonality. The Autoregressive Integrated Moving Average (ARIMA) model [BJ76] illuminated these traditional parametric approaches, alongside exponential smoothing techniques [Gar85]. While these methods provided acceptable baseline performance, they exhibited significant limitations when confronted with non-linear data patterns and struggled to effectively incorporate multiple information sources.

The emergence of machine learning techniques marked a new shift in the forecasting landscape. These approaches demonstrated superior capability in handling data non-linearity and complex temporal dependencies. The recent exponential growth in computational resources and the availability of sophisticated learning frameworks have further accelerated the adoption of machine learning-based forecasting models, leading to a proliferation of novel architectural proposals.

2.2 Neural Network Architectures for Time Series

Among neural network approaches, Long Short-Term Memory (LSTM) networks have been extensively used for time series forecasting tasks, and, to a minor extent, continue to be subjects of active research [LCY^{+}18; YSL^{+}19]. These recurrent architectures excelled at capturing long-term dependencies in sequential data, making them particularly suitable for temporal forecasting applications.

However, the introduction of the transformer architecture by Vaswani *et al.*[VSP^{+}17] marked a new turn for sequence modeling — originally in natural language processing, and subsequently found widespread application in time series forecasting. Multiple transformer-based models have been proposed specifically for forecasting tasks, including the Informer [ZZP^{+}21], Reformer [KKL20], and Autoformer [whXW^{+}21]. These architectures primarily differ in their attention mechanisms, which are strategically modified to reduce computational complexity while maintaining or improving forecasting accuracy.

Recent innovations have further advanced the field through specialized techniques. Patching strategies, as very well represented by PatchTST [NNS^{+}23], decompose time series into overlapping subseries to improve both computational efficiency and model performance. These approaches leverage the inherent seasonal properties of time series data to optimize architectural design for enhanced time and resource efficiency.

More computer-vision inspired architectures more recently emerged, such as diffusion models or VQ-VAEs. These architectures will be further treated in specific background sections in the following chapters.

Overall, the continuous evolution of these architectures reflects the ongoing effort to balance forecasting accuracy with computational efficiency, while addressing the diverse challenges present across different time series forecasting applications.

2.3 The CREOS dataset

In the first months of the PhD, Creos shared a dataset containing the consumption time series from 6,010 smart meters. The consumption is measured in kWh every 15 minutes, over a period of two years, from January 1, 2020, 0:00 to December 31, 2021, 23:45. To reduce consumption noise and to align with time series benchmarks, the frequency of the dataset is reduced to hourly measures by summing the consumption within every hour.

3

Transplit: Faster and Cheaper Seasonal Forecasting at Scale

Transformer models have made significant progress in seasonal time series forecasting, but their high computational costs and training time have limited practical adoption. We introduce in this chapter Transplit, a global lightweight transformer-based model that learns typical seasonal patterns to efficiently forecast multiple time series while running several times faster than state-of-the-art models — even on a single CPU —- with competitive performance across domains like power consumption forecasting.

Contents

3.1 Context 12
3.2 Background 13
3.3 Proposed approach 13
3.4 Evaluation 18
3.5 Results 20
3.6 Optimizing Transplit 24
3.7 Discussion 26

3.1 Context

Time series with seasonal patterns are omnipresent across various domains, from weather patterns and traffic flows to energy consumption. These datasets present unique challenges for forecasting, particularly when the underlying patterns evolve over time due to external factors. Traditional, statistical forecasting approaches have relied on historical data combined with domain-specific features, while more recent neural network-based methods have demonstrated promising results across different applications.

However, the increasing availability of high-frequency data through modern methods has introduced new challenges. While recent deep learning approaches achieve impressive accuracy, they often require substantial computational resources and training time, making them impractical for scenarios requiring frequent model updates to adapt to evolving patterns. This limitation is particularly evident in real-world applications where patterns change due to various factors such as climatic, societal and economic context changes. These computational challenges, among others, led to the emergence of the *Frugal Machine Learning* field [EVH^{+}21][PG21], which aims to develop low-resource models without compromising accuracy.

In line with these challenge responses, we introduce Transplit, a transformer-based model designed for efficient seasonal time series forecasting. Transplit exploits the seasonal properties of time series by means of a novel encoder/decoder method. The key principles of this method is to slice the time series and map the obtained slices to a base of curve patterns. This enables efficient processing of temporal patterns while maintaining high accuracy with reduced resource requirements, making it suitable for deployment on low-cost hardware configurations without requiring specialized resources like GPU.

To demonstrate the genericity and effectiveness of Transplit, we conduct extensive evaluations across different types of seasonal time series, including weather patterns, traffic flows, and electricity consumption data. In particular, we present an in-depth case study in the context of electricity demand forecasting on the CREOS dataset, where accurate predictions at scale are crucial for grid management, maintenance scheduling, and load-demand planning.

Our results show that Transplit achieves comparable accuracy to the best-performing methods while being significantly faster (**3** to **940** times) when trained on a single CPU, making it particularly suitable for applications requiring frequent model updates to adapt to evolving patterns while operating under computational constraints.

3.2 Background

Among ML-based approaches, we can distinguish two categories of learning models.

Small models on individual time series: The first category consists in training one model per individual. The model learns the time series profile (e.g. power consumption from a single user) and predicts future variation according to its characteristics (e.g. the user's habits). This is a task historically achieved by models such as random forests (RF), SVMs, LSTMs or linear regressions [PKA$^+$21][GHC$^+$19]. A very recent example of such a model is Neural Prophet (2021) [THP$^+$21], which has been applied notably to traffic flow forecasting [ZHB$^+$24], as well as to electric load consumption [SFF22]. Recently, Mallen *et al.*also found a lightweight and simple method based on the Koopman operator theory to make *"in phase"* long-term forecasts [MLK21], particularly useful when applied to seasonal data such as consumption load. Although these recent models perform well, they have to be trained on a per-user basis, which rapidly becomes expensive to deploy at large scale.

Bigger models on many profiles: The second category relies on a single model for several customers. These models will generalize different behaviors of the time series, discern user's profile, and prolong their curve according to their shape. The biggest advantage of these approaches is that we only need to train the model once to forecast every user's time series, including those of new comers. In particular, approaches such as LogTrans [LJX$^+$19], the Reformer [KKL20], then the Informer [ZZP$^+$21] and most recently — at the time of the contribution — the Autoformer [whXW$^+$21] derived from it have been suggested for efficient long-term forecasting. The main differences between these approaches lie in their inner attention mechanism – *i.e.* what to focus on to forecast the next steps – that is changed to lower the computation complexity. Since time representation is also important for efficiency, Kazemi *et al.*also proposed Time2Vec [KGE$^+$19], a method to reduce the dimensionality of sequences while preserving essential information, leading to improvement in terms of computational complexity.

3.3 Proposed approach

3.3.1 Motivations

In most applications, seasonal time series forecasting can be performed at different granularity levels: local to global temperature for weather forecasting, street traffic to overall traffic in a city, or residential to global electrical load forecasting. Local forecasting provides greater details, but is also problematic due to the computational cost increase, and sometimes harder to perform accurately

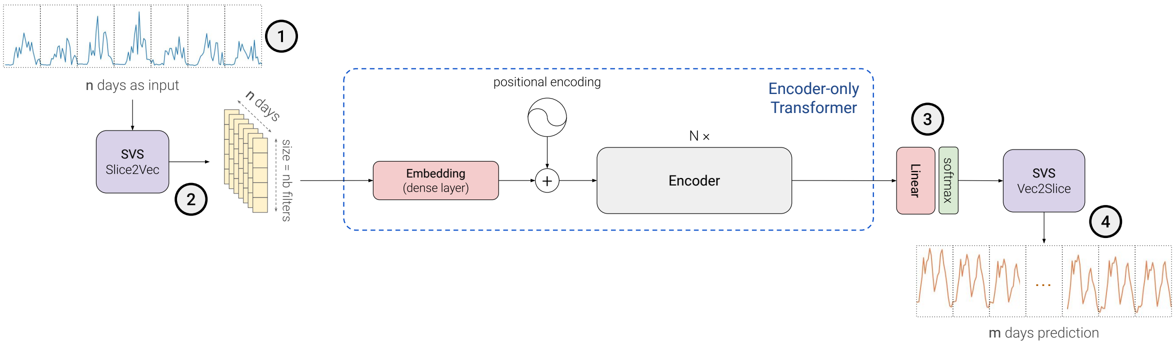


Figure 3.1: Simple, yet efficient architecture of Transplit: (1) the input time series is split into seasons; (2) Slice2Vec converts each cycle into a vector; (3) the transformer forecasts vectors (4) which are converted back to a time series with Vec2Slice.

due to unpredictable events — for example, a consumer finally decides to cook him/herself instead of ordering food, changing the local consumption time series. In other terms, at the local level, more data is involved and models need to constantly take change in consumption into account. At higher granularity, however, time series are smoothed and the effect of deviation in individual behavior is less noticeable.

In this regards, the last category of approaches presented in section 3.2 is very promising due to its single model design which significantly reduces the amount of models to train. Still, these models are not entirely applicable to seasonal forecasting as they require high computation power and time to train, which is problematic when models need to be retrained often.

There is therefore a need for lighter models, requiring less computational power and time, with at least similar performances. With this in mind, we introduce Transplit, a lightweight load forecasting model. To design it, we started from the observation that recent models are processing time series one value at a time, *i.e.* a sequence of n values results in an inner representation of n vectors to process. Our solution then consists in reasoning in processing blocks of seasons (e.g. consumption days) rather than single values.

3.3.2 Transplit

From these observations, we propose *Transplit*, along with a seasonal encoder and decoder. Transplit is also an encoder-only forecasting model based on the original transformer [VSP+17], then being composed of multi-head attention layers with a simple full attention mechanism. The architecture of the model is presented in figure 3.1.

3.3.3 Season tokenization

The architecture introduces a novel approach to load forecasting by processing the time series data in meaningful segments. Instead of analyzing individual data points, the method divides the load curve into fixed-period slices. These slices are then matched against a dictionary of characteristic curve patterns, enabling their conversion into numerical vectors. This dictionary vectors can be assimilated to tokens, although the encoding process in itself doesn't correspond to tokenization, as the result is a combined sum of the different vectors of the dictionary.

The forecasting process operates by predicting a weighted combination of these characteristic patterns to generate each future slice of the load curve. Multiple slices are sequentially predicted and joined together to construct the complete load forecast. This segmentation-based approach allows the model to capture and reproduce complex temporal patterns while maintaining computational efficiency, and aims to be much more suited for seasonal datasets than embedding them into arbitrary-sized and overlapping patches.

Before presenting our model itself, we introduce a new module called Slice to Vector to Slice (SVS), composed of two parts:

- **Slice2Vec**: A layer placed at the very beginning of the model, that splits the input time series into many slices of a fixed size, and then convert them into vectors;
- **Vec2Slice**: Another layer, placed at the very end of the model, that converts back the output vectors into slices of time series, to finally concatenate them.

Simpler processing for seasonal time series. The key concept of SVS for load forecasting is to recognize typical seasons from a wide range of profiles, and to embed all the necessary information of one cycle into one vector.
SVS also allows for the model to capture temporal dependencies within each slice, while still maintaining the ability to capture long-term dependencies across different slices. This is achieved through the use of convolutional layers in Slice2Vec, which can capture local patterns within each slice, and attention layers in the main model, which can capture global patterns across different slices. As opposed to patches [NNS$^+$23], slices don't overlap and enable a coherent *per-season* inner representation of the time series.

Slice2Vec

As described in figure 3.2, Slice2Vec takes a sequence $X \in \mathbb{R}^L$, where $L = m \times T$ is the input sequence length which is a multiple of a period T. X is split into m periods of size T, that we denote $(X^k)_{k\in\{1,\dots,m\}}$[1].

---

[1] Concretely, if the period T is one day, then X^k is the k^{th} day of the input.

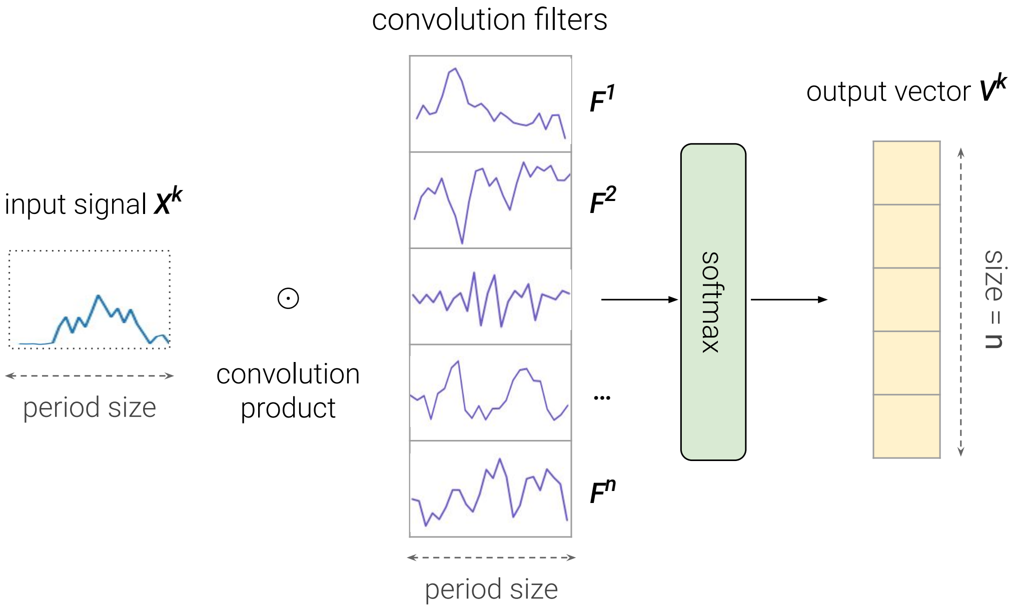


Figure 3.2: The Slice2Vec process

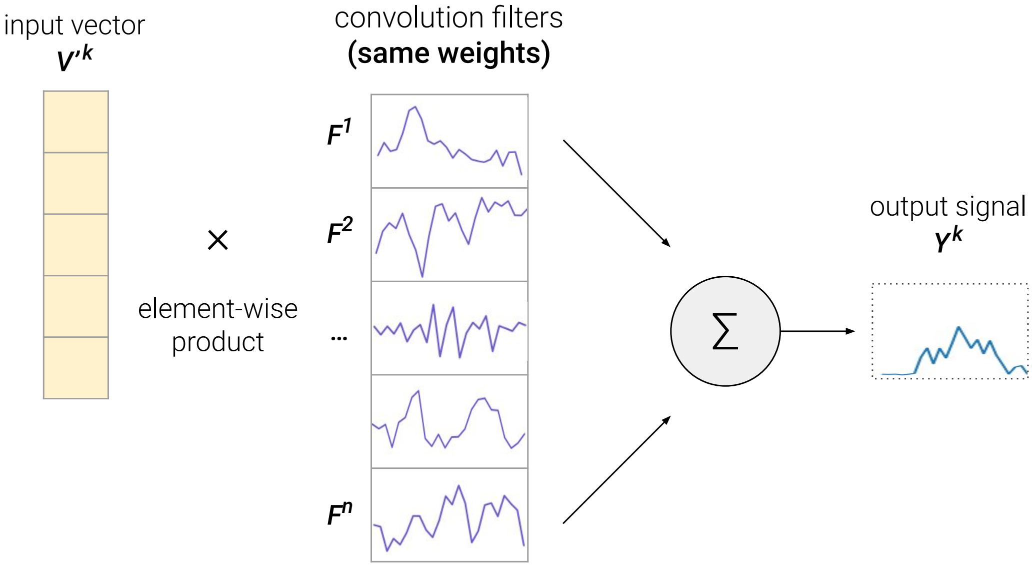


Figure 3.3: The Vec2Slice process

For each X^k, we define a vector V^k with the following convolution product:

$$\forall i \in \{1, \dots, n\}, V_i^k = X^k \odot F^i \tag{3.1}$$

where n is the chosen size for the vectors, V_i^k is the i^{th} coefficient of V^k, and F^i

is a convolution filter of size T. Since X^k and F^i have the same size, the convolution product is also equivalent to the sum of their term-wise product. The filters are modified during the training phase with gradient descent, so that they match the shape of typical days of consumption.

The idea is then to obtain a *vocabulary* of different profiles of curves during one period, and to construct one vector based on which elements of the vocabulary are most similar with the input slice. This therefore allows the model to "*reason*" in terms of entire days instead of individual values.

We should also note that the SVS module shares some similarities with VQ-VAEs [VV+17a], as both employ a *code book* to represent data. However, the SVS approach does not involve quantization. Its code book is constructed by extracting the weights of a convolutional layer, and comprises a variety of distinct raw shapes that are utilized for time series decomposition – in contrast to VQ-VAEs, which store semantic vectors.

Vec2Slice

In order to transform a vector V'^k back in a T-sized slice of time series called Y^k, the filters F^i are weighted according to the coefficients of V'^k, then summed:

$$Y^k = \sum_i V_i'^k F^i \tag{3.2}$$

as illustrated in figure 3.3. The slices Y^k are concatenated to form the output Y. It is important to underline that Vec2Slice *is not* the inverse function of Slice2Vec, *i.e.* Vec2Slice(Slice2Vec(X)) $\neq X$.

Vec2Slice shares the filters weights with Slice2Vec. The filters are thus used twice: to recognize the input and to match the shape of the expected output.

As an example, in the use case of load forecasting, the period T for SVS (Slice to Vector to Slice) is set to 1 day, so that the model can reason in terms of days. The input of the decoder is a sequence repeating the same vector, corresponding to the average of the input days embeddings — semantically representing the average typical day of the input.
Taking this as a starting point, the role of the decoder is then to alter each day of the sequence to make the prediction, considering the context from the encoder.

Multivariate time series embedding

The SVS module essentially consists in one convolutional layer with a kernel size and a stride as large as the season length. The number of input channels of this convolutional layer is adapted to consider multivariate time series, i.e. the shape of the weights is set to $(n \times L \times C)$, where C is the number of input and output channels. Hence, each channel of the time series is associated with n distinct filters.

3.4 Evaluation

3.4.1 Research Questions

Transplit is designed to be lighter and faster than state-of-the-art solutions for medium- and long-term forecasting, while maintaining their performance. As frugality is the main purpose of Transplit, we start by investigating **RQ1:** How fast can Transplit be optimized on seasonal data and how does it compare to existing solutions? Transplit also needs to be on par with existing solutions regarding accuracy (**RQ2**): can Transplit maintain state-of-the-art performances?

Given the Transplit's focus on seasonality, we also explore wether Transplit could be extended to non-seasonal datasets, or if this represents a fair limitation of the model (**RQ3**).

Finally, we will study its applicability to bigger CREOS dataset, corresponding to a specific domain (**RQ4**): how does Transplit perform on electrical consumption forecasting?

3.4.2 Models

The aforementioned state-of-the-art approaches are time series transformers [WZZ+22]: Autoformer [whXW+21], Informer [ZZP+21], Reformer [KKL20] and a vanilla Transformer architecture [VSP+17]. Classical models such as LSTMs are disregarded as it has been shown that transformers perform better for load forecasting [whXW+21][ZZP+21][KKL20]. All of these approaches fall into the same category as Transplit, *i.e.* one global model for all users.
Although other approaches that build one model per household could also have been evaluated, their cost (in terms of resources) far outweigh their supposed gain in accuracy, rendering them impractical to compare with.

3.4.3 Data

To answer our research questions, we evaluate those approaches on 2 benchmarks:

- The first is composed of five datasets, commonly used to compare time series model performance [SAM+22][whXW+21][ZZP+21][KKL20]:
  - **Weather**: This dataset contains 21 meteorological indicators, such as air temperature, humidity, etc.[2] The data is recorded every 10 minutes, for the whole year of 2020. T is set to 144 to reflect the daily seasonality.
  - **Traffic**: This dataset contains hourly measurements of traffic volume on freeways over a period of 4 years[3]. The rate is 24 measures per day. As one day again typically represents one cycle, T is set to 24.

---

[2] Max Plank Institut – `https://www.bgc-jena.mpg.de/wetter/`

[3] California Department of Transportation, San Francisco – `https://pems.dot.ca.gov/`

  - **Electricity**: We also evaluate Transplit on the electricity dataset[4]. The electricity dataset is an open dataset frequently used in the evaluation of load forecasting approach and is composed of 321 households hourly load over 3 years (2012-2014). T is set to 24 to reflect the daily consumption cycles.
  - **ETTm2**: Electricty Transformer Temperature, a critical indicator for power grid management, collected from two counties in China [ZZP+21]. As for the previous, since electrical consumption patterns typically exhibit daily cycles, we choose the day as the seasonality period ($T = 96$).
  - **Exchange**: This dataset contains daily exchange rates between eight different currencies over a period of 10 years [LCY+18]. The rate is of 1 measurement per day, and T is then set to 7, although it doesn't have any clear seasonality.
- The second focuses on electricity consumption forecasting at scale, corresponding to a practical application of Transplit:
  - One dataset comes from CREOS and is composed of the hourly load of 6010 households over 2 years (2020-2021) within Luxembourg. We refer to this dataset in the following as **CREOS**.
  - For reproducibility purposes, we also evaluate Transplit on the aforementioned **electricity** dataset.

3.4.4 Experimental settings

General configuration. We proceed to experiments by first standardizing the values for each time series channel. Each dataset is then split into training, validation and testing set with a ratio of 70/10/20%. To provide the model with relevant inputs and expected outputs, we decomposed those sets into pairs of consecutive months (2 * 720 values). The first month is given to the model as a context, and the second month is the ground truth consumption to predict. Each sequence passed to the model always start at midnight. Exceptionally for **exchange**, due to the small size of the dataset, the input and output length are set to 168.

We configure all baseline approaches according to the recommendation from their authors. Regarding Transplit, we:

- set 256 filters for the SVS module and the value of T accordingly with the dataset being used
- use 1 encoder and 1 decoder, with a vector size (d_{model}) of 84

All models are trained with respect to the **MSE** (Mean Squared Error) loss. and **MAE** (Mean Absolute Error) is used as an additional metric.

---

[4] Electricity Consumption Load – `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`

Case study. Nevertheless, these metrics hardly reflect the accuracy of consumption peaks prediction. To better compare these predictions, we also compute the MAE on local maxima above the time series' average value. We consider a point y_t as a peak if $y_{t-1} < y_t$ and $y_t > y_{t+1}$ and $y_t > \text{avg}(y)$. We denote the MAE on these specific points **PMAE** (Peak Mean Absolute Error). We disregard the MAPE (Mean Absolute Percentage Error), that divides the error by the expected value to measure a relative error. In our case, this metric is not fitted when the expected value can be 0, which can happen when forecasting residential consumption, but also road traffic and weather attributes such as wind speed. In electrical consumption applications, this metric can, however, be used for the overall load forecasting, *e.g.* at the scale of a neighborhood, where the total consumption is unlikely to be zero.

On error scales. Recent time series literature preprocesses dataset by applying standardization over time series channels, and directly compares errors — typically MSE and MAE — on these standardized values. While we maintain this practice for comparison purposes on the dataset benchmark, we reestablish the output's original scale in the testing phase in the case study. This allows to express the MAE in kWh, and also enables more realistic losses, as standardized values give as much importance to very small consumers — where the consumption is sometimes near zero — as heavy consumption users.

Training. The training is performed similarly to previously suggested approaches [ZZP$^+$21][whXW$^+$21] using the Adam optimizer [KB15] with a learning rate of 10^{-4}, divided by 2 at every epoch. All experiments are run on an NVIDIA RTX A2000 4GB with a batch size of 32. The CPU used for time measurement is an Intel i7 11th generation, and the code is available on our GitHub repository[5].

3.5 Results

3.5.1 Speed and lightness

Speed. The key element of Transplit is its lightness: what is important to distinguish here is the time taken to train these models, shown in table 3.1. We observe that Transplit is consistently 3 times faster than NHits, for all datasets, and 300 times faster than the Performer. This can be explained by many factors. The first one is the length of sequence manipulated by the encoder and decoder for each model. Whereas the other models (except NHits and PatchTST) process 720 values as 720 vectors, our model only processes this information with T times less vectors, resulting in a much shorter sequence of length 30 if $T = 24$. NHits and PatchTST are exceptions, as they fully or partially embed time series along the time axis, enabling them to be more efficient than other models.

---

[5] `https://github.com/serval-uni-lu/transplit-framework`

	# of Parameters	CREOS	Electricity	Weather	Traffic	ETTm2	Exchange
Transplit	129,024	8m 8s	45.0s	6.2s	1m 20s	2.98s	0.39s
NHits	3,421,082	24m 41s	2m 23s	19.6s	4m 16s	9.47s	1.25s
PatchTST	4,254,738	56m	4m	32.9s	7m 9s	15.8s	2.11s
FEDformer	105,691,433	1d 16h 12m	4h 7m	33m 50s	7h 21m 23s	16m 21s	2m 10s
Performer	2,104,833	7h 15m	36m 3s	4m 56s	1h 4m 25s	2m 23s	18.9s
Autoformer	10,505,217	1d 12h 15m	4h 16m	35m 4s	7h 37m 28s	16m 57s	2m 15s
Informer	11,306,497	19h 37m	1h 53m	15m 29s	3h 21m 56s	7m 29s	59.4s
Reformer	5,782,529	20h 56m	4h 13m	34m 40s	7h 32m 7s	16m 45s	2m 13s
Transformer	10,518,529	1d 9h	2h 51m	23m 26s	5h 5m 35s	11m 19s	1m 30s

Table 3.1: Number of trainable parameters and **GPU** training time for each model and dataset used, for a 720 → 720 values forecast.

	CREOS	Electricity	Weather	Traffic	ETTm2	Exchange
Transplit	28m 10s	2m 47s	22.9s	4m 58s	11.0s	1.46s
NHits	1h 27m	7m 6s	58.4s	12m 41s	28.2s	3.74s
PatchTST	2h 36m	14m 24s	1m 58s	25m 44s	57.2s	7.58s
FEDformer	28d 4h	1d 18h 30m	5h 49m 19s	3d 3h 56m 51s	2h 48m 52s	22m 22s
Performer	5d 10h 36m	11h 59m	1h 38m 30s	21h 24m 51s	47m 37s	6m 18s
Autoformer	20d 21h 59m	1d 18h 1m	5h 45m 20s	3d 3h 5m 2s	2h 46m 57s	22m 7s
Informer	8d 13h 21m	17h 11m	2h 21m 14s	1d 6h 42m 24s	1h 8m 17s	9m 3s
Reformer	18d 19h 47m	1d 13h 49m	5h 10m 49s	2d 19h 34m 42s	2h 30m 16s	19m 54s
Transformer	13d 10h 42m	1d 3h 1m	3h 42m 3s	2d 16m 44s	1h 47m 21s	14m 13s

Table 3.2: **CPU** training time for each model and dataset used, for a 720 → 720 values forecast.

Memory efficiency. Nevertheless, Transplit also has significantly less parameters to train (Table 3.1), which contributes to reduce time and required memory to train the model. Since the model is lighter, less training is required: only one epoch was sufficient to obtain the best results, vs. two epochs of training for the others.

CPU performance. Another big advantage for Transplit is that using a CPU instead of the GPU doesn't significantly slow down the model's computation time (table 3.2), as processing smaller sequences implies smaller operations, and data still has to be cached in the GPU, which can take a lot of time in total. Only using a CPU, this makes Transplit **3 times** faster than NHits (second-fastest model), and **1440 times** faster than the Performer (slowest model), the latter taking more than 28 days to be trained on the CREOS dataset, whereas Transplit only needs 28 minutes (table 3.2). The non-necessity of GPU to efficiently run the model allows it to be used on less expensive infrastructures, while being faster and conserving good forecasting performances.

3.5.2 Accuracy

	weather		**traffic**		**electricity**		**ETTm2**		**exchange**	
Model	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Informer	1.187	0.797	1.794	0.911	1.069	0.824	0.505	0.514	0.505	0.514
Reformer	0.895	0.729	1.355	0.654	0.842	0.683	1.475	0.907	1.475	0.907
FEDformer	0.629	0.577	0.617	0.377	0.247	0.361	0.247	0.345	0.247	0.345
Performer	0.492	0.509	1.142	0.572	1.030	0.775	0.430	0.491	0.430	0.491
Autoformer	0.409	0.428	0.687	0.425	0.278	0.377	0.276	0.366	0.276	0.366
NHits	0.331	0.370	0.621	0.374	0.233	0.325	**0.235**	**0.334**	**0.235**	0.334
PatchTST	**0.322**	**0.336**	0.589	0.394	0.248	0.355	0.415	0.428	0.251	**0.319**
Transplit	0.349	0.410	**0.535**	**0.288**	**0.213**	**0.326**	0.286	0.399	0.389	0.512

Table 3.3: Performances on seasonal datasets for 1 week forecasting

Now that speed and ligthness has been assessed, it is essential to verify that the Transplit's architecture does not affect performance in terms of accuracy.

In table 5.1 we compare the performance of Transplit with a benchmark of general datasets. We report the average standardized performance metrics (MSE and MAE) for 1-week forecasting over 5 experiments on the previously cited datasets. Transplit performs competitively on most of the seasonal datasets compared to the other models, with the exception of the `exchange` dataset.

For seasonal datasets (`weather`, `traffic`, `electricity` and `ETTm2`), Transplit achieves MSE and MAE values that are comparable to or better than transformer-based models such as Informer, Reformer, or FEDformer. This indicates that Transplit is well-suited for capturing seasonal patterns in this type of time series data, while keeping a strong advantage on speed and being lightweight.

However, on the `exchange` dataset, Transplit has a noticeably higher MSE (+65%) and MAE (+60%) compared to the best models. This is likely because the `exchange` data does not exhibit clear seasonality, which is what Transplit is optimized to recognize.

Following this empirical demonstration of the strengths and limitations of the Transplit approach depending on the characteristics of the input time series, we conclude that while optimizing Transplit for fast and lightweight performance on seasonal data, it comes at the cost of reduced accuracy on non-seasonal datasets like exchange rates.

3.5.3 Case study

Back on tables 3.1 and 3.2: on the much bigger industrial dataset, we observe that Transplit took 8 minutes to learn the 6010 consumers profiles of the CREOS

	electricity					
	7 days			**30 days**		
	MSE	MAE	PMAE	MSE	MAE	PMAE
Autoformer	1.09	0.235	0.398	1.22	0.241	0.408
Informer	4.02	0.481	0.566	3.93	0.478	0.578
Performer	1.09	0.252	0.396	1.13	0.260	0.411
Reformer	2.54	0.395	0.456	2.45	0.392	0.451
Transformer	2.11	0.305	0.452	2.20	0.331	0.468
NHits	0.557	0.351	0.412	0.748	0.402	0.635
PatchTST	**0.484**	**0.140**	0.369	0.697	0.182	0.392
Transplit	0.515	0.162	**0.340**	**0.607**	**0.171**	**0.381**

	CREOS					
	7 days			**30 days**		
	MSE	MAE	PMAE	MSE	MAE	PMAE
Autoformer	0.185	0.210	0.405	0.204	0.217	**0.429**
Informer	0.201	0.213	0.450	0.226	0.231	0.462
Performer	0.225	0.230	0.562	0.248	0.244	0.601
Reformer	0.172	0.185	0.414	0.205	0.207	0.443
Transformer	0.177	0.191	0.423	0.203	0.207	0.449
NHits	0.170	0.191	0.393	0.200	0.209	0.431
PatchTST	0.170	**0.179**	0.585	0.204	**0.196**	0.664
Transplit	**0.169**	0.183	**0.379**	**0.191**	0.200	0.430

Table 3.4: Case study: Error metrics for the electricity and CREOS datasets (kWh scale)

dataset using a GPU, and 28 minutes on CPU — being only 4 minutes more than NHits on GPU.

In table 5.1, Transplit successfully manages being on par with PatchTST for all error metrics. Overall, there is a clear performance gap between medium-term (weekly) and long-term (monthly) predictions for all models.

Bringing the original scale back also highlights difference performances among all models in general when looking at the `electricity` dataset. For example, while NHits performed better than PatchTST on the standardized scale, their rank is reversed with the original scale.

Regarding the CREOS dataset, we notice that all models perform similarly and manage to predict the correct consumption in average to the nearest 0.2 kW (MAE).

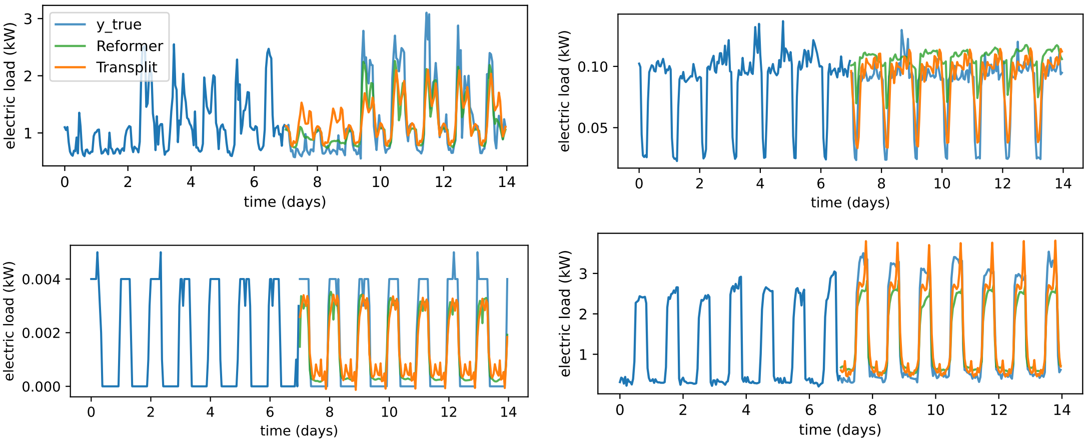


Figure 3.4: Predictions of Transplit (orange) and Reformer (green), showing that both models are able to recognize various profiles at different scales, while being trained only once.

We eventually measure the peak MAE and notice that two different models rank as first for this task in the two datasets: Transplit seems the most adapted model for the electricity dataset, while PatchTST manages to get a better MAE on the CREOS dataset. Transplit nonetheless consistently conserves good performances, in line with other models, in both cases.

Finally, figure 3.4 shows examples of predictions for one week on the CREOS dataset and compares it with the Reformer [KKL20], *i.e.* the model that shows the best results with Transplit (Table 5.1).

3.6 Optimizing Transplit

In this section, we discuss the possibilities of Transplit (RQ3), regarding the choice of seasonality hyperparameter, the impact of different input sequence lengths, and custom losses that could help peak prediction for time series.

3.6.1 The seasonality hyperparameter

Transplit is designed for seasonal time series forecasting. In this section, we explore the impact of the seasonality hyperparameter and highlight its importance. In each seasonal dataset previously cited, it appears that seasonality is easy to identify when analyzing the plots and knowing the context. It also appears that this seasonality corresponds to the highest peak amplitude in a simple Fast-Fourier Transform showing that this hyperparameter can be automatically determined.

Severe performance drops can occur when changing the Transplit's seasonality. Figure 3.5 indeed shows that test loss is generally better when seasonality corresponds to periods with important amplitudes in Fourier decomposition. The experiment conditions are the same as described in the previous sections and repeated 5 times (720-values forecast, 70% / 10% / 20% dataset split).

3.6.2 Optimal sequence length

One of the main advantages of Transplit is the following: Since the sequence length processed by the transformer part is significantly shortened, there is room for larger inputs. We can then reasonably provide larger input to give more context to the model. This task is more difficult on classical transformer models, since large inputs result in very long sequences to process, often taking too much resources, especially to train, if we don't reduce the batch size.

We proceed to another experiment where the number of days given to Transplit is variable during the training stage. More specifically, given a date, instead of giving a fixed sequence length as input to the model, we provide to Transplit the entire consumption history available in the dataset. To do so, we create batches aligned on the same date, so that every item of the batch has the same shape and can be stacked together.

Figure 3.6 shows the performance of Transplit by varying the input sequence length. Again, Transplit is only trained once, but with random input sizes during the training. The experiment is repeated 5 times in order to avoid randomness in the results.

We observe a significant drop of forecasting performance starting from 6216 samples given in input for the *electricity* dataset – which corresponds to precisely 37 weeks (8 months) – thus confirming the existence of a limit from which the quantity of information provided to the model brings more noise than useful information. An interesting property is that the MAE curves show periodical waves that tend to flatten with the sequence length. We measure that this period is of 6 weeks Since Transplit makes one vectorial representation for each day, this means that the sequence processed internally has a length of 6216 / 24 = 259.

The same conclusion can be taken for the *traffic* dataset, which presents a different performance curve shape. The optimal input length is 168, corresponding to 1 week of past data. With further data, the MSE slowly starts to raise, as well as the MAE, though it tends to stabilize.

Computation time impact

Increasing the input sequence length also impacts the computation time. Transplit has a full attention mechanism, meaning that self-attention in its encoder has a $\mathcal{O}(n^2)$ complexity. Figure 3.8 shows the average time taken by Transplit to forecast 1 week from the electricity dataset, in 5 experiments. We use the same GPU as described in the experimental setting (3.4.4) and a constant batch size of 64. The computation time first grows linearly, before rapidly doubling when the sequence length becomes higher than 4096.

3.6.3 Loss trade-off

In the case of electrical consumption, one key subtask of time series forecasting is *peak forecasting*. To measure their performance, we then measure their PMAE and PMSE, as defined in subsection 3.4.4. The graph shows the trade-off between optimizing for overall mean squared error (MSE) across all time points versus optimizing specifically for peak mean squared error (PMSE). The x-axis represents different loss functions, ranging from 100% weight on MSE at the left extreme to 100% weight on PMSE at the right extreme. Intermediate points represent weighted combinations of MSE and PMSE.

3.7 Discussion

Although Transplit excels at handling seasonal time series, it may not perform as well when the data lacks clear seasonal patterns. This limitation should be considered when applying the model to non-seasonal datasets.

Exploiting seasonality is nonetheless a strength: we have seen that Transplit and its SVS module splitting the consumption curve makes the process lighter, faster, and allows extensive sequence length computation at a low cost, although there is no general optimal sequence length – this hyperparameter depending on the dataset.

Overall, Transplit offers a fast and effective solution for seasonal time series forecasting across various domains. Its ability to handle different sequence lengths and automatically adjust the seasonal parameter makes it a flexible tool for practitioners and researchers alike. One important application may include energy, where inference and retraining might be performed frequently at a large scale, in order to predict a detailed state of the grid in terms of electric load. However, as with any model, it is essential to carefully evaluate its suitability for the specific problem at hand and consider potential limitations.

Other seasonalities could be taken into account, such as weeks and years. For example, weekly seasonality has indeed to be learned by the model in the case of electricity load forecasting.

Conclusion

We have seen that Transplit remains on par with state-of-the-art time series deep learning models, that are able to generalize time series profiles, allowing detailed predictions at scale, while being several hundreds of times faster and lighter. Its architecture allows it to be generalized to seasonal time series in a wide range of applications; thus, this opens up the possibility to run a deep-learning predictive model globally at a lower cost, as only one CPU is enough to keep

Transplit efficient, making it possible to run it locally, as well as the possibility to update the model regularly with new data.

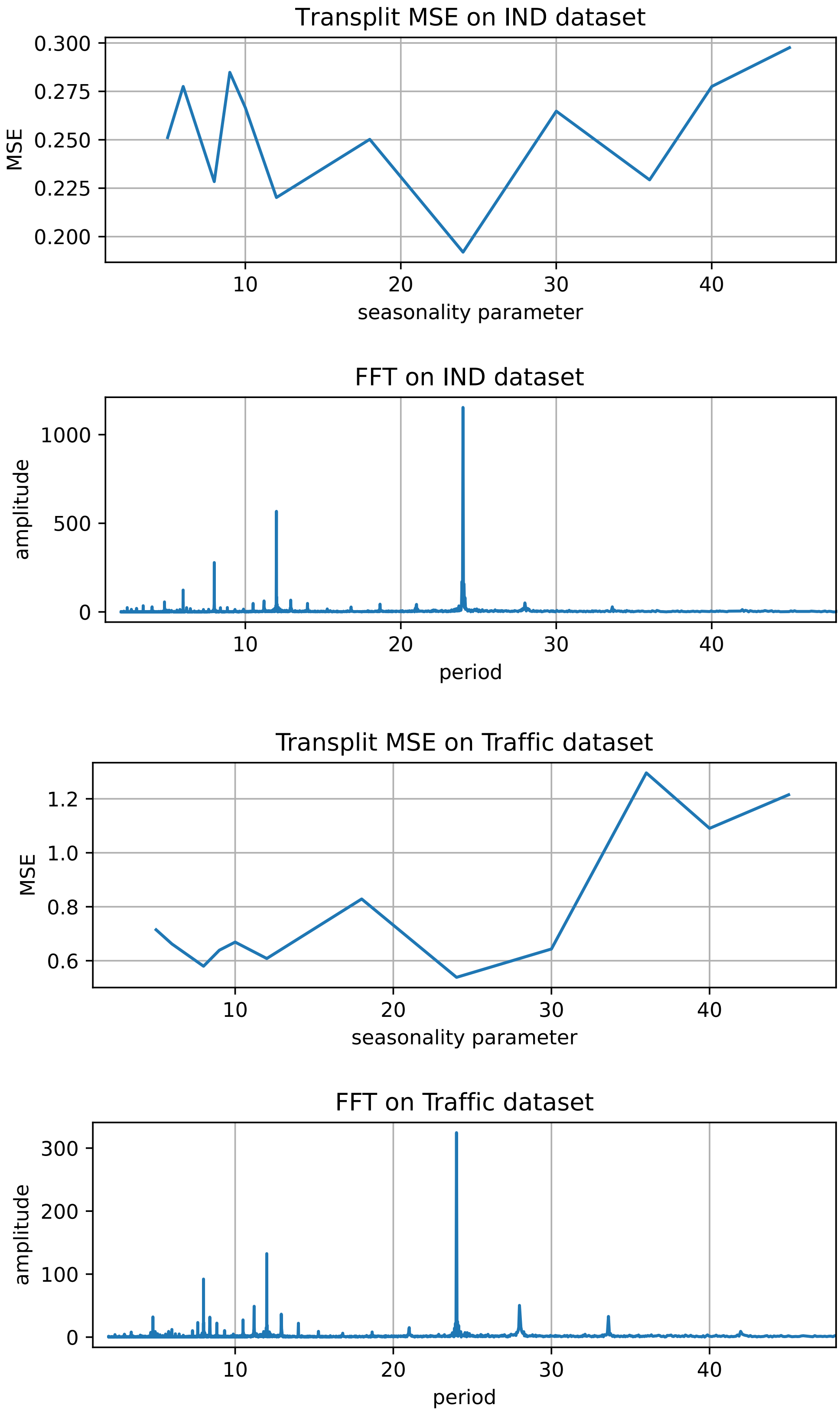


Figure 3.5: Test loss variation when varying the seasonality parameter on the CREOS and Traffic datasets. The Fourier decomposition is shown below as reference (period on the x-axis).

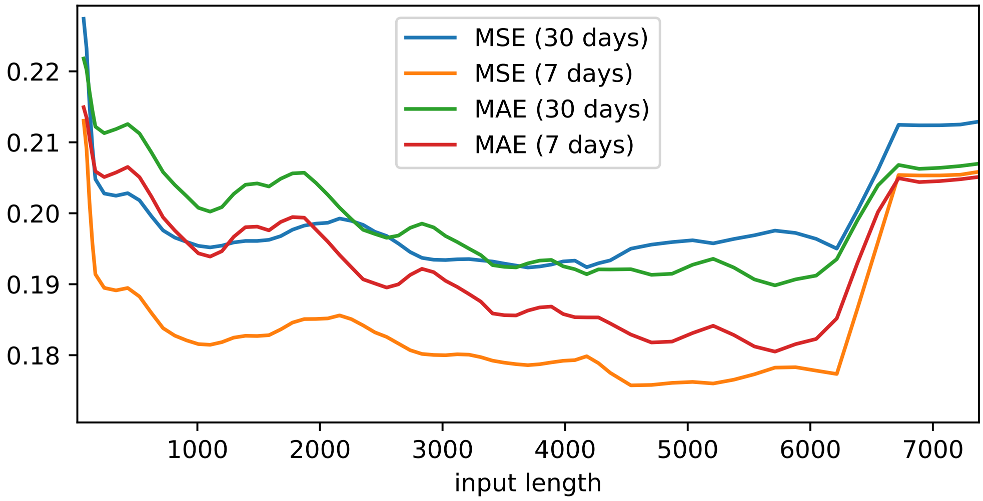


Figure 3.6: Transplit performance on the *electricity* dataset, w.r.t. sequence length provided in input

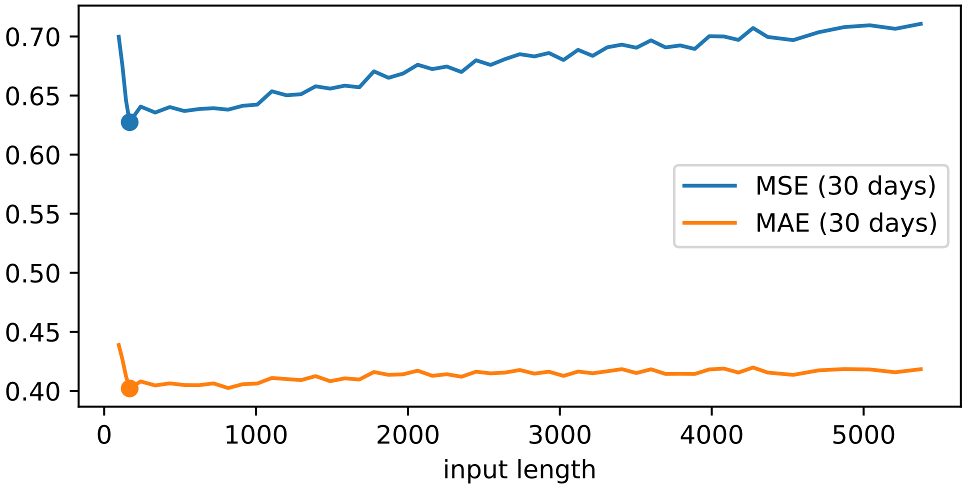


Figure 3.7: Transplit performance on the *traffic* dataset, w.r.t. sequence length provided in input

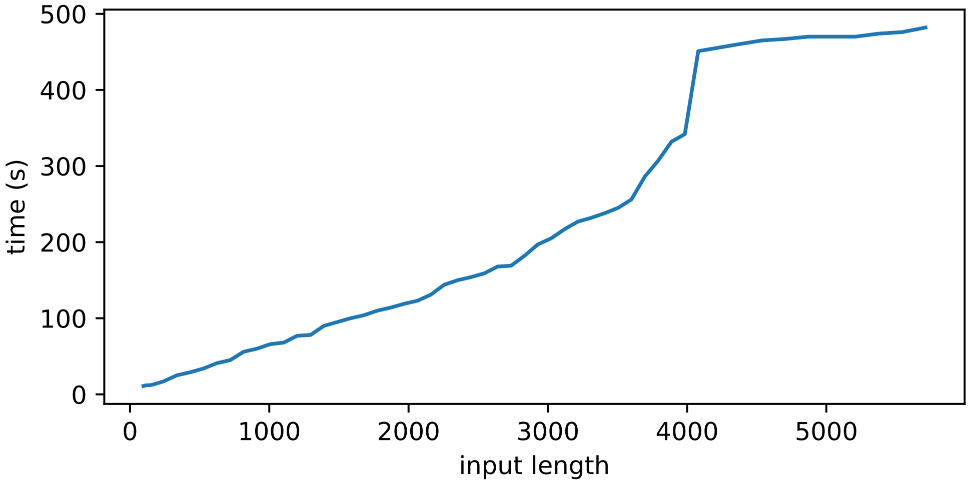


Figure 3.8: Average time taken by Transplit to forecast 1 week from different input sequence lengths.

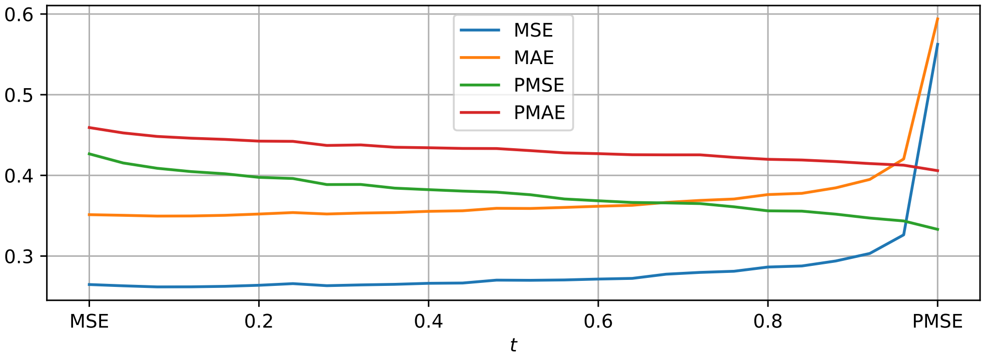


Figure 3.9: The model is trained on the electricity dataset, with a loss being a mix of a range of trade-offs between MSE and PMSE. $t = 0$ means that Transplit is only trained on the MSE loss, $t = 1$ means that the loss is fully defined by the PMSE. The variation of MSE, MAE, PMSE and PMAE is reported.

4

Peak Forecasting Enhancement with Deblurring Diffusion Models

This chapter introduces DeDiPeak, a framework that addresses the challenge of accurately forecasting peaks in time series data by providing two new evaluation indicators ($P3_{\mathcal{E}}$ and $P3_{sw}$) and implementing a deblurring diffusion model to enhance peak prediction without sacrificing overall forecasting accuracy. The framework aims to overcome the limitations of traditional error metrics (MSE, MAE) that tend to flatten peaks, and empirical validation shows improved performance compared to existing state-of-the-art approaches in peak forecasting tasks.

Contents

4.1 Context 32
4.2 A small history with early experiments 35
4.3 Background and Related Work 39
4.4 DeDiPeak 46
4.5 Experiments 50
4.6 Details and additional results 55
4.7 Discussion 68

4.1 Context

4.1.1 Transplit limitations

Now that we have seen that Transplit efficiently produce qualitative forecasts, let us have a look at some examples in Figure 4.1.

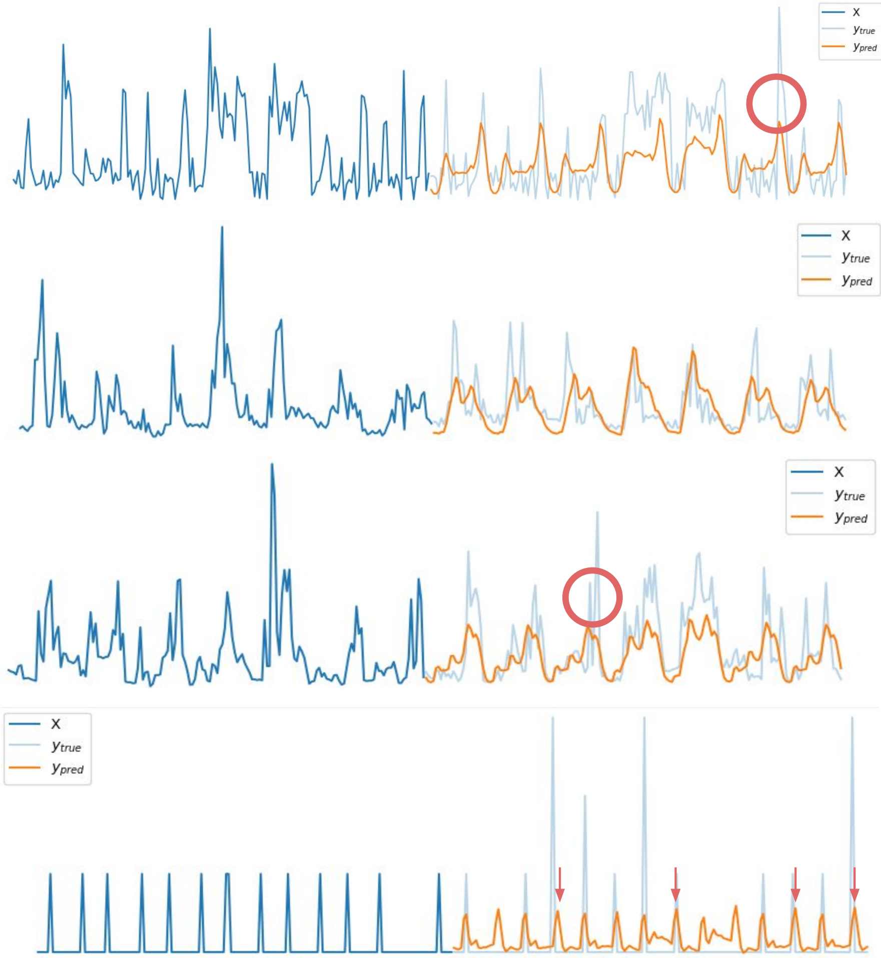


Figure 4.1: Looks like Transplit fits well with different consumption patterns. That's great. But what about peak amplitudes (highlighted in red)? Even the obvious ones (bottom)? That's exactly what this chapter is about.

While Transplit is able to adapt to different consumption patterns, it systematically fails in predicting consumption peaks.

Disclaimer. Let us keep in mind that these are household-level consumption patterns. We cannot know if a person will find the strength to start his washing machine now or if they are too lazy today to gather their clothes. Maybe in the evening? Perhaps even this person doesn't know. At the household level, there is an important degree of uncertainty that makes impossible perfect forecasts, especially with peak timings. But if there is something that we almost surely know, it is that this person will sooner or later start his washing machine, as they will not stay wearing dirty clothes forever. And this machine will contribute to a consumption peak we have already seen in the past — a small pattern over 1 to 3 hours we can find in the historical consumption. Similar logic can be thought for cooking, vacuum cleaner, etc.

We can therefore grant to Transplit that given the uncertainty of peaks, it might be more optimal to *play it safe* by predicting "shy" peaks only, as their timing and amplitude is uncertain. However, even when the peak timing and amplitude are evident (last plot of Figure 4.1), Transplit fails in predicting the right amplitude. In a general manner, the peaks look *smoothed*.

4.1.2 MSE loss: an already-known culprit

Over the past decades, increasingly accurate models have been continuously designed. A trend that has been accelerated considerably by recent advances in neural network architectures [ZM98; MM18], with many promising approaches based on LSTMs [YSL+19; LCY+18], convolutional neural networks (CNN) [MS22; LPP20], transformers [VSP+17; whXW+21; ZZP+21], and, more recently, diffusion models [RSS+21] being suggested. In parallel, benchmarks and metrics facilitating the comparison between new and established methodologies have been introduced and standardized [SAM+22].

While this standardization fosters the development of the field, most standardized metrics, such as the Mean Squared Error (MSE), evaluate approaches exclusively on their ability to optimize the average prediction, which can prove sub-optimal for certain tasks, such as peak forecasting.

Peak forecasting can be defined as the accurate prediction of local maxima in a time series. It constitutes a critical task in many domains such as electricity grid management [DMD+21] or finance, where an accurate prediction of peaks enables better resource allocation, cost savings, and improved decision-making.

Figure 4.2 visualizes the misalignment of objectives in peak forecasting by illustrating the impact of slight variations in peak timing on the MSE, commonly used as a loss function in time series forecasting [ZZP+21]. In this example, while $\hat{y}_1$ is overall more similar to the ground truth and provides a more accurate peak prediction compared to $\hat{y}_2$, its MSE is higher by a factor of two.

Consequently, models trained on the MSE will generally favour the forecast of *shy* peaks (smoothed, with low amplitude) rather than predicting a right-amplitude peak, as the latter option carries the risk of drastically increasing the loss in case of off timing. This preference for "smooth" results aligns with findings from image generation studies [DB16; ERO21; Bar19].

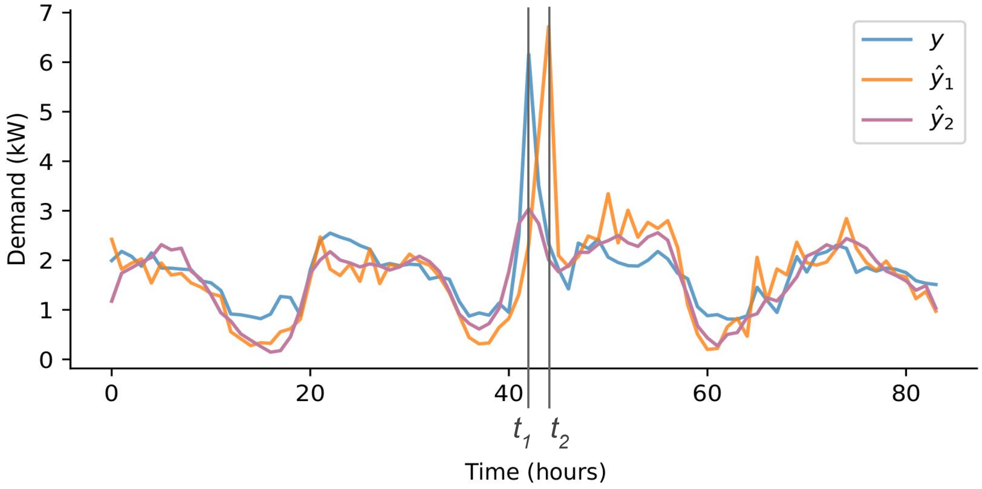


Figure 4.2: Example of a time series y and two different approximations $\hat{y}_1$ and $\hat{y}_2$, with an MSE of 0.62 and 0.26, respectively.

More generally, loss functions that compare time series element-wise (*e.g.* MAE, RMSE, MAPE, MSPE) strongly discourage the forecast $\hat{y}_1$ compared to $\hat{y}_2$, since $|y(t_1) - \hat{y}_1(t_1)|$ is high, as well as $|y(t_2) - \hat{y}_1(t_2)|$; whereas $|y(t_2) - \hat{y}_2(t_2)|$ is small. This does generally not align with real-world requirements as, typically, a slight offset in time is acceptable as long as the peak intensity is accurately predicted.

4.1.3 Dedipeak

In this work, we bring two new indicators to evaluate the peak forecasting capability as a combination of peak timing and intensity. Then, to improve peak forecasting, we propose enhancing the forecasts of existing approaches rather than offering a stand-alone approach. To this end, we suggest a novel method that leverages the final stages of deblurring diffusion models [RHS22; HS22b]. During inference, these models generate time series by progressively adding finer details to the signals, the last steps of which can be interpreted as a peak enhancement task. To the best of our knowledge, these models have not yet been applied to time series forecasting.

In this chapter, we introduce *DeDiPeak*, a framework for peak prediction that integrates:

1. Two performance indicators $P3_{sw}$ and $P3_{\mathcal{E}}$ specifically designed to assess peak prediction performance through sliding windows and Euclidean distance;

2. The first time-series deblurring diffusion model approach enhancing peak

forecasting capabilities by improving *shy* peaks generated by state-of-the-art approaches.

We empirically validate *DeDiPeak* by initially assessing the peak forecasting performance of state-of-the-art time series forecasting methods using our indicators. Subsequently, we demonstrate the advantages of applying our deblurring diffusion models to these methods. Finally, we compare *DeDiPeak* to previously proposed approaches aimed at improving peak forecasting, such as the Extreme Value Loss or DILATE Loss. Overall, we demonstrate that *DeDiPeak* enhances peak prediction performance by up to 36%, in almost every case (96%), on our performance indicators, regardless of the base models and datasets.

4.2 A small history with early experiments

To better understand how we ended up with a diffusion refinement model, let us introduce how we came from Transplit to Dedipeak.

4.2.1 Analogy with image generation

The phenomenon of blurry predictions resulting from MSE optimization is not unique to time series forecasting. This fundamental limitation has already been encountered and addressed in the computer vision community, particularly in image generation tasks, which offers to us valuable insights for temporal modeling. Lessons from computer vision are particularly interesting, as images are signals — just like time series. Images are merely two-dimensional, while time series are one-dimensional. Let us dive in this context first.

Autoencoders

The blurring artifact induced by MSE loss can first be observed in autoencoder architectures for image reconstruction. When trained to minimize pixel-wise MSE, these models systematically produced overly smooth, blurred reconstructions that failed to capture fine-grained details and sharp transitions — a phenomenon analyzed, among others, by [PKD$^+$16].

Several alternative loss functions have been proposed to mitigate this issue, including perceptual losses [JAF16], adaptive loss functions [Bar19], and various forms of adversarial training objectives. However, these early approaches provided only incremental improvements while introducing additional computational complexity.

GANs

A paradigm shift occurred with the introduction of Generative Adversarial Networks (GANs) [GPM$^+$14] — rather than directly optimizing MSE or similar pixel-wise losses, GANs train a generator network to fool a discriminator through an adversarial objective. This approach effectively bypasses the averaging effect of MSE by encouraging the generator to produce samples from the actual data distribution rather than its mean.

VQ-VAEs

The next major advancement came with Vector Quantized Variational Autoencoders (VQ-VAEs) [VV$^+$17b; RVV19], allowed to encode images into discrete token representations, converting the continuous pixel space into tokens, from discrete vocabulary. This token conversion allows generative models to train on the latent space instead of the pixel space directly, avoiding the classical drawbacks of the MSE loss. We therefore distinguish two stages: (1) VQ-VAE training and (2) generative model training.

This discrete tokenization approach proved highly effective, leading to the development of DALL-E by OpenAI [RPG$^+$21], which demonstrated high capabilities in text-to-image generation using a Transformer architecture [VSP$^+$17]. The key insight was that by discretizing the representation space, the model could avoid the averaging artifacts inherent in continuous optimization while leveraging the powerful sequence modeling capabilities of Transformers [ERO21; CRC$^+$20].

Diffusion

Most recently, diffusion models have emerged as the dominant paradigm for high-quality image generation, in particular denoising diffusion probabilistic models (DDPMs) [HJA20; SSK$^+$20; DN21a]. These models learn to gradually denoise random noise into coherent images through a series of denoising steps, effectively learning to reverse a forward diffusion process that gradually adds noise to real images.

Initially, diffusion models operated directly in pixel space [HJA20], which, while producing exceptional quality, suffered from significant computational overhead. Later, subsequent work demonstrated substantial efficiency gains by performing diffusion in learned latent spaces rather than raw pixel space [RBL$^+$22b]. Noteworthy examples include the (famous!) Stable Diffusion [RBL$^+$22b] and similar latent diffusion models, which achieve state-of-the-art results while reducing computational costs, thanks to the lower dimensionality of the diffusion space.

4.2.2 Using a GPT-like architecture

Given the context for image generation above, the first natural transition from Transplit to a more peak-efficient forecaster — i.e. no blurry predictions — is to use Transplit not to predict the consumption in the original space (in kWh), but to predict *consumption tokens*, which would result from a VQ-VAE trained on consumption time series.

Time series VQ-VAE

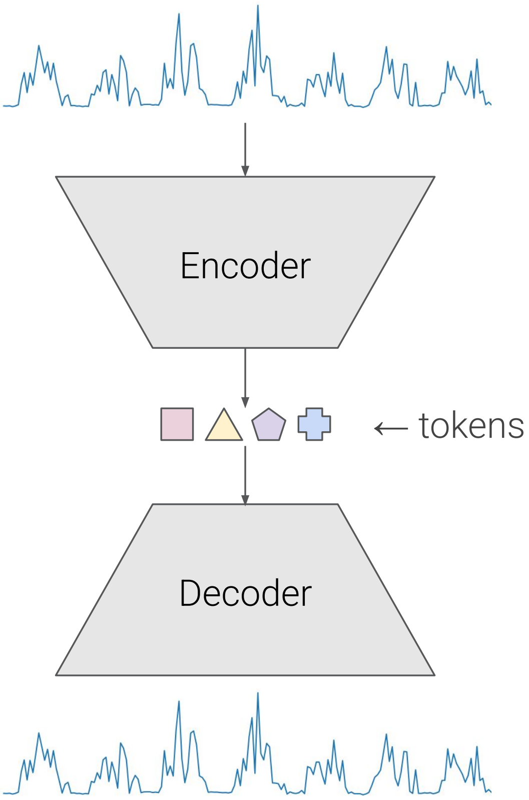


Figure 4.3: A VQ-VAE model converts time series into discrete tokens, enabling prediction in the token space rather than directly on time series values, potentially avoiding point-wise metric limitations.

We adapted here the VQ-VAE architecture for time series data. This implementation builds upon the official VQ-VAE codebase[1], with 2D convolutional filters replaced by 1D convolutions to accommodate the temporal nature of our data.

The architecture includes a vocabulary of 256 32-dimensional vectors with a compression ratio of 24:1 (i.e., one week of hourly consumption yields a sequence of 7 tokens). This configuration achieves reconstruction MSE below 10^{-6}, indicating near-perfect reconstruction. The high reconstruction quality can be attributed to the fact that the embedding dimension is not lower than the input dimensionality, resulting in minimal information loss. Visual inspection confirms that reconstructed consumption curves match the original data with high fidelity.

---

[1] From the authors [VV$^+$17b], adapted to PyTorch: `https://github.com/MishaLaskin/vqvae`

This tokenization approach successfully converts continuous consumption time series into discrete representations. We are now ready for the model approach.

MinGPT

Autoregressive language models, particularly GPT-style architectures, have demonstrated impressive capabilities in sequential token prediction tasks [RWC$^+$19; BMR$^+$20; RPG$^+$21]. Given their success in modeling discrete token sequences, these architectures present a more natural choice than the vanilla encoder-decoder Transformer architecture used in Transplit for predicting consumption tokens.

We implemented this approach using the open-source MinGPT framework[2], which provides a clean, minimal implementation of the GPT architecture. The overall pipeline, illustrated in Figure 4.4, combines VQ-VAE tokenization with autoregressive token prediction.

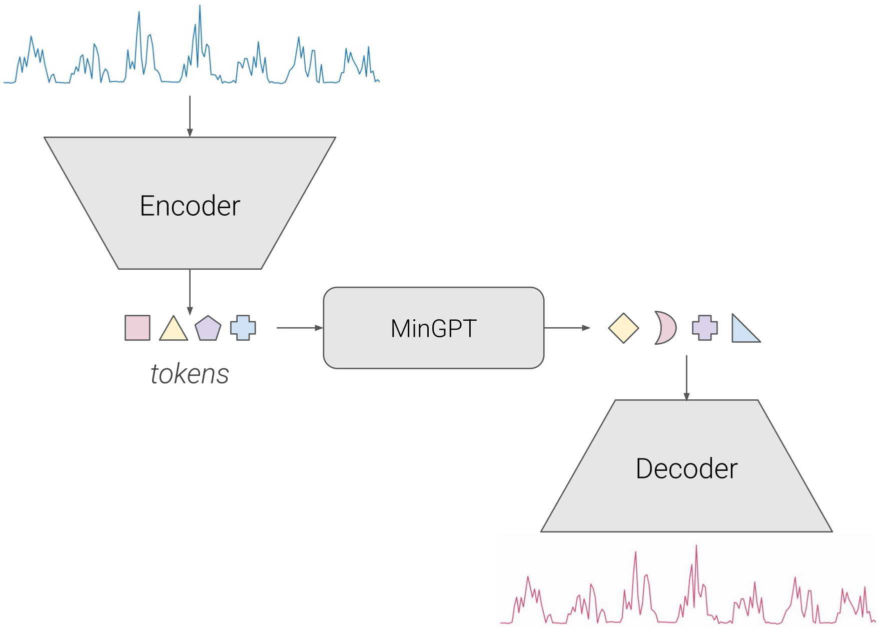


Figure 4.4: MinGPT architecture for time series forecasting: a GPT-like model trained to predict subsequent tokens in tokenized consumption sequences.

Despite the theoretical promise of this approach, empirical results were disappointing. The MSE increased dramatically to 0.313 for 30-day forecasts and 0.274 for 7-day horizons, compared to Transplit's performance of 0.191 and 0.169 respectively. The tokenized approach failed to preserve essential consumption patterns, resulting in significant degradation of overall forecasting accuracy.

---

[2] Andrej Karpathy repository: `https://github.com/karpathy/minGPT`

DDPM

Motivated by the exceptional generative capabilities of DDPMs in computer vision [HJA20; SSK^{+}20], we explored their application to conditional time series generation. Following the paradigm of text-conditioned image generation [RBL^{+}22a; SCS^{+}22], we conditioned DDPMs on historical consumption data to generate future consumption patterns.

The conditioning mechanism allows the diffusion process to incorporate temporal dependencies from past observations while generating diverse, high-quality future trajectories. Classifier-free guidance is used in this context [HS22a]. Figure 4.5 demonstrates the model's ability to generate consumption patterns from the same initial noise, with different guidance factors controlling the trade-off between diversity and conditioning strength.

While the generated patterns exhibit visually appealing characteristics with well-defined peaks, quantitative evaluation revealed limited improvement over the GPT-based approach. The DDPM achieved MSE values of 0.289 for 30-day and 0.269 for 7-day forecasts — marginally better than MinGPT but still substantially inferior to Transplit's performance. The inference time moreover was unexpectedly high: 30 seconds to process a batch of 32 samples, which corresponds to roughly 10 days to forecast nearly one million samples.

These results highlighted a critical limitation: while these generative approaches could produce visually compelling forecasts with prominent peaks, they came at the cost of overall forecasting accuracy. Our objective was not to sacrifice general forecasting quality for peak enhancement, but rather to refine the subdued peaks observed in Transplit's predictions (Figure 4.1) while maintaining competitive MSE performance.

This realization led us to reconsider our approach and ultimately motivated the development of our refinement-based methodology, which we describe in the following sections.

4.3 Background and Related Work

4.3.1 Time series and peak forecasting

A subdomain of time series forecasting, peak forecasting, aims to accurately predict the maximum value of a given variable and its timing in a given interval. In the energy sector, for instance, an accurate prediction of peak load demand is vital for grid operators, energy retailers, as well as customers [DMD^{+}21]. Interestingly, most of the research on electricity demand has been split into either predicting the timing of a peak [LB19] or its intensity [DMD^{+}21], but rarely both simultaneously [XXW12], a shortcoming reflected in the field's metrics: Peak Absolute Percentage Error (PAPE), a derivate of MAPE, for intensity and Hit Rate (HR)

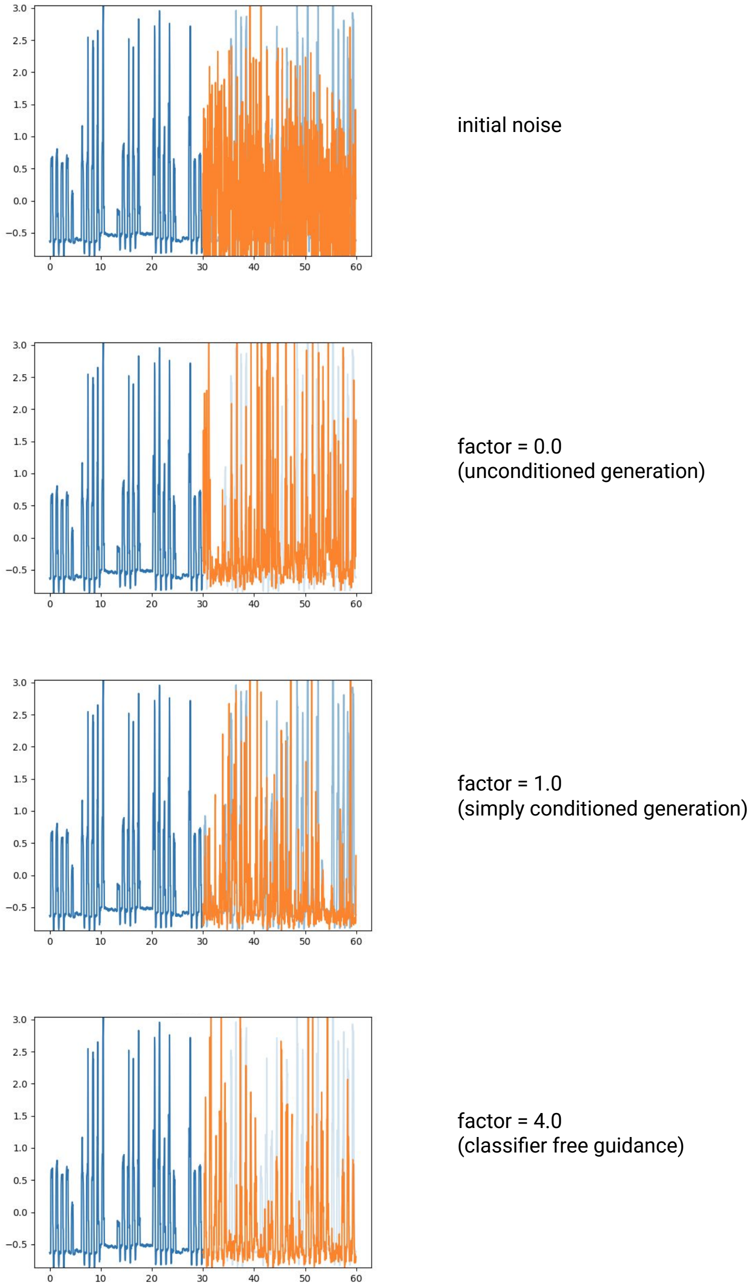


Figure 4.5: Example of generation with a DDPM with difference guidance factors, from the same initial noise.

for timing [XXW12]. Ideally, peaks should be directly contained within the time series forecasts instead of requiring dedicated models.

Regarding research from the machine learning perspective, several works have attempted to address the limitations of traditional metrics, [DZP$^+$19] proposed a recurrent model specifically designed to recognize the advent of extreme events and adjust their values based on historically similar events in the input time series. The model is optimized with a loss derived from the field of extreme value theory, called *Extreme Value Loss* (EVL). Another notable contribution, particularly in handling both the timing and magnitude of events, is the DILATE loss [LT19]. This loss function is distinctive as it simultaneously optimizes models for temporal dynamics and amplitudes, using an approach to penalize discrepancies in both timing and intensity of predicted vs. actual events. DILATE aims to bridge the gap in forecasting models by focusing on the alignment of time series data, but, however, doesn't focus on peak accuracy.

While both approaches show promises for peak forecasting, they have their limitations. EVL necessitates a custom-made recurrent-based model and cannot benefit from advances in time series forecasting. In contrast, DILATE is computationally intensive, its applicability is limited to a small number of data points, and models trained with this loss generally underperform in peak forecasting compared to models trained on MSE (see Table 4.3).

Therefore, we propose a pluggable approach to enhance the peak forecasting capabilities of state-of-the-art models trained on MSE, thus benefiting from the latest advances in the field while addressing their subpar peak performance. Diffusion models emerge as a natural candidate for this enhancement step.

4.3.2 Denoising Diffusion Probabilistic Models

Another type of model for time series forecasting are Denoising diffusion probabilistic models (DDPM). DDPMs are powerful generative models that learn to generate data by inverting an information-destruction process [SWM$^+$15]. Recently, they have been successfully used to generate realistic images [DN21b; RBL$^+$22a], and form the basis for recently developed, state-of-the-art probabilistic models [RSS$^+$21; LLW$^+$22]. As they are trained to reproduce a distribution, they do not suffer from the above-mentioned drawbacks of point-wise losses and could potentially constitute a solution for the peak forecasting issue.

They, nonetheless, exhibit limited performances in time series forecasting, which rarely reaches the one of non-probabilistic models [LLW$^+$22; SAM$^+$22]. Their long inference time and their need for significant resources also constitute major drawbacks for real-world applications. With this in mind, we aim to introduce a cheaper and deterministic approach.

4.3.3 Inverse Heat Dissipation

Inverse heat dissipation models (IHDMs) use, similarly to DDPMs, the inverse process of an information-eroding mechanism to generate new data points. Instead of adding Gaussian noise, they, as the name suggests, are based on the physical heat dissipation process. Initially proposed for image data, they can be adapted to time series as well. Figure 4.6 illustrates the effect on time series of IHDMs and DDPMs. Intuitively, the dissipation process smooths input data, until all finer details in the data are erased.

In the following sections, we first lay out the forward process of IHDMs before introducing the backward process, which, by *deblurring* or *de-smoothing* input time series, ultimately generates new data points.

Forward process. The forward process is governed by the following linear partial differential equation:

$$\frac{\partial}{\partial t}\mathbf{u}(x, y, t) = \Delta\mathbf{u}(x, y, t) \tag{4.1}$$

where $\mathbf{u}(x, y, t) \in \mathbb{R}$ is the heat profile of a physical body in two dimensions, $t \in \mathbb{R}$ is time, $(x, y) \in \mathbb{R}^2$ is the position vector, and $\Delta = \nabla^2$ is the Laplace operator. Instead of performing multiple small incremental diffusion steps, the equation can be solved analytically and the forward process can be performed in one step. Using the fact that the Laplacian operator Δ has a finite Eigendecomposition in frequency space[3] $\Delta = \mathbf{V\Lambda V}^T$ where $\mathbf{V}^T$ is the cosine projection matrix and $\mathbf{\Lambda}$ is a diagonal matrix containing the negative squared Eigenfrequencies, one can perform a Discrete Cosine Transformation (DCT) in accordance with boundary conditions and rewrite the forward process as [RHS22]:

$$\begin{aligned} \mathbf{u}(x, y, t) &= F(t)\mathbf{u}(x, y, t = 0) \\ \iff \tilde{\mathbf{u}}(t) &= \exp(-\mathbf{\Lambda} t)\tilde{\mathbf{u}}(t = 0) \end{aligned} \tag{4.2}$$

where $F(t) = \exp((t - t_0)\Delta) \in \mathbb{R}$ is an evolution operator [DZ14] and $\tilde{\mathbf{u}}(t) = \mathbf{V}^T\mathbf{u}(t)$ is the projection of $\mathbf{u}(t)$ in the DCT space. Note that the dependency of $\tilde{\mathbf{u}}$ on the DCT space variables has been omitted for readability. At the end of the diffusion process, we add Gaussian noise with variance σ^2 to the blurred data point $\mathbf{u}_t$ such that the forward process can be expressed as a probabilistic model and that the reverse process can branch out into different paths. Note that in contrast to DDPMs, the forward process is entirely deterministic and only becomes probabilistic with this step. Finally, this leaves us with the following statistical model:

$$q(\mathbf{u}_t|\mathbf{u}_0) = \mathcal{N}(\mathbf{u}(t); \mathbf{V}\exp(-\mathbf{\Lambda}\tau_t)\mathbf{V^T}\mathbf{u}(t = 0), \sigma^2\mathbb{1}) \tag{4.3}$$

---

[3] A *finite* Eigendecomposition is only possible as the input signal is discrete. That allows for the definition of a Nyquist limit as a natural cut-off frequency

where we omitted the dependency of $\mathbf{u}$ on x and y and τ_t is an arbitrary dissipation schedule, which increases monotonically. The exact set of values $\{\tau_t \in [0,1]\}_{t=1}^{T}$ is referred to as the scheduler, and the right choice of this set influences the generative performance of diffusion models [Che23].

Backward process. The generative process is defined by approximately inverting the blurring process with a learned deblurring process, and applying the following equation T times:

$$p(\mathbf{u}_{t-1}|\mathbf{u}_t) = \mathcal{N}(\mathbf{u}_{t-1}; f_\theta(\mathbf{u}_t), \delta^2 \mathbb{1}) \tag{4.4}$$

where f_θ refers to the U-Net and δ^2 is the constant sampling variance. The denoising process generates a new data point $\hat{\mathbf{u}}_0$ from the latent noise variable $\mathbf{u}_T$ by calling the U-Net T times, every call performing a small deblurring step. IHDMs differ from DDPMs in two crucial ways. First, the U-Net f_θ directly predicts the latent representation $\mathbf{u}_t$ of the initial data point $\mathbf{u}_0$ instead of the noise added in the forward process. Secondly, the IDHMs affect the different frequencies of the original signal differently. In the forward process, higher frequencies decay faster compared to lower frequencies, as evidenced by the fact that $\Lambda_{n,m}$ are proportional to the negative squared frequencies in (4.3). Reversely, higher frequency parts are generated in the later steps of the generative process. This contrasts with DDPMs, which add white noise to the data, thereby affecting every frequency of the signal equally until all frequencies are drowned out. A behavior we exploit in our approach [RHS22].

Sampling Noise. Empirically, we find that the variance of the sampling noise δ has only a small impact on the predictive performance of the IHDM (cf. Section 4.5.3). In all our experiments, we set $\delta = 0$, thus making the generative model *deterministic*. This greatly increases the speed of the framework, as only a single "sample" is generated. This ultimately makes DeDiPeak more lightweight than other proposed approaches based on probabilistic models [RSS^{+}21; LLW^{+}22]. A detailed comparison is included in the Appendix.

Optimization. During training, the U-Net is tasked to predict the data point $\mathbf{u}_{t-1}$ given the $\mathbf{u}_t$, while $\mathbf{u}_t$ is calculated with equation 4.2. Empirically, one uses the following loss function to evaluate the squared difference between the prediction $f_\theta(\mathbf{u}_t, t)$ and the expected value of the blurred data point from the time step $t-1$ given by $\bar{\mathbf{u}}_{t-1} = \mathbf{V}\exp(\mathbf{\Lambda}\tau_{t-1})\mathbf{V}^T\mathbf{u}_0$.

$$\mathcal{L} = \mathbb{E}_{t\sim\mathcal{U}(0,\ldots,T)}\mathbb{E}_{\mathbf{u}_t\sim q(\mathbf{u}_t|\mathbf{u}_0)}\left[||f_\theta(\mathbf{u}_t, t) - \bar{\mathbf{u}}_{t-1}||^2\right] \tag{4.5}$$

This loss function constitutes a single Monte Carlo inference step from the inference distribution $q(\mathbf{u}_t|\mathbf{u}_0)$ performed on a lower bound on the model likelihood $\log p(\mathbf{u_0})$ [RHS22].

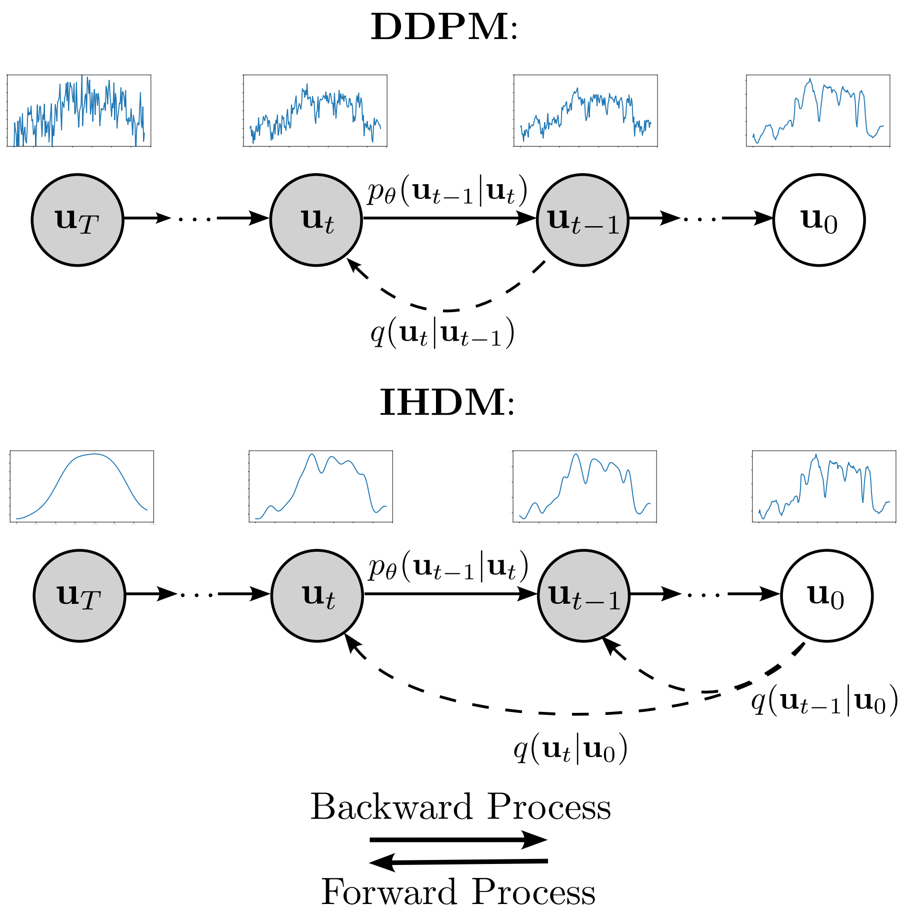

DDPM:
$\mathbf{u}_T$
$\mathbf{u}_t$
$p_\theta(\mathbf{u}_{t-1}|\mathbf{u}_t)$
$\mathbf{u}_{t-1}$
$\mathbf{u}_0$
$q(\mathbf{u}_t|\mathbf{u}_{t-1})$
IHDM:
$\mathbf{u}_T$
$\mathbf{u}_t$
$p_\theta(\mathbf{u}_{t-1}|\mathbf{u}_t)$
$\mathbf{u}_{t-1}$
$\mathbf{u}_0$
$q(\mathbf{u}_{t-1}|\mathbf{u}_0)$
$q(\mathbf{u}_t|\mathbf{u}_0)$
Backward Process
Forward Process

Figure 4.6: Illustrations of DDPM (denoising) and IHDM (deblurring, or *desmoothing*) and their effects on time series.

4.3.4 Parallel with DDPMs

DDPMs consist of a forward process, which adds an incrementally increasing amount of noise to data points, and a backward process which generates data points from noisy latent representations. DDPMs and IHDMs both draw on the concept of inverting an information-eroding process. The primary distinction between them lies in the underlying mechanisms: DDPMs rely on the addition of Gaussian noise, while IHDMs are based on the process of heat dissipation. Key differences between the two are discussed in section 2.3 of the chapter and in 4.6.1.

Forward process. The diffusion, or forward process, is defined by gradually adding Gaussian noise to an original data point $\mathbf{u}_0$ over multiple steps. The data distribution at time step t, given the preceding data point $\mathbf{u}_{t-1}$, is governed by the following Markov transition kernel:

$$q\left(\mathbf{u}_t|\mathbf{u}_{t-1}\right) = \mathcal{N}(\mathbf{u}_t; \bar{\alpha}_t\mathbf{u}_{t-1}, \bar{\sigma}_t\mathbb{1}). \tag{4.6}$$

Utilizing the reparametrization trick ($\mathbf{u}_t = \bar{\alpha}_t\mathbf{u}_{t-1} + \sqrt{\bar{\sigma}_t}\epsilon_t$, with $\epsilon_t \sim \mathcal{N}(0, \mathbb{1})$) as well as the fact that one can sample $\mathbf{u}_t$ at arbitrary time steps in a closed form, the forward process can be written as

$$q\left(\mathbf{u}_t|\mathbf{u}_0\right) = \mathcal{N}(\mathbf{u}_t; \alpha_t\mathbf{u}_0, \sigma_t\mathbb{1}) \tag{4.7}$$

Here $\alpha_t = \prod_{i=0}^{t}\bar{\alpha}_i$ and $\sigma_t = \sum_{i=0}^{t}\bar{\sigma}_{t-i}\prod_{j=1}^{i}\bar{\alpha}_{t-j+1}^2$. Similarly to IHDMs, DDPMs incorporate a scheduler, namely $\{(\alpha_t, \sigma_t) \in [0,1]^2\}_{t=1}^{T}$, which can be arbitrarily chosen, under the constraint that the noise amplitude σ_t increases monotonically and the factor α_t monotonically decreases. With every forward step, the information contained in $\mathbf{u}_0$ is gradually lost such that in the limit $t \to \infty$ the data distribution $q(\mathbf{u}_t|\mathbf{u}_0) \to \mathcal{N}(\mathbf{u}_t; 0, \mathbb{1})$.

Backward process. Generated data points are sampled from the joint distribution $p_\theta(\mathbf{u}_{0:T})$ of the learned denoising process which is an approximation of the true denoising distribution $q(\mathbf{u}_0|\mathbf{u}_T)$ i.e. the inverse process of the diffusion process:

$$p_\theta(\mathbf{u}_{0:T}) = p(\mathbf{u}_T)\prod_{t=1}^{T}p_\theta(\mathbf{u}_{t-1}|\mathbf{u}_t); \qquad p_\theta(\mathbf{u}_{t-1}|\mathbf{u}_t) = \mathcal{N}(\mathbf{u}_{t-1}; \mu_\theta(\mathbf{u}_t, t), \Sigma_\theta(\mathbf{u}_t, t)) \tag{4.8}$$

Here μ_θ and Σ_θ are the outputs of a trainable U-Net f_θ, tasked to predict the latent variable $\mathbf{u}_{t-1}$ from $\mathbf{u}_t$. By invoking this U-Net T times—each call executing a small denoising step as per Equation 4.8—a new data point $\hat{\mathbf{u}}_0$ is ultimately generated. The mathematical justification for this approach can be found in [SSK+20]. Essentially, if the denoising steps are sufficiently small and $\mathbf{u}_0$ is expressive enough, the model can accurately capture the target distribution. Empirically, the quality of the generated samples can be improved by modeling the denoising process as a function of the noisy latent variable $\mathbf{u}_t$ and time t, rather than directly predicting the data point $\mathbf{u}_0$, i.e. the U-Net predicts the noise ϵ_t added during one diffusion step: $\epsilon_t = f_\theta(\mathbf{u}_t, t)$. One step of the generative process is finally given by

$$\mathbf{u}_{t-1} = \frac{1}{\alpha_t}\mathbf{u}_t - \frac{\sigma_t}{\alpha_t}\epsilon_t. \tag{4.9}$$

Optimization Throughout the training phase, the U-Net is tasked to predict the noise ϵ_t that would be added to an already noisy data point u_t at time step t. Empirically, one uses the following loss function

$$\mathcal{L} = \mathbb{E}_{t\sim\mathcal{U}(0,1)}\mathbb{E}_{\epsilon_t\sim\mathbb{N}(0,\mathbb{1})}\left[||f_\theta(\mathbf{u}_t, t) - \epsilon_t||^2\right]. \tag{4.10}$$

This loss function is derived from the variational lower bound of the model likelihood $-\mathbb{E}_{\mathbf{u}_0 \sim q(\mathbf{u}_0)} \log p(\mathbf{u}_0)$, and has been found to be more stable during training [KSP+21; HJA20].

4.3.5 Noise Scheduler

Realizing that IHDMs can be considered as a DDPM in DCT space, [HS22b] extracted the scheduler of the heat diffusion equation and improved it. Specifically, the scheduler $\mathbf{d}_t = \exp(-\lambda \tau_t)$ in IHDMs, where λ contains the diagonal matrix elements of $\mathbf{\Lambda}$, suppresses high frequencies too strongly, thereby inflating small errors over many steps of the deblurring process.

This issue is solved by modifying the scheduler as follows:

$$\mathbf{d}_t = (1 - d_{min}) \exp(-\lambda \tau_t) + d_{min} \tag{4.11}$$

with $d_{min} = 0.001$, avoiding that higher frequencies are dampened too strongly. In all of our experiments will use the IHDM architecture with the improved scheduler.

4.3.6 Adaptive loss

The adaptive loss is a training objective introduced in 2019 by [Bar19]. Its application to times series has been studied [SAM+22] and shows slight improvements when used instead of MSE during training. Let $\xi = \hat{y}_i - y_i$, with $\hat{y}_i$ as a predicted data point and y_i the ground truth, the loss is defined as follows:

$$f(\xi, \alpha, c) = \frac{|\alpha - 2|}{\alpha} \left(\left(\frac{(\xi/c)^2}{|\alpha - 2|} + 1 \right)^{\alpha/2} - 1 \right) \tag{4.12}$$

where $\alpha \in \mathbb{R}$ and $c \in \mathbb{R}_+^*$ are trainable parameters that control the robustness of the loss function and the shape of the loss's quadratic bowl near $x = 0$, respectively.

4.4 DeDiPeak

DeDiPeak is a novel framework that (1) introduces two performance indicators for peak predictions in time series and (2) a deblurring diffusion model built upon forecasting models to improve peak prediction in time series.

4.4.1 Peak Forecasting Metrics

To address the issues with classical metrics, such as MSE and MAE in peak forecasting, we propose two new metrics which do not rely on an element-wise comparison between the ground truth value $y(t)$ and the predicted one $\hat{y}(t)$. Both metrics are parameterized and reflect different operational properties that the forecasted peaks should follow.

The first one, $P3_{\mathcal{E}}$, called Peak Prediction Performance Euclidean, is partially defined by:

$$E_{\mathcal{E}}(y, \hat{y}) = \frac{1}{|\Pi(y)|} \sum_{t \in \Pi(y)} \min_{t' \in \Pi(\hat{y})} (\alpha |t - t'|^2 + \beta |y(t) - \hat{y}(t')|^2) \tag{4.13}$$

where $\Pi(y)$ is a function mapping time series to a set containing the time stamps of all the peaks contained within $y(t)$. $E_{\mathcal{E}}(y, \hat{y})$ is specifically designed to evaluate both the time and height of the peaks. Each of those contributions is weighted, allowing the metric to be adapted. $E_{\mathcal{E}}(y, \hat{y})$ faces one important issue: a constant prediction equal to the level of the highest peak in the time series is deemed as a good prediction. Therefore, we define $P3_{\mathcal{E}} = E_{\mathcal{E}}(y, \hat{y}) + E_{\mathcal{E}}(\hat{y}, y)$. In this form, the metric has two terms mirrored under the transformation $y \to \hat{y}$ and $t \to t'$ which penalizes said trivial predictions.

The second metric we propose, the Peak Prediction Performance sliding window metric, is defined as $P3_{sw} = E_{sw}(y, \hat{y}) + E_{sw}(\hat{y}, y)$, with:

$$E_{sw}(y, \hat{y}) = \frac{1}{|\Pi(y)|} \sum_{t \in \Pi(y)} \Big(y(t) - \max_{t' \in [t-T, t+T]} \hat{y}(t') \Big)^2 \tag{4.14}$$

Again, we ensure that $P3_{sw}$ is symmetric under the transformation $y \to \hat{y}$ and $t \to t'$. For every peak in the ground truth, $E_{sw}(y, \hat{y})$ identifies the local maximum in the prediction located in a window of size $2T + 1$ centered around t. Ideally, this local maximum is a peak. The squared difference in amplitude between the original peak and the local maximum is then added to the metric. Intuitively, the metric looks for peaks in the prediction in the vicinity of the original peak and then compares those instead of performing element-wise comparisons. The parameter T controls the size of the sliding window and by extension the sensitivity of the metric to temporal delays in peak predictions compared to the ground truth. The term $E_{sw}(\hat{y}, y)$ works analogously, with the prediction and ground truth exchanged.

Although they both enable peak performance comparison, $P3_{\mathcal{E}}$ and $P3_{sw}$ differ in their parametrization and intended usage as they reflect different properties of a peak. $P3_{\mathcal{E}}$ aims at estimating the general peak performance while balancing the peak timing and intensity more precisely. On the other hand, $P3_{sw}$ constitute an out-of-the-box metric for the typical peak forecasting use case where the right peak value needs to be within a given time window margin. Their parameters offer a flexible usage that can be adapted to specific needs and has allowed an application with our industrial partner.

Defining peaks. An essential part of the suggested metrics is the set of timestamps to be considered as peaks $\Pi(y)$, which depends on the definition of peaks in the first place. While multiple definitions are possible, we empirically define a peak as a data point having a larger amplitude than its n neighbors to the left and the right, i.e.:

$$y_t \in \Pi(y) \iff y_t > y_{t+k}, \forall k \in -n, n \setminus \{0\} \tag{4.15}$$

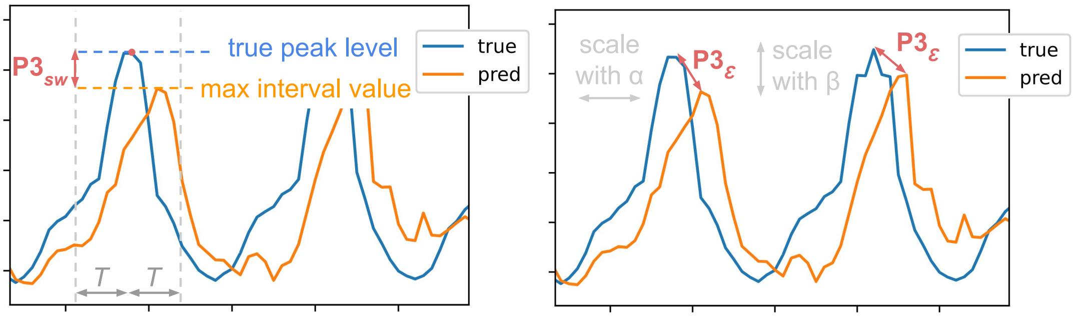


Figure 4.7: Illustration of $P3_{sw}$ and $P3_{\mathcal{E}}$ metrics, respectively.

A visual example led by this definition is illustrated on Figure 4.8.

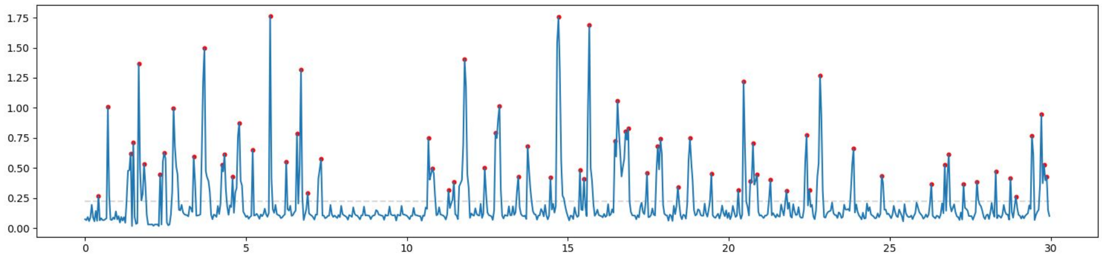


Figure 4.8: Visualization of peaks within a consumption time series. The dashed grey line corresponds to the average amplitude.

4.4.2 Effectively Forecasting Peaks

Shortcomings of current models. As mentioned in the introduction, transformer-based models are good at trend prediction but often fail to capture the non-smooth nature of peaks and their correct amplitude. While DDPMs presented in the background section can incorporate sensible peaks in their predictions, they appear to be quite sensitive to the input noise and thus produce peaks at irregular intervals [RSS+21; LLW+22]. For now, their performance remains inferior compared to transformer-based models for general forecasting tasks [AS22]. Averaging over many predictions generated with different noise inputs solves that issue, but comes at the price of averaging out the peaks, which makes them on their own ill-suited for peak prediction tasks. Additionally, diffusion-based models suffer from the fact that the underlying U-Net has to be called multiple times during inference, resulting in high computational costs.

Model synergy. We propose a new approach that uses IHDMs on top of an existing base model, *e.g.* transformer-based models, thereby navigating the shortcomings of the expensive IHDMs and the risk-averse base models. DeDiPeak

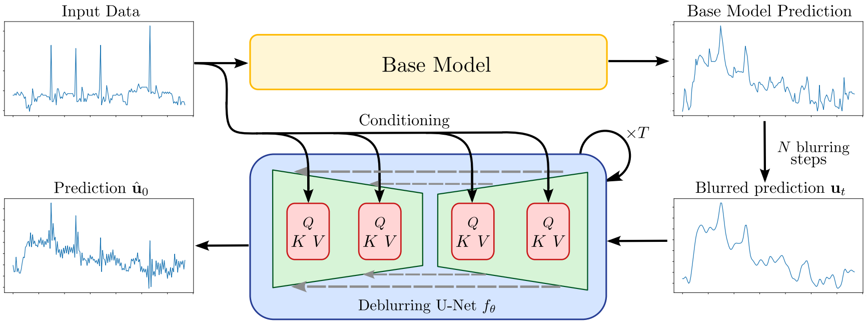

Input Data
Base Model
Base Model Prediction
Conditioning
×T
N blurring steps
Prediction $\hat{\mathbf{u}}_0$
Q
K V
Q
K V
Q
K V
Q
K V
Blurred prediction $\mathbf{u}_t$
Deblurring U-Net f_θ

Figure 4.9: Figure illustrating our new framework DeDiPeak.

improves peak forecasting by enhancing existing peaks and occasionally generating new ones on top of an initial prediction of a base model. This is achieved by performing the final steps of a dedicated IHDM, offering a lighter solution than using a fully-fletched IHDM. Our approach explicitly leverages the fact that during the blurring process, high-frequency parts of the initial signal are suppressed the strongest and, thus, are the last to be generated in the backward process. An illustration is shown in Figure 4.9, showing an example of 96 data points being passed to a base model producing a preliminary prediction. Empirically, we found that performing $N = 7$ blurring steps according to equation 4.2 on the initial prediction increases the overall performance of the framework. Finally, calling the trained deblurring U-Net f_θ $T = 15$ times produces a prediction containing clear peaks.

Adjusting IHDMs for peak prediction. IHDMs have, to the best of our knowledge, not been applied to times series and so we have made two changes to their overall structure. First, we implement causal attention in the U-Net of the IHDMs to prevent any leakage of information from the future and thereby preserve temporal causality. Concretely, this means setting the values of future time steps to zero during the attention computation, ensuring that the model only uses information from the current and past time steps [NBS19].

Secondly, the formulas of the IHDM have been adapted to time series, instead of images, resulting in a change in input variables: $\mathbf{u}(x, y, t) \to \mathbf{u}(t, k)$. The input data depends on the real-time of the measurements t and the time step of the blurring process k. This changes the equation of the diffusion process (4.2) to $\partial_k \mathbf{u}(t, k) = \partial_t^2 \mathbf{u}(t, k)$. Ultimately, this has some implications for the implementation of the IHDM, which are further highlighted in this chapter.

4.5 Experiments

In this section, we empirically assess DeDiPeak, by investigating the following points: **(1)** Do our $P3$ indicators accurately capture peak forecasting performance? **(2)** Is the addition of a deblurring diffusion model improving peak forecasting? and **(3)** How does Dedipeak compare to other approaches?

4.5.1 Empirical Setup

Datasets. We conduct our experiment on a classical benchmark of various datasets, commonly used for time series forecasting evaluation [ZZP$^+$21; KKL20; whXW$^+$21; CLD$^+$20; ZMW$^+$22b; SAM$^+$22; COO$^+$22] composed of a weather [wet20], traffic [tra20], and 3 electricity and energy-based datasets [Dat14; ZZP$^+$21] (electricity, ETT). We set ratios of training / validation / testing sets to respectively 70 / 10 / 20%, along the time axis.

Metrics. Regarding the peak definition equation (4.15), we set $n = 5$ neighbors. We evaluate the result according to 6 metrics. The first two, MSE and MAE, are traditionally used for time series forecasting. We also add Peak MSE (PMSE) and Peak MAE (PMAE), their peak value equivalent. While these metrics report accuracy for well-timed peaks, they suffer from the same drawbacks as MSE and MAE regarding off-timed forecasted peaks. This motivated the development of our two metrics: $P3_{sw}$ for which we set T to the number of neighbors, i.e. $T = n = 5$, and $P3_{\mathcal{E}}$ for which we set $\alpha = 1/n^2 = 0.04$ and $\beta = 1.0$. The rationale behind these choices along with additional experiments can be found in section 4.6.

4.5.2 Implementation details

Forecasting models. In our study, we rely on a set of recent transformer models as base models: FiLM [ZMW$^+$22a], Autoformer [whXW$^+$21], NHits [COO$^+$22], FEDformer [ZMW$^+$22b], Performer [CLD$^+$20], Informer [ZZP$^+$21], Reformer [KKL20]. As the models are designed for long-term forecasting, we provide a 96-long sequence as input to the transformer models in order to forecast 720 values. We perform multichannel forecasting, i.e., all the datasets' channels are given as input to the model. The input data is first converted by an embedding layer, combining its channel values with timestamps embedding. The transformers are built with 2 encoder and 1 decoder blocks, and their inner representation size d_{model} is set to 256. The model is trained by using the Adam optimizer [KB15] with the adaptive loss for 10 epochs with early stopping (patience 3), an initial learning rate of 10^{-4}, and a batch size of 16.

We then carry out a comparative study with models trained with the EVL [DZP$^+$19] and DILATE loss [LT19], taking the model architecture and hyperparameters proposed by their respective authors:

- The DILATE loss is used on a Seq2Seq model [SVL14], takes 168 values as input, and has a forecast length of 24.
- A custom GRU-based model is optimized with the EVL. We will refer to this approach as *EVT* (Extreme Value Theory). The EVT model is fed with an input length of 336, and the forecast length is set to 168.

Inverse Heat Dissipation. The U-Net is trained in a single-channel fashion such that one IHDM is trained per dataset, every channel being taken independently. The U-Net is composed of 4 levels (factors of 1, 2, 2, 2 respectively), such that the sequence length is divided by 8 at the deepest latent level. We use a batch size of 32 and train the model using the Adam optimizer with the adaptive loss. We set 50 total diffusion steps for training. At inference time, we use classifier-free guidance [HS22a] with a factor of 5.0. The conditioning input is the same as for the base models, including embedding. Additionally, we blur the base models' outputs by 7 steps and deblur them using the 15 last backward diffusion steps[4]. More details about the number of steps impact are available in section 4.6. We set $\tau_0 = 0.5$ moving exponentially to $\tau_{50} = 16$.

Experiments are conducted on an Nvidia Quadro RTX 4000 8GB GPU and run five times to limit the effect of randomness.

4.5.3 Results

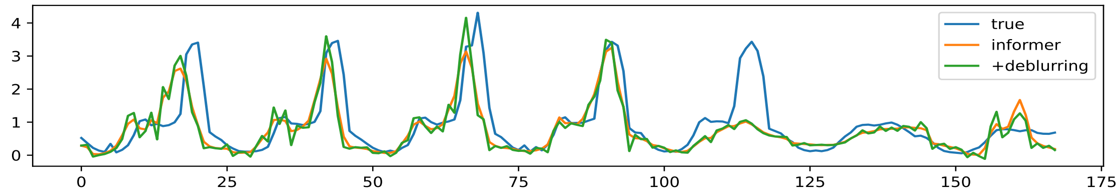
P3 as peak forecasting metrics. Figure 4.10 showcases a forecast made by the informer model on the traffic dataset along with the improved peak forecast using our deblurring model. While the enhancement is evident in the visual representation, we seek a quantitative measure of this improvement. The attached table presents the values of our 6 selected metrics on this forecast. We observe that all traditional metrics, *i.e.* MSE, MAE, PMAE and PMSE, see an increase in error after the application of the deblurring model. In contrast, our metrics indicate a decrease in error, revealing that our metrics are correctly capturing the improvement in terms of peak prediction. Notice that in this case, most peak forecasts exhibit slight temporal misalignment. While this slight deviation is tolerable for peak forecasting, it does influence the average error. More examples are provided in section 4.6.

Building on this validation, we further analyze the peak forecasting efficacy of our base models as detailed in Table 5.1. Overall, results reveal that the MSE and $P3$ metrics are strongly correlated, indicating that a good peak forecasting performance is linked to a lower MSE. However, this correlation breaks down when the MSE drops below a certain threshold. In this regime, the decorrelation between the MSE and $P3$ metrics, metrics that exclusively evaluate the amplitude and

---

[4] Note that since only 15 steps are used in practice, training on these 15 steps only without changing the τ values instead of 50 would be a better option, but has not been done here to allow for a comparison with more blurring steps without having to re-train the model.

timing of peaks, indicates that the MSE fails to properly capture peak predictive performance. It follows that the $P3P$ metrics are able to capture valuable additional information on the peak prediction performance of a model, which can ultimately guide the model selection and optimization process.



	MSE	MAE	PMSE	PMAE	$P3_{sw}$	$P3_{\mathcal{E}}$
Informer	**0.47**	**0.40**	**4.05**	**1.69**	1.36	1.91
Deblurring	0.54	0.45	4.54	1.75	**0.97**	**1.51**

Figure 4.10: Example of improved peaks by the diffusion model taken from the *traffic* dataset.

Deblurring for peak enhancement. As mentioned above, Figure 4.10 illustrates the improvement in peak forecasting that DeDiPeak can offer. By itself, the informer tends to predict shy peaks, an issue our deblurring model was specifically developed to address. Note however that while the deblurring model increases the accuracy of peak forecasts, it may also occasionally introduce noise to the non-peak parts of predictions as a result of inducing high-frequency perturbations into the blurred transformer prediction. This can degrade the prediction outside peak areas, which partly explains the slight rise in MSE observed upon the application of our deblurring model. Interestingly, at the end of the forecast window, we notice that the deblurring can also lower a peak when necessary.

Table 5.1 lists the MSE and $P3$ of the predictions made with a subset of our base models on each dataset, before and after the use of the deblurring model. We consistently observe improvements in $P3_{sw}$ in 24 out of 25 cases, as for $P3_{\mathcal{E}}$. Examining the top three baseline models by MSE, we find that predictions from FiLM benefit the most from applying the deblurring model. They achieve the highest $P3$, despite ranking lowest in $P3$ before the deblurring model is applied. Predictions made by FEDFormer are also improved by the deblurring model but to a lesser extent. On the other hand, despite an overall average improvement, the predictions made by NHits are sometimes degraded by the deblurring model as indicated by $P3_{\mathcal{E}}$. This indicates that a small subset of combinations of dataset and model are not well-suitable for the DeDiPeak framework, revealing the need for further investigation in that direction.

Noise Parameters. To optimize both the training noise variance σ^2 and the sampling noise variance δ^2 we ran additional experiments using different combinations of base model and dataset. Specifically, Fig. 4.11 illustrates the MSE and P3 metrics of DeDiPeak framework applied op top of the FiLM model on the training set of ETTm2. First, we observe the best predictive performance of the framework when the standard deviation of the training noise $\sigma = 0.01$, just as in the original IHDM paper [RHS22]. Secondly, we find that generally the model performance weakly depends on the sampling noise variance δ^2. Therefore, we chose to set $\delta = 0$, as it presents a reasonable trade-off between the performance of the IHDM and the computational load of the framework. Note that with $\delta = 0$, only a single "sample" time series needs to be generated during inference.

4.5.4 Comparison Studies

Extreme Value Loss. First, we compare our model to a related study based on a unique loss function called the *Extreme Value Loss* (EVL) and a dedicated architecture for extreme event prediction using gated recurrent units (GRU) [DZP$^+$19]. Similarly to DeDiPeak, the EVL approach aims to predict extreme events (peaks) in time series. For comparison, Table 4.2 illustrates the performance of the GRU-based architecture when trained with EVL versus when it is trained with MSE and subsequently enhanced by our deblurring model. The forecast length for both models is set at 168 values, consistent with the original study.

The results demonstrate the superior performance of DeDiPeak across different datasets. Using both methods at once i.e., applying DeDiPeak on top of predictions from EVL results in poor forecasts due to the excessive inflation of already existing peaks. DeDiPeak works best with base models that predict the general trend of time series but fail to produce accurate peaks. Although the EVL can be measured for the EVT model, it cannot be applied to the deblurring model, as the computation of this loss requires outputs from a submodule predicting the probability of an extreme event, specifically designed in the EVT model. Finally, these experiments highlight the versatility of DeDiPeak, showcasing its compatibility with recurrent neural networks as well

DILATE Loss. The DILATE loss is designed to address the weaknesses of point-wise loss functions in peak forecasting tasks [LT19]. Originally, it was introduced to train a Seq2Seq transformer. Table 4.3 compares the performance of the Seq2Seq model trained with the DILATE loss to the performance of a Seq2Seq model trained with the MSE and improved with DeDiPeak. To guarantee the same experimental conditions as in the original paper, we only forecast 24 values for both approaches. Note that the complexity of calculating the DILATE loss scales quadratically with the forecast length, making longer forecasts much more costly.

The results indicate that even on short length scales the DeDiPeak framework on the Seq2Seq model outperforms the Seq2Seq trained with the DILATE loss. Notably, our framework, with Seq2Seq as a base model, achieves better DILATE scores than a Seq2Seq trained on the DILATE loss itself.

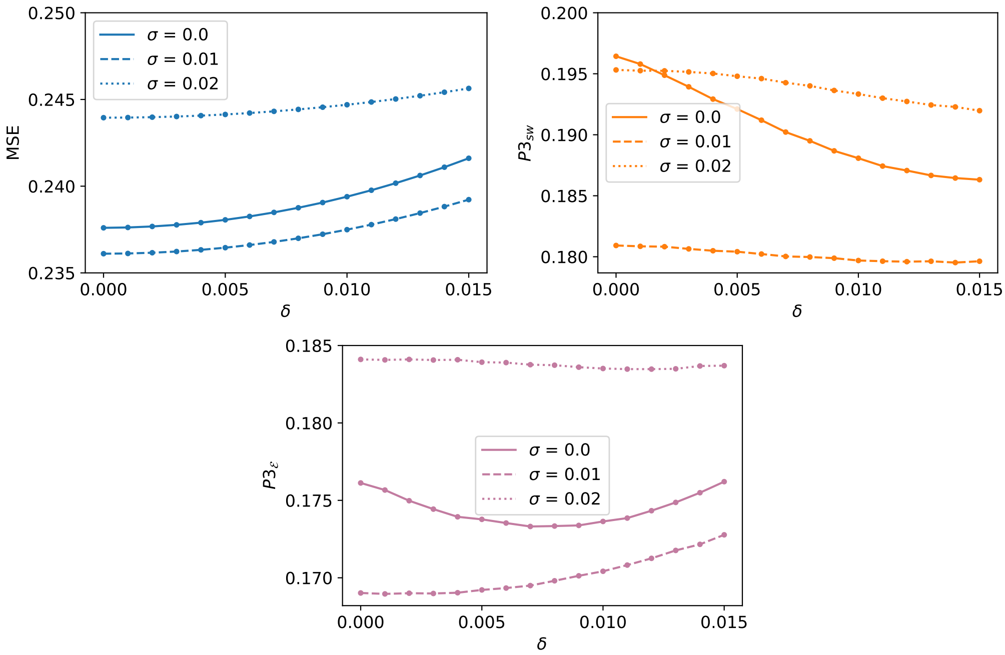


Figure 4.11: Dependence of the IHDM predictive performance (MSE in blue, $P3_{sw}$ in orange, $P3_{\mathcal{E}}$ in violet) applied on top of the FiLM base model for the ETTm2 dataset on the variance of the sampling noise δ^2 and of the training noise σ^2.

Stand-Alone Deblurring Inference. With our modifications to the original IHDM architecture, it is in theory possible to produce time series forecasts with a stand-alone deblurring model, that does not rely on the predictions of base models. However, its implementation currently demands significant resources, as even more calls to the U-Net are necessary. Still, as a comparison, we implemented a standalone deblurring model starting from a flat baseline (equivalent to the input's average value).

Overall, the results proved unsatisfactory due to poor peak timing prediction. An example of a forecast on the *traffic* dataset is illustrated in Fig. 4.12. The MSE of a pure deblurring model forecasting on the *traffic* dataset equals 2.109, which is a factor of 3 greater than our best-performing approach, illustrating its poor performance compared to other models used in this chapter. This indicates that relying on an initial forecast as a baseline is more efficient.

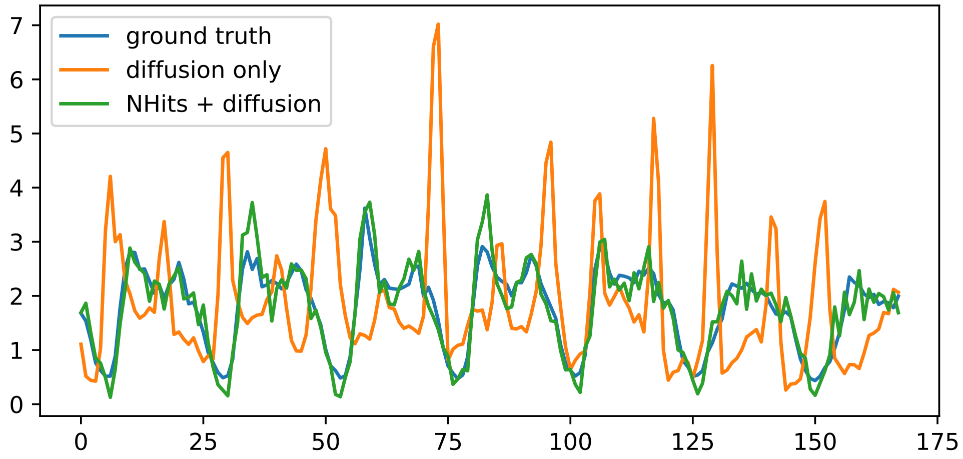


Figure 4.12: Comparison between a stand-alone deblurring model and DeDiPeak

4.6 Details and additional results

4.6.1 Implementation

Blurring, deblurring, and training

This section contains the pseudo-code of the essential parts of the IHDM used in DeDiPeak. We would like to highlight the fact that since the time series data is one-dimensional, DCT is only performed along one direction and the variable `freqs` is a one-dimensional tensor compared to a two-dimensional one used for images.

It is also important to note that, before training any model, the data is preprocessed with standardization scaling.

```
blur_taus = exp(linspace(log(tau_min), log(tau_max), num=total_steps))
freqs = linspace(0, 1, num=pred_len, endpoint=False) ** 2 / 2

def blur(x0: array, t: int):
    # x0.shape = (pred_len,)
    tau = blur_taus[t]
    dct_coefs = DCT(x0, norm='ortho')
    dct_coefs = dct_coefs * exp(- tau * freqs)
    xt = IDCT(dct_coefs, norm='ortho') # inverse DCT
    return xt
```

```
def q_xt_x0(x0: array, t: int):
    xt1 = blur(x0, t)
    xt2 = blur(x0, t - 1)
    eps = xt2 - xt1
    return xt1, eps
```

```
def loss(x0: array, timestamps: array, inputs: array, t: int):
    # x0.shape = (pred_len,)
    # timestamps.shape = (input_len, 4 [= hours + days + weeks + months])
    # inputs.shape = (input_len,)
    t = randint(1, total_steps)
    xt, eps = q_xt_x0(x0, t)
    cond = embedding1(inputs) + embedding2(timestamps)
    error = adaptive(unet(xt, cond=cond, t=t), eps)
    return error
```

Table 4.1: Comparison of the MSE and $P3$ metrics for several models on five datasets. Each pair of columns compares the score of the baseline model (on the left) and its update after deblurring (on the right). The $P3$ results are put in bold when improved by the deblurring process.

		weather		**traffic**		**electricity**		**ETTm1**		**ETTm2**	
		Original	Deblur.	Original	Deblur.	Original	Deblur.	Original	Deblur.	Original	Deblur.
Informer	mse	1.116	1.124	1.695	1.796	0.907	0.925	1.297	1.309	0.546	0.549
	mae	0.767	0.770	0.862	0.879	0.766	0.776	0.866	0.870	0.531	0.532
	pmse	1.766	1.863	5.358	4.262	1.264	1.309	1.552	1.566	0.585	0.631
	pmae	0.942	0.965	1.567	1.522	0.895	0.984	0.985	0.990	0.510	0.530
	$P3_{sw}$	1.856	**1.760**	5.498	**5.118**	1.154	**1.087**	1.473	**1.450**	0.560	**0.515**
	$P3_{\mathcal{E}}$	1.770	**1.694**	3.823	**3.305**	0.881	**0.854**	1.277	**1.263**	0.547	**0.513**
Reformer	mse	0.776	0.778	0.742	0.848	0.348	0.375	0.876	0.886	1.634	1.665
	mae	0.662	0.663	0.391	0.405	0.416	0.449	0.722	0.726	0.915	0.923
	pmse	1.065	1.066	3.224	4.746	0.535	0.796	1.150	1.155	1.977	2.076
	pmae	0.710	0.704	0.979	1.446	0.514	0.706	0.832	0.833	1.011	1.041
	$P3_{sw}$	**1.074**	1.076	4.426	**3.465**	0.619	**0.492**	1.134	**1.093**	1.904	**1.799**
	$P3_{\mathcal{E}}$	1.029	**1.019**	2.466	**2.056**	0.433	**0.366**	0.945	**0.927**	1.769	**1.712**
FEDformer	mse	0.609	0.610	0.614	0.655	0.246	0.255	0.599	0.604	0.249	0.254
	mae	0.564	0.564	0.376	0.410	0.359	0.385	0.576	0.579	0.342	0.347
	pmse	1.066	1.061	2.605	4.406	0.377	0.967	0.666	0.658	0.208	0.211
	pmae	0.633	0.633	0.868	1.438	0.439	0.775	0.629	0.625	0.305	0.307
	$P3_{sw}$	1.049	**1.045**	4.329	**2.729**	0.430	**0.342**	0.632	**0.595**	0.201	**0.194**
	$P3_{\mathcal{E}}$	1.006	**0.997**	2.386	**1.664**	0.289	**0.255**	0.551	**0.538**	0.179	**0.178**
Performer	mse	0.506	0.509	0.709	0.842	0.378	0.417	1.105	1.121	0.440	0.445
	mae	0.525	0.526	0.376	0.384	0.438	0.487	0.819	0.824	0.492	0.495
	pmse	0.895	0.907	3.109	4.947	0.562	0.660	1.460	1.453	0.513	0.533
	pmae	0.630	0.637	0.939	1.172	0.529	0.744	0.955	0.954	0.536	0.544
	$P3_{sw}$	0.913	**0.906**	4.724	**3.307**	0.603	**0.529**	1.524	**1.461**	0.519	**0.502**
	$P3_{\mathcal{E}}$	0.856	**0.852**	2.593	**1.974**	0.470	**0.391**	1.241	**1.225**	0.488	**0.473**
Autoformer	mse	0.409	0.418	0.687	0.714	0.278	0.283	0.699	0.714	0.276	0.279
	mae	0.428	0.437	0.425	0.458	0.377	0.413	0.612	0.620	0.366	0.371
	pmse	0.625	0.636	2.892	3.230	0.421	0.526	0.754	0.750	0.340	0.338
	pmae	0.499	0.514	0.958	1.083	0.460	0.538	0.670	0.667	0.433	0.431
	$P3_{sw}$	0.642	**0.626**	3.022	**2.745**	0.365	**0.361**	0.708	**0.686**	0.321	**0.312**
	$P3_{\mathcal{E}}$	0.588	**0.587**	1.994	**1.829**	0.347	**0.344**	0.703	**0.698**	0.330	**0.329**
NHits	mse	0.304	0.306	0.565	0.613	0.233	0.264	0.514	0.515	0.233	0.233
	mae	0.350	0.353	0.370	0.401	0.325	0.354	0.503	0.504	0.331	0.331
	pmse	0.778	0.769	2.857	2.510	0.436	0.467	0.628	0.617	0.188	0.189
	pmae	0.478	0.483	0.970	1.055	0.459	0.485	0.578	0.573	0.283	0.284
	$P3_{sw}$	**0.785**	0.809	3.037	**2.246**	0.401	**0.374**	0.582	**0.565**	0.176	**0.173**
	$P3_{\mathcal{E}}$	0.715	**0.714**	1.680	**1.345**	**0.360**	0.363	0.514	**0.507**	0.165	**0.163**
FiLM	mse	0.322	0.324	0.690	0.744	0.278	0.284	0.555	0.560	0.237	0.239
	mae	0.334	0.340	0.424	0.445	0.359	0.362	0.499	0.502	0.324	0.327
	pmse	0.825	0.765	3.259	3.204	0.494	0.630	0.590	0.585	0.189	0.198
	pmae	0.461	0.438	1.134	1.115	0.484	0.561	0.547	0.539	0.280	0.286
	$P3_{sw}$	0.907	**0.788**	3.580	**3.099**	0.537	**0.482**	0.551	**0.516**	0.187	**0.185**
	$P3_{\mathcal{E}}$	0.736	**0.715**	1.977	**1.828**	0.392	**0.338**	0.471	**0.453**	0.170	**0.165**

		weather	**traffic**	**electricity**
EVL	mse	2.766	1.453	0.265
	$P3_{sw}$	3.144	4.505	0.399
	$P3_{\mathcal{E}}$	3.011	3.179	**0.302**
MSE loss	mse	**1.811**	**1.100**	**0.235**
	$P3_{sw}$	1.989	5.013	0.427
	$P3_{\mathcal{E}}$	1.978	3.381	0.390
MSE loss + deblurring	mse	1.985	1.112	0.254
	$P3_{sw}$	**1.814**	**3.920**	**0.386**
	$P3_{\mathcal{E}}$	**1.670**	**3.041**	0.358

Table 4.2: Comparison between DeDiPeak and EVL – EVT model, input length 336, forecast length 168.

		weather	**traffic**	**electricity**
DILATE loss	mse	0.140	0.812	0.358
	dilate	1.916	3.757	1.451
	$P3_{sw}$	0.600	2.074	0.515
	$P3_{\mathcal{E}}$	0.520	2.061	0.766
MSE loss	mse	**0.099**	**0.641**	**0.147**
	dilate	2.109	3.933	1.199
	$P3_{sw}$	0.248	2.634	0.430
	$P3_{\mathcal{E}}$	0.212	1.570	0.265
MSE loss + deblurring	mse	0.114	0.689	0.152
	dilate	**1.863**	**3.666**	**1.131**
	$P3_{sw}$	**0.236**	**2.011**	**0.290**
	$P3_{\mathcal{E}}$	**0.209**	**1.358**	**0.225**

Table 4.3: Comparison between DeDiPeak and DILATE – Seq2Seq model, input length 168, forecast length 24.

```
def deblur(x: array, timestamps: array, inputs: array,
           t: int, guidance: float = 5.0):
    # guidance = classifier-free guidance factor
    cond = embedding1(inputs) + embedding2(timestamps)
    dx = unet(x, cond=None, t=t)
    dxc = unet(x, cond=cond, t=t)
    dx = dx + (dxc - dx) * guidance
    x = x + dx  # (+ noise)
    return x
```

U-Net conditioning

The U-Net is composed of residual blocks at each level. Each of these residual blocks is conditioned both on the deblurring step (the integer t) and the input time series (noted c for *conditioning* or *context*). As shown in Figure 4.13, the input time series c is averaged along the time axis after being embedded. This forces the model to summarize the input time series into one vector, which is then added to each latent vector of the sequence being deblurred. This also allows the model to be conditioned on an input sequence of any length, adding more flexibility regarding the size of the available historical data.

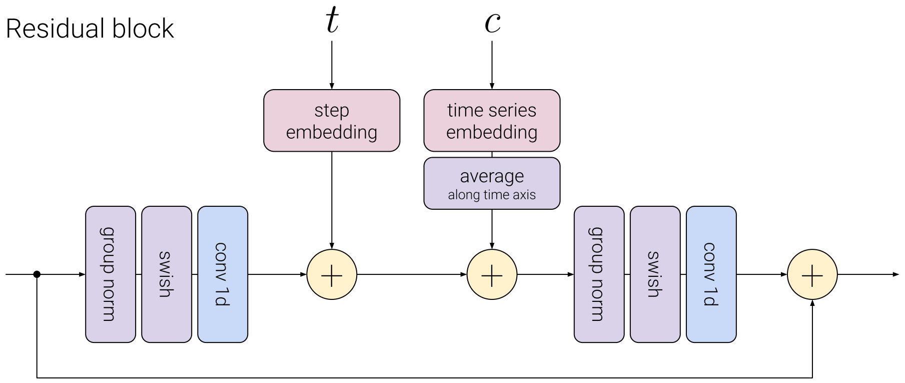
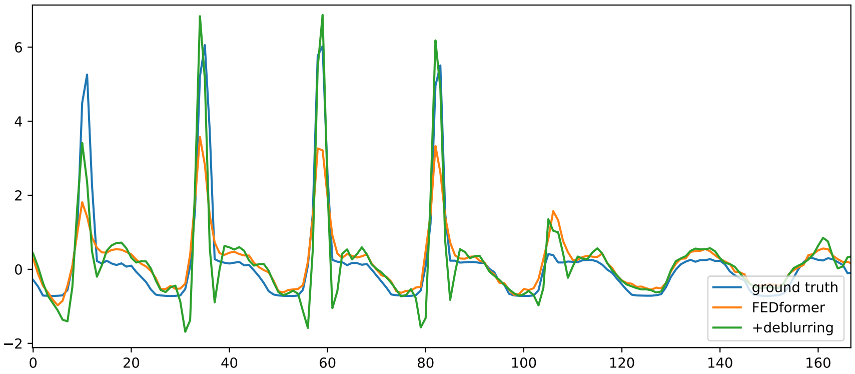


Figure 4.13: Definition of a residual block in the model. Each residual block in the U-Net is conditioned both on the deblurring step t, and the input time series c.

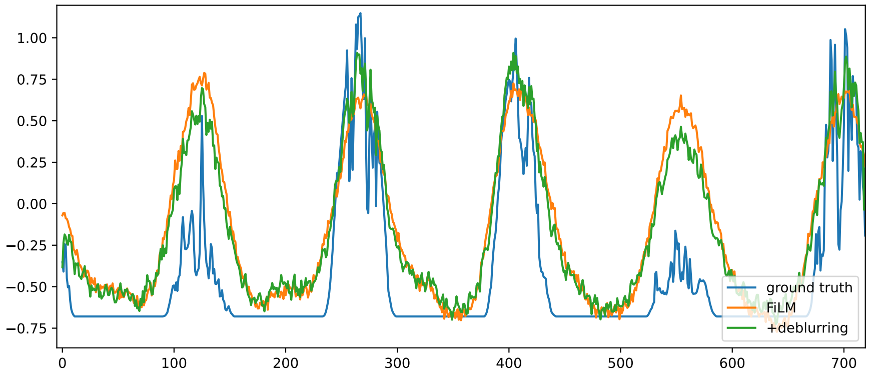
4.6.2 Additional Visual Results

The following figures illustrate the peak enhancement of the deblurring model on top of various base models for different datasets. The deblurring model improves peak forecasting either by correctly lowering (first peak of Fig. 4.15) or increasing (all the peaks in Fig. 4.16) the peak amplitude or even improving the peak timing (3rd peak in Fig. 4.14). These improvements are then captured by the dedicated peak metrics. However, the deblurring model also tends to include high-frequency noise far away from the peaks, sometimes degrading the overall forecasts. This is especially true in Fig. 4.14 where unwanted downward-facing peaks appear close to the actual peaks. In certain cases this observation leads to an increase of the point-wise loss functions (such as the MAE in Fig. 4.14).



	MSE	MAE	PMSE	PMAE	$P3_{sw}$	$P3_{\mathcal{E}}$
Informer	0.47	**0.36**	5.21	1.70	3.99	2.41
Deblurring	**0.35**	0.39	**1.36**	**0.71**	**0.60**	**1.51**

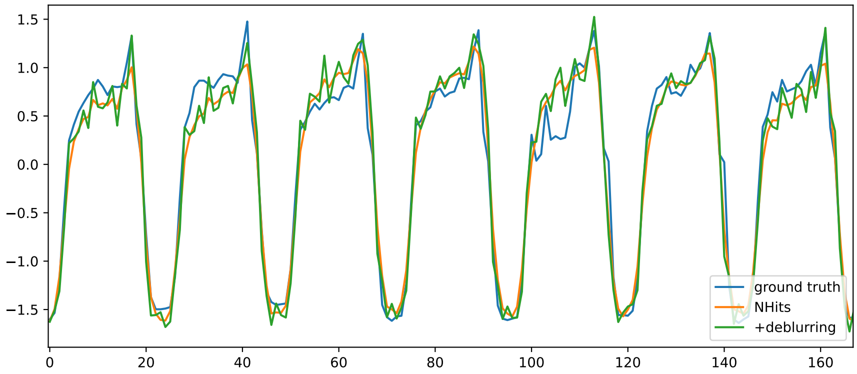
Figure 4.14: FEDformer on the *Traffic* dataset



	MSE	MAE	PMSE	PMAE	$P3_{sw}$	$P3_{\mathcal{E}}$
FiLM	0.17	0.31	0.25	0.46	0.24	0.30
Deblurring	**0.12**	**0.28**	**0.14**	**0.33**	**0.14**	**0.16**

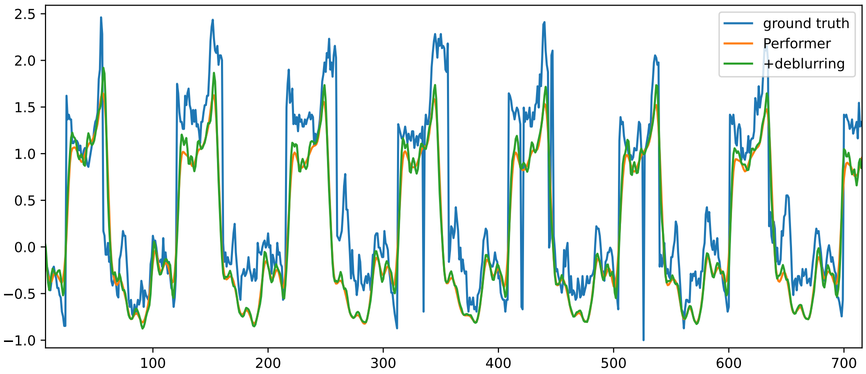
Figure 4.15: FiLM on the *Weather* dataset

The improvements for *ETTm1* on the performer are the least significant ones from the results shown here (Fig. 4.17). We observed a general trend that DeDiPeak performs better on bigger datasets, and thus its improvements on the *ETTm* datasets are smallest.



	MSE	MAE	PMSE	PMAE	$P3_{sw}$	$P3_{\mathcal{E}}$
NHits	**0.04**	**0.15**	0.09	0.25	0.09	0.08
Deblurring	0.04	0.16	**0.04**	**0.17**	**0.05**	**0.05**

Figure 4.16: NHits on the *Electricity* dataset



	MSE	MAE	PMSE	PMAE	$P3_{sw}$	$P3_{\mathcal{E}}$
Performer	0.33	0.41	1.19	1.00	0.45	0.60
Deblurring	**0.32**	**0.39**	**1.18**	**0.96**	**0.29**	**0.51**

Figure 4.17: Performer on the *ETTm1* dataset

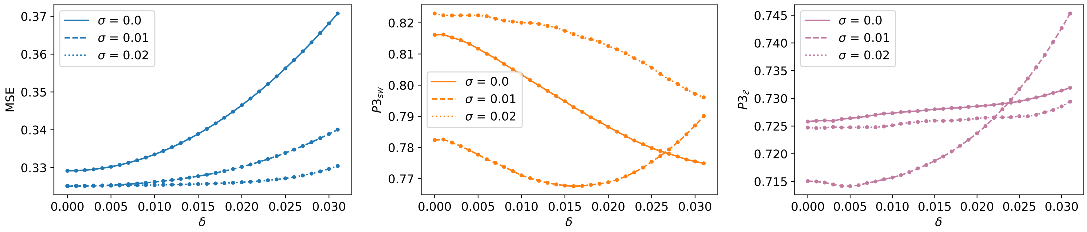
4.6.3 Effect of noise

Here we present additional experiments complementing the analysis on the dependence of the IHDM's performance on training and sampling noise. Each reported measurement in Fig. 4.18 represents the mean outcome derived from five independent experiments. As stated in the previous section, the choice of $\sigma = 0.01$ and $\delta = 0$ represents a reasonable trade-off between, on one hand, the speed of inference and the ease-of-use of the deblurring model and, on the other hand, the optimal performance of the IHDM. Note however that for some combination of base model and dataset, notably for FEDformer on the *ETTm1* dataset (Fig. 4.18 (b)), the performance of the IHDM can be further improved. In that case, it is indeed possible to bring small random variations to each deblurring iteration (`deblur` function, line 8 in Section 4.6.1).

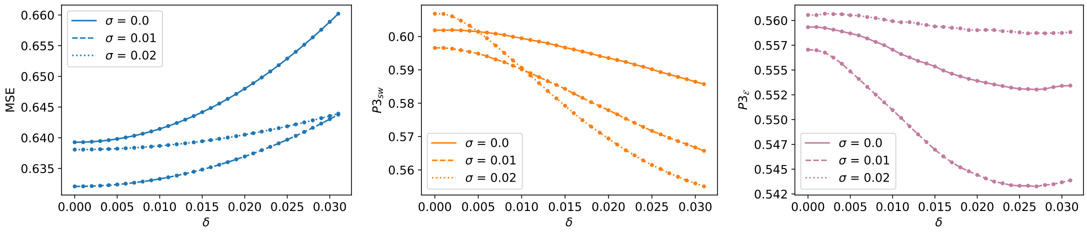
4.6.4 Metric parameters

The $P3$ metrics are equipped with tunable parameters – more precisely α, β, T, and n – which are assigned default values. This section discusses the influence of these parameters on the presented results.

Regarding α and β, we first need to note that y and t are not homogeneous quantities. While y represents an amplitude, t represents indices. Given their different units, a coefficient multiplication becomes necessary before summing $|t - t'|^2$ and $|y(t) - \hat{y}(t')|^2$, hence the role of α and β. Thus, the choice of α and β essentially addresses the question: *"How much should timing outweigh amplitude?"*



(a) FiLM / weather



(b) FEDformer / ETTm1

Figure 4.18: Dependence of the IHDM's predictive performance (MSE in blue, $P3_{sw}$ in orange, $P3_{\mathcal{E}}$ in violet) applied on top of the FiLM and FEDformer base models for the *weather* and *ETTm1* datasets on the variance of the sampling noise δ^2 and of the training noise σ^2.

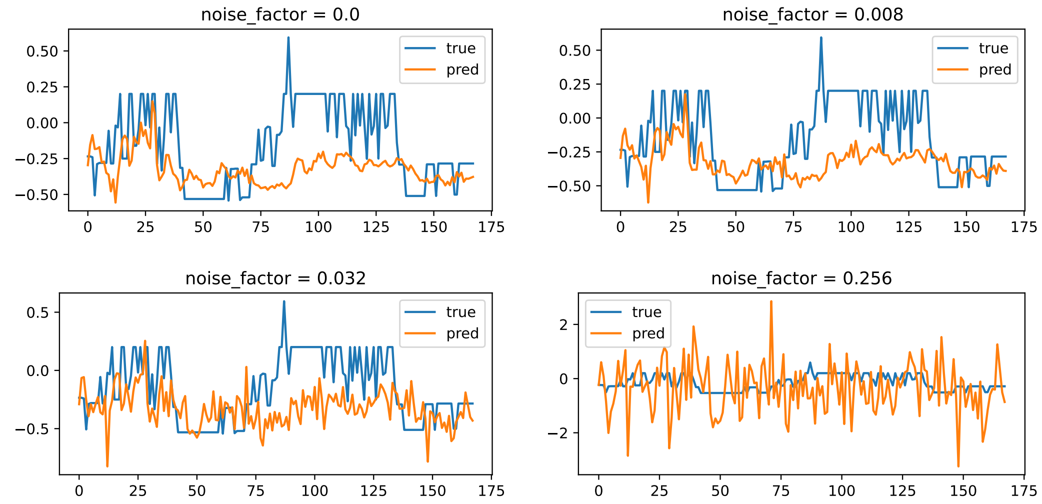


Figure 4.19: Impact of different values of sampling noise. Deblurring from the FiLM model on ETTm2.

$\beta < \alpha$ implies a larger time shift tolerance, thus assigning greater significance to the amplitude, whereas $\beta > \alpha$ prioritizes the time scale.

We illustrate the impact of the choice of α and β in Figures 4.20a and 4.20b by reporting the value of the metric as a function of the ratio α/β. In these figures, we observe that $P3_{\mathcal{E}}$ decreases as we tolerate a larger time shift.

Figure 4.20b reveals that increasing α/β (increasing the emphasis on timing) can, in some situations, make the performance of the deblurring model appear to perform worse than the base model while performing significantly better with lower α/β. This indicates that the amplitude improved after the diffusion process, but the time shift may have deteriorated slightly. Such a case can be observed in Figure 4.10, on the fourth peak, where the time shift changes from 0 to 1 step.

The pursued objective for setting the default parameters was to acquit the same importance to both the time and peak amplitude errors. Given that the data values are standardized, the time axis has a larger scale than the amplitude axis. Assuming according to the definition of peaks that there is a peak every n data points, the term $|t - t'|^2$ is of the order $\mathcal{O}(n^2)$, while the term $|y(t) - \hat{y}(t')|$ is independent of n and generally is of order $\mathcal{O}(1)$. Consequently, setting $\alpha = \frac{1}{n^2}$ and $\beta = 1.0$ first of all prevents timing errors from eclipsing amplitude errors and also brings both terms to the same order. This value has been standardized across datasets for consistency.

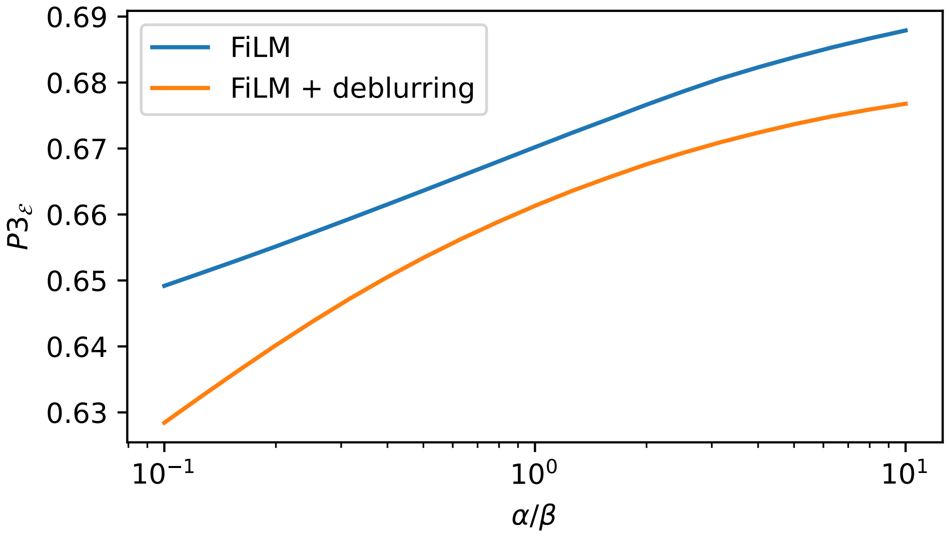
The number of neighbor n was set to 5 to avoid considering noise as peaks in our datasets. This implies that peaks should be at least separated by 5 time steps. For the *traffic*, *weather* and *electricity* datasets, this allows approximately 4 peaks in a day, whereas it allows up to 19 peaks per day for the ETTm datasets (which have a different time scale).

Finally, T was set to 5 as well to check for other peaks in this same time window at maximum. Further results about different values for n and T are presented in Table 4.4.

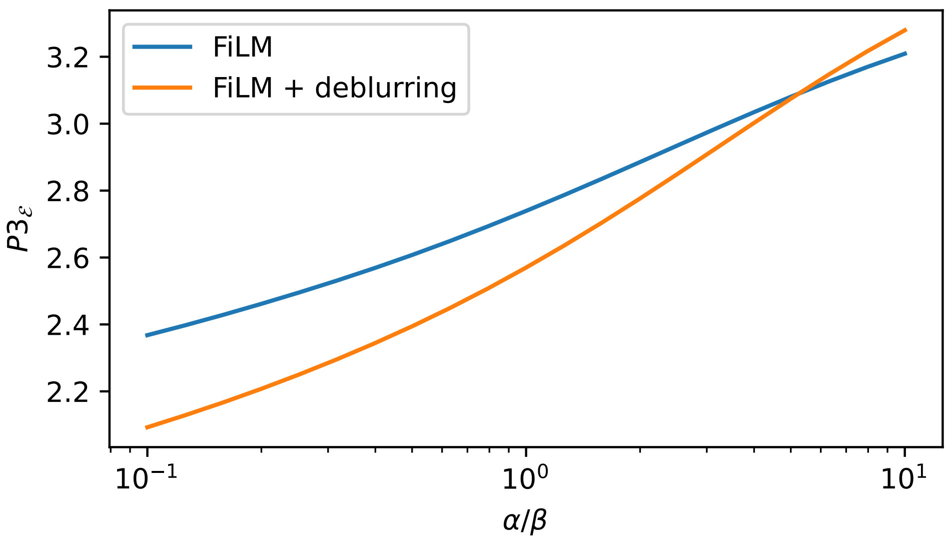
4.6.5 Inference time

In this short section, we compare the inference time of our approach with other baseline models on the *weather* dataset with a batch size of 16. We proceed to an evaluation with both 1 and 21 channels, as the inference time might not be proportional to the number of channels.

Although using a diffusion model has some computational overhead, the efficiency of the IHDM is good enough to enable daily real-world application: one hour is enough to forecast around 200,000 long-term time series forecasts on a single



(a) FiLM on the weather dataset



(b) FiLM on the traffic dataset

Figure 4.20: $P3_{\mathcal{E}}$ evolution when varying the ratio α/β

laptop GPU (RTX A2000), without parallelization. This should be contrasted with the currently suggested diffusion models in time series forecasting which would require several minutes on similar equipment. The key lies in reducing the number

	T											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\boldsymbol{n=2}$	-3.80	-5.28	-6.28	-6.94	-7.50	-8.01	-8.45	-8.78	-9.08	-9.36	-9.64	-9.79
$\boldsymbol{n=3}$	-4.82	-6.42	-7.43	-8.14	-8.77	-9.32	-9.76	-10.15	-10.54	-10.87	-11.19	-11.36
$\boldsymbol{n=4}$	-5.90	-7.53	-8.63	-9.34	-9.98	-10.55	-11.02	-11.43	-11.83	-12.19	-12.52	-12.72
$\boldsymbol{n=5}$	-6.06	-7.73	-8.82	-9.56	-10.19	-10.81	-11.36	-11.80	-12.22	-12.59	-12.94	-13.15
$\boldsymbol{n=6}$	-6.31	-7.98	-9.06	-9.84	-10.47	-11.06	-11.56	-12.03	-12.46	-12.87	-13.24	-13.45
$\boldsymbol{n=8}$	-6.90	-8.60	-9.72	-10.50	-11.13	-11.72	-12.25	-12.71	-13.14	-13.57	-13.96	-14.17
$\boldsymbol{n=10}$	-7.97	-9.66	-10.78	-11.53	-12.18	-12.78	-13.32	-13.79	-14.19	-14.60	-14.94	-15.13
$\boldsymbol{n=12}$	-7.61	-9.29	-10.38	-11.06	-11.71	-12.29	-12.77	-13.21	-13.58	-13.97	-14.29	-14.51
$\boldsymbol{n=15}$	-7.70	-9.39	-10.52	-11.22	-11.90	-12.51	-13.01	-13.46	-13.83	-14.18	-14.49	-14.76
$\boldsymbol{n=17}$	-7.76	-9.45	-10.52	-11.21	-11.91	-12.52	-12.99	-13.41	-13.75	-14.05	-14.34	-14.60
$\boldsymbol{n=20}$	-7.33	-8.88	-9.81	-10.40	-10.99	-11.51	-11.89	-12.26	-12.52	-12.75	-12.99	-13.19

Table 4.4: *Traffic* dataset: average variation of the $P3_{sw}$ in %, after applying the diffusion model on FiLM predictions. $P3_{sw}$ is expressed with different values of T and n.

	`y_batch.shape` = (16, 720, 1)
NHits	0.0055s
Autoformer	0.0781s
Informer	0.0251s
FEDformer	0.2326s
IHDM (15 deblurring steps)	0.6015s

	`y_batch.shape` = (16, 720, 21)
NHits	0.118s
Autoformer	0.142s
Informer	0.125s
FEDformer	0.330s
IHDM (15 deblurring steps)	6.2s

Table 4.5: Computation time comparison on the weather dataset, performed on an NVIDIA RTX A2000 Laptop GPU

of calls made during the inference process. The TimeGrad model, as an example, in its default configuration needs $100 \times 100 = 10,000$ model calls to produce a forecast, whereas only one baseline model call and 15 U-Net calls are sufficient in our configuration.

4.6.6 Performance of a probabilistic model

As mentioned in the *background* section of the chapter, recent probabilistic models are not trained with point-wise losses and thus might not suffer from the same problems as the non-probabilistic models we've used as baselines.

We reproduce the experiment on D3VAE [LLW^{+}22], a state-of-the-art probabilistic model for time series forecasting. However, the hardware specification used in our experimental settings is not sufficient for D3VAE to run on this dataset in the same conditions (lack of VRAM). Thus, we restrict the predicted sequence length to 168 values, instead of 720, and set the batch size to 8 instead of 16.

		weather	**traffic**	**electricity**	**ETTm1**	**ETTm2**
D3VAE 168 values forecast	mse	1.457	1.724	1.852	0.850	0.206
	mae	0.794	1.004	1.126	0.649	0.329
	pmse	2.808	4.482	2.310	0.958	0.241
	pmae	1.002	1.644	1.180	0.745	0.376
	$P3_{sw}$	2.804	3.434	1.336	0.864	0.179
	$P3_{\mathcal{E}}$	2.579	3.532	1.651	0.860	0.221
NHits + deblurring 720 values forecast	mse	0.306	0.613	0.264	0.515	0.233
	mae	0.353	0.401	0.354	0.504	0.331
	pmse	0.769	2.510	0.467	0.617	0.189
	pmae	0.483	1.055	0.485	0.573	0.284
	$P3_{sw}$	0.809	2.246	0.374	0.565	0.173
	$P3_{\mathcal{E}}$	0.714	1.345	0.363	0.507	0.163

Table 4.6: Comparison of the MSE and $P3$ metrics between the D3VAE model and Dedipeak/NHits on five datasets.

Table 4.6 indicates that D3VAE generally underperforms compared to our methods, despite having fewer values to forecast. Coupled with its higher resource requirements, this renders D3VAE currently impractical. However, it is noteworthy that in the specific context of ETTm2, D3VAE achieves scores nearly comparable to our best model, if we disregard the forecasting period. This underscores the potential of such models.

As it stands, recent probabilistic models are currently inadequate for efficiently and effectively enhancing peak prediction capabilities. Despite their promise, including the ability to quantify uncertainty around peaks, there remains a compelling avenue for future research in developing probabilistic models tailored for peak predictions. This research direction diverges from our focus, which aims to improve existing time-series forecasting models in their ability to more accurately predict peak amplitudes.

4.6.7 Peak rate

In this section, we compare the ability of baseline models to predict the advent of a peak, regardless their amplitude. Since our solution aims to adjust the amplitude of peaks, it is essential to choose a baseline model according to its ability to predict such peaks, before applying the deblurring process. Given a peak in the ground truth, with $n = 5$[5], we check if a peak was predicted with a timing tolerance of 5 time steps. We then measure the rate of peaks successfully predicted in the test set, or *peak rate*. This can be seen as a metric similar to the recall for a classification task.

	weather	**traffic**	**electricity**	**ETTm1**	**ETTm2**
FEDformer	0.175	0.815	0.734	0.331	0.346
Performer	0.139	0.543	0.558	0.246	0.194
Informer	0.126	0.310	0.357	0.199	0.196
NHits	0.174	0.795	0.722	0.314	0.330

Table 4.7: Proportion of peaks successfully predicted by different baseline models within 5 neighbors, regardless of their amplitude.

Table 4.7 reveals a clear disparity amongst models and datasets. NHits and the FEDformer significantly outperform the Performer and the Informer, which is consistent with their overall performances on those datasets. On the dataset side, *traffic* and *electricity* seem to have much more predictable peaks than the other ones. This can be explained by their seasonality and regular patterns, where peaks happen regularly, albeit with varying amplitudes. Regarding the *weather* dataset, all models perform poorly in predicting the presence of a peak. Determining when peak will happen is indeed one of the harder task in weather forecasting, more than determining their amplitude. Such tasks, where the presence of peak may be hard to predict – and might depend on several external factors – are then still a field of ongoing research, which might be very domain-specific.

4.7 Discussion

Limitations. Deblurring models are still in their early days and this work, to the best of our knowledge, is the first one to use them in the context of time series. One of their main limitations is their long inference time, stemming from the multiple calls to the U-Net, which ultimately limits their development as a standalone. While our modular design somewhat mitigates this factor, making IHDM more feasible for practical applications, the overhead remains significant. A second limitation

---

[5] i.e. a value above the average greater than its 5 previous and following neighbors

lies in the fact that, in some cases, the improvements provided to the base model are only moderate. Further studies into model compatibility for DeDiPeak might shed some light on the causes of this. There is also an inherent dependence of the deblurring model on the base model, as evident from the omitted fifth peak in figure 4.10. While our deblurring model is designed to enhance existing peaks, it faces challenges in creating non-existent ones. We put this concern into perspective with a brief study in the Appendix.

Finally, we have established default parameters for the $P3$ metrics, equipping them for immediate application based on our experimental setup. The selection of these parameters can significantly impact outcomes and should be considered carefully according to the desired peak properties.

Future work. We plan to tackle these issues by investigating possible parameter optimizations such as the number of blurring and deblurring steps as well as the customization of the attention mechanisms in the U-Net. One further improvement we would like to investigate would be to only add data points from the vicinity of the diffusion-generated peaks to the initial prediction of the base model and keep the rest of it. This strategy could potentially solve the worsening of the prediction values far away from peaks

Conclusion

In this work, we introduce a novel framework named DeDiPeak aimed at improving peak forecasting of time series forecasting models. The proposed framework includes two novel parameterized metrics for evaluating the peak prediction capabilities of time series models, along with a deblurring diffusion model that can be plugged into the output of state-of-the-art solutions to enhance their peaks. Empirically, we highlight the necessity of our suggested metrics, establish their validity and demonstrate the benefits of adding the deblurring models to obtain better and clearer peaks.

5

Leveraging External Factors in Household-Level Electrical Consumption Forecasting using Hypernetworks

This chapter addresses a key challenge in electrical consumption forecasting where adding external factors like weather data improves individual household models but degrades global models trained on entire populations. We propose a hypernetwork architecture that successfully leverages external factors to enhance global forecasting accuracy by adjusting model weights for each consumer, demonstrating superior performance on the CREOS dataset.

Contents

5.1 Context 72
5.2 Early attempts in Transplit adaptation 74
5.3 Background 76
5.4 Hypernetworks for Time Series Forecasting 77
5.5 Results 82
5.6 Discussion 86
5.7 Conclusion 87

5.1 Context

In the specific case of electrical load forecasting, numerous studies have demonstrated that consumption patterns are heavily influenced by external factors such as weather conditions, calendar effects, and socio-economic indicators [HF16]. Temperature, in particular, has been shown to have a strong relationship with electricity demand, as heating and cooling needs vary significantly with ambient temperature [CXC$^+$17]. Additionally, calendar variables including holidays, weekends, and seasonal patterns have been shown to capture regular variations in consumption behavior effectively [WCH$^+$19][ZZL$^+$17]. These behaviors, however, are household-specific — e.g., a household using electric heating has a consumption more sensitive to cold temperatures than a household relying on gas. This represents a challenge to global forecasting models, which therefore have to capture specific behaviors when predicting the consumption.

In order to forecast consumption, two strategies can be distinguished:

- **Global model:** A unified model trained on aggregated data across the entire consumer population. This centralized approach facilitates comprehensive pattern recognition across diverse consumption behaviors, enhancing generalization capabilities while minimizing computational infrastructure requirements. Furthermore, recent architectural innovations specifically address multi-channel time series [LHZ$^+$24][XZW$^+$24].
- **Individual models:** A dedicated model trained for each consumer entity. These specialized models capture household-specific consumption patterns with high fidelity. While traditionally resource-intensive in terms of computation and storage, recent advances in federated learning mitigate these constraints [SO21], though hardware limitations for on-device machine learning deployment remain significant.

To compare these two paradigms, we assess them on the CREOS dataset — containing more than 6000 households consumptions over two years — and corresponding external factors, ranging from weather data to football[1] events. As our results later demonstrate, although incorporating external factors as features theoretically enhances performance, these lead to overall performance degradation in global models. Conversely, individual models excel at mapping external factors to consumer-specific responses, but introduce substantial computational and storage overhead that scales linearly with the consumer population. In particular, this approach fails to capitalize on the substantial behavioral similarities across consumers. Since many households share comparable consumption patterns [WCK$^+$16], training completely separate models results in significant parameter redundancy, as each individual model essentially learns the same forecasting task (electricity

---

[1] *Football* in this paper refers to *"soccer"*

consumption) with variations to accommodate specific consumer profiles. This redundancy wastes computational resources and misses opportunities for knowledge sharing across similar consumer segments.

In order to bridge the gap between global models efficiency and individual models precision, hypernetworks offer a promising architectural paradigm, illustrated in Figure 5.1. Hypernetworks [HDL16] are meta-models designed to generate the weights of a primary task network conditioned on specific inputs. In our context, a hypernetwork can dynamically produce customized parameters for each consumer based on their unique embedding and current situation. This approach maintains the personalization advantages of individual models while dramatically reducing the parameter space, rather than maintaining thousands of separate forecasting models.

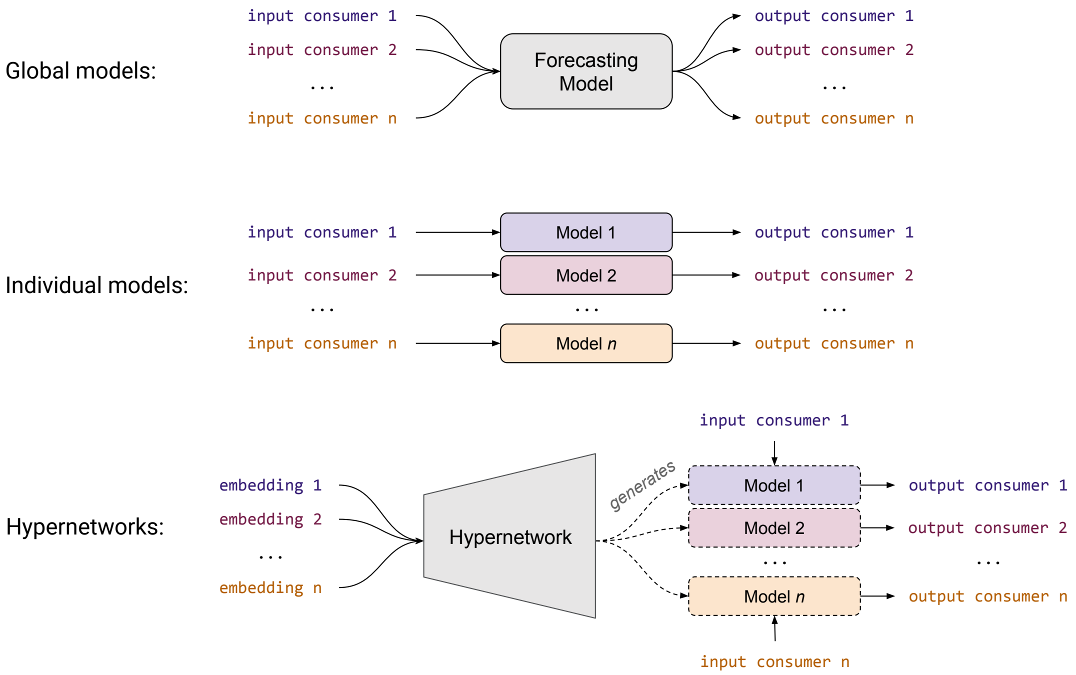


Figure 5.1: Difference between global and individual models, and the proposed in-between solution using hypernetworks.

In this chapter, we introduce a novel approach using hypernetworks and consumer-specific embeddings that enable global models to differentiate between individual households. These compact embeddings require minimal storage compared to full individual model parameters while preserving household-specific information. Our experimental results demonstrate that the hypernetwork architecture is the only one in the tested benchmark to leverage external factors to reduce forecasting error — and ultimately get the lowest error, beating state-of-the-art models by up to 16% — while conventional approaches result in performance degradation. This improvement enables more accurate, individualized forecasting within a computationally efficient framework.

5.2 Early attempts in Transplit adaptation

Just as in the previous chapter, it is worth mentioning one particular attempt that could have perhaps better plug into our existing framework, but that effectively failed in handling external factors w.r.t. consumer profiles.

Introducing the information on the consumer first necessitates the creation of an embedding. The simplest approach is to take our existing model Transplit and adapt the attention mechanism to make it capable of handling external factors **with respect to** the consumer. It is worth emphasizing again: the way we should handle external factors really depends on the consumer profile we are dealing with.

Intuition. The core hypothesis behind our approach is that different consumers exhibit varying sensitivities to external factors. For instance, a household with electric heating will show strong correlation with temperature variations, while a household with gas heating may be largely insensitive to temperature but highly responsive to daylight patterns due to lighting usage.

Mathematically, we can formalize this as follows. Let $\mathbf{c} \in \mathbb{R}^d$ represent the consumer embedding, and $\mathbf{f}_i \in \mathbb{R}^d$ represent the i-th external factor embedding. The consumer-specific sensitivity to factor i can be computed as:

$$\alpha_i = \sigma(\mathbf{c}^T \mathbf{f}_i) \tag{5.1}$$

where σ is the sigmoid function ensuring $\alpha_i \in [0, 1]$. The scalar product $\mathbf{c}^T \mathbf{f}_i$ captures the alignment between the consumer's characteristics and the factor's influence pattern. A high positive value indicates strong consumer-factor affinity, while values close to zero suggest minimal impact.

The weighted representation of external factors for consumer c becomes:

$$\mathbf{h}_{external} = \sum_{i=1}^{N} \alpha_i \mathbf{f}_i = \sum_{i=1}^{N} \sigma(\mathbf{c}^T \mathbf{f}_i) \mathbf{f}_i \tag{5.2}$$

This formulation allows the model to automatically learn which external factors are most relevant for each consumer type, creating a personalized weighting scheme that should theoretically improve forecasting accuracy.

In a nutshell and explained less formally, the implementation process follows three main steps:

- **Projection:**
  - The consumer's vector is taken from an embedding table;

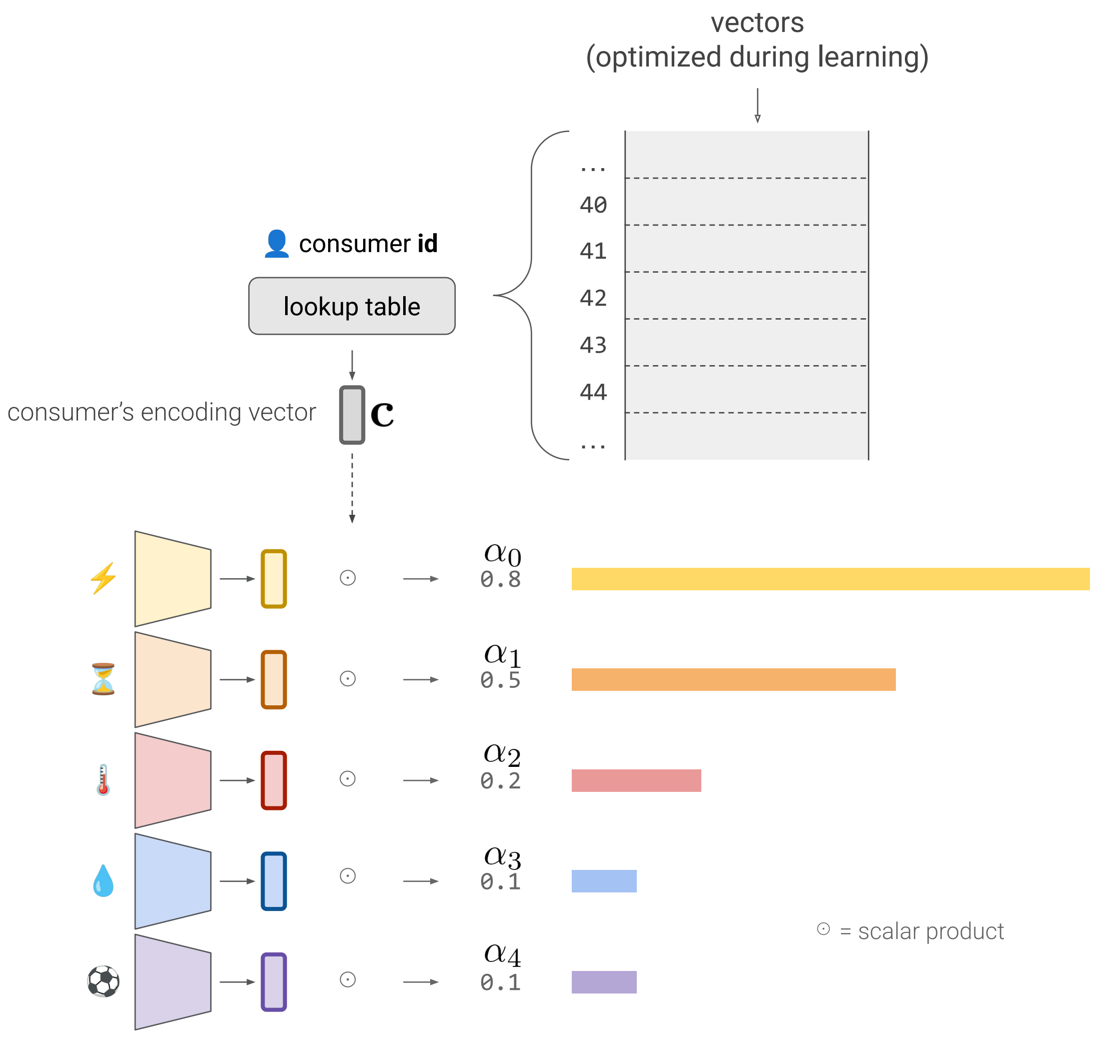


Figure 5.2: Weighting the different external factors with the consumer embedding in the custom attention mechanism. Each consumer embedding **c** is used to compute attention weights α_i for external factors through scalar products, creating a personalized representation of environmental influences.

  - Each factor is projected into one vector;
- **Scalar product:** every factor's vector is compared with the user representation (Figure 5.2);
- **Weighting:** the next sequence representation inside the model is a weighted sum of the different factors ones.

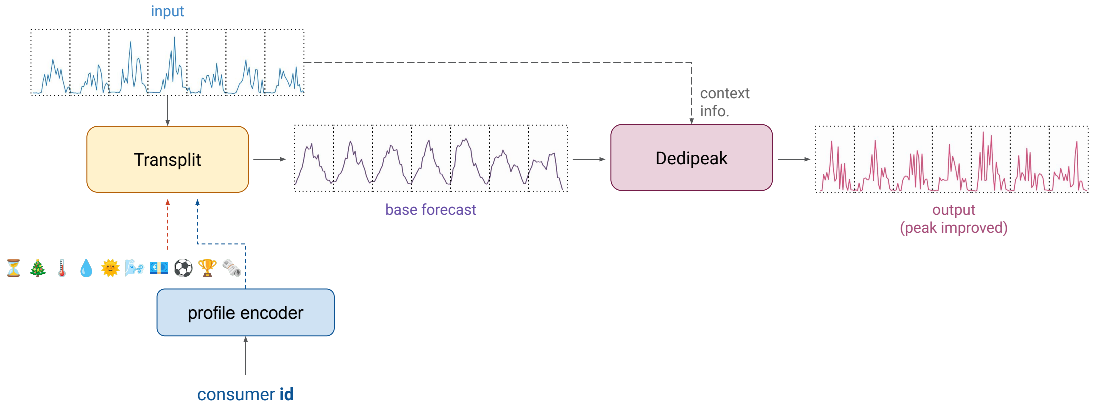


Figure 5.3: Overview of the whole forecasting pipeline, with the initial idea of adapting Transplit to handle external factors w.r.t. a consumer embedding.

Experimental Results and Limitations

The experimental evaluation revealed disappointing results. For a 1-week → 1-week forecast configuration, the baseline Transplit model achieved an MSE of 0.192. Adding external factors without consumer-specific weighting yielded an MSE of 0.193, showing no meaningful improvement. Including our specific attention mechanism with consumer-weighted external factors resulted in an MSE of 0.192, essentially identical to the baseline.

These results suggest several potential issues with this approach:

1. **Insufficient model capacity:** The consumer embeddings may not capture enough nuanced information to effectively differentiate factor sensitivities;
2. **Data sparsity:** Individual consumers may not have sufficient data to learn meaningful embeddings;
3. **Architectural limitations:** The scalar product attention mechanism may be too simplistic to capture complex consumer-factor interactions.

Notably, small models trained exclusively on individual consumer data performed significantly better, highlighting the importance of consumer-specific modeling approaches. This observation motivated our subsequent investigations into more sophisticated consumer-centric architectures, which we explore in this chapter.

5.3 Background

When it comes to electricity load forecasting, a critical challenge is to effectively incorporate multiple information channels, including historical consumption and various exogenous factors. Recent architectures specifically target this *multivariate* challenge: iTransformer [LHZ+24] revolutionizes time series modeling by treating individual features as tokens and timestamps as channels, inverting the traditional approach. PatchTST [NNS+23] applies patching strategies to decompose time series into subseries, enabling more robust feature extraction. Lately, CARD [XZW+24] introduced channel attention mechanisms that dynamically weight the importance of different input variables.

These models however still have to process the input time series to figure out the consumer's profile, which can be highly different from one time series to another. Additionally, recognizing consumers profiles may also require longer input time series (e.g. in order to analyze their behaviors during vacations). Mixture of Experts (MoE) models [JJN+91] offer another approach to handling heterogeneous patterns in time series data. These architectures dynamically route inputs to specialized subnetworks, allowing the model to develop expertise given a specific embedding. Mixture of Linear Experts (MoLE) [NLW+24] extends this concept by creating embeddings that represent input characteristics in order to create this embedding, further improving adaptability to diverse time series behaviors.

Hypernetworks [HDL16] represent a powerful paradigm where one network generates the weights for another. In the domain of time series, this approach has shown particular promise for addressing distribution shifts in time series [DHZ+23] and has been applied to implicit neural representations as demonstrated in HyperTime [FSE+22]. Hypernetworks are especially relevant for our work as they can efficiently generate consumer-specific parameters from compact embeddings, potentially capturing individual household behaviors without requiring separate models for each consumer.

In the context of electricity load forecasting, these architectural innovations offer promising directions for improving prediction accuracy while maintaining computational efficiency. Our work builds upon these foundations to address the specific challenges of capturing consumer-specific responses to exogenous factors.

5.4 Hypernetworks for Time Series Forecasting

5.4.1 Problem Formulation

We address the task of forecasting electrical consumption time series for a diverse set of consumers while incorporating various external factors. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ represent the set of N consumer entities, each with its own hourly electrical consumption time series. For each consumer x_i, we denote its consumption at time t as $x_{i,t} \in \mathbb{R}$. Additionally, we have a set of numerical external factors $\Phi = \{\phi_1, \phi_2, \ldots, \phi_k\}$ (additional time series, such as temperature) and categorical external factors $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$.

Our objective is to predict future consumption values $y_{i,t:t+h} := x_{i,t+L:t+L+h}$ for a horizon h for every consumer i, given historical consumption $x_{i,t:t+L}$ for an input length L and external factors $\Phi_{t:t+L}$ and $\mathcal{C}_{t:t+L}$.

5.4.2 Model Architecture

Our proposed architecture consists of three main components: (1) an embedding layer for categorical variables, (2) a hypernetwork that generates consumer-specific weights, and (3) a linear forecasting model with these consumer-specific weights. The hypernetwork itself can be seen as a weights generator — that outputs matrices for the linear model — and essentially shares the same architecture than an image decoder [VV+17a]. An overview of the pipeline is illustrated in Figure 5.4.

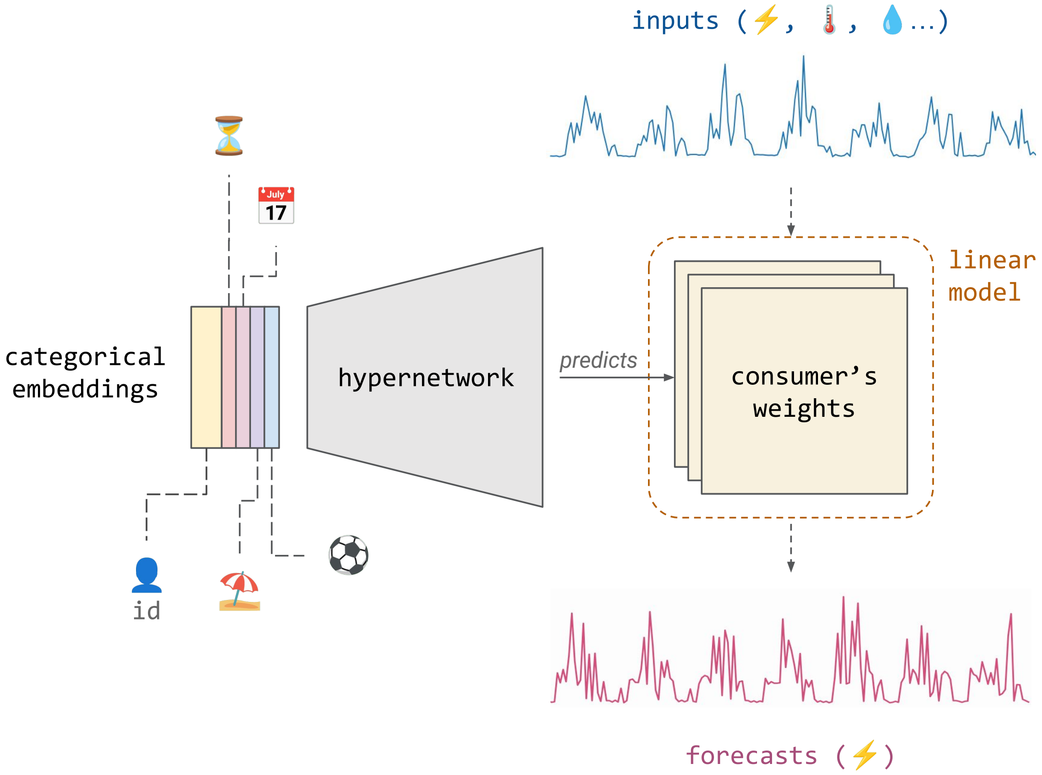


Figure 5.4: Overview of the hypernetwork pipeline

Embedding Representation for Categorical Variables.

For each categorical external factor $c_j \in \mathcal{C}$, we learn a dense embedding representation:

$$\mathbf{e}_j = \text{Embed}(c_j) \in \mathbb{R}^{d_j} \tag{5.3}$$

where d_j is the embedding dimension for factor j. Specifically, when categorical features are related and complementary, we sum their embeddings as follows:

$$\mathbf{e}_{\text{event}} = \begin{cases} \mathbf{e}_{\text{no event}}, & \text{if } c_{\text{event}_k} = 0 \text{ for all } k \\ \sum_{k \in \{k | c_{\text{event}_k} = 1\}} \mathbf{e}_{\text{event}_k}, & \text{otherwise} \end{cases} \tag{5.4}$$

All categorical embeddings are reshaped to matrices of size (p, q) and stacked together to form the hypernetwork input, as shown in Figure 5.5. The resulting input tensor is denoted $\mathbf{z}_{i,t}$. The output matrices predicted by the hypernetwork have proportional dimensions from the inputs, and are of shape $(p \times u, q \times u)$, where u is the upscaling factor.

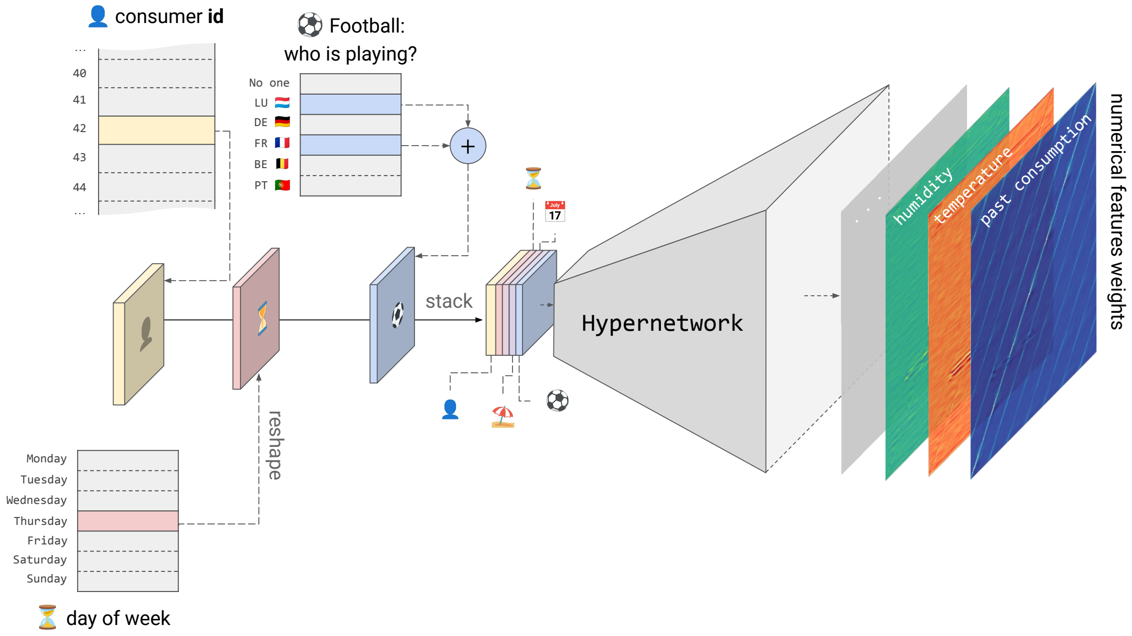


Figure 5.5: Illustration of example embeddings. Each consumer ID and other known categorical features are transformed to embeddings, which are reshaped and stacked together to form the Hypernetwork input.

Forecasting Mechanism.

The hypernetwork H_θ with parameters θ takes the concatenated features $\mathbf{z}_t$ and generates the weights for a consumer-specific linear forecasting model:

$$\mathbf{W}_{i,t} = H_\theta(\mathbf{z}_{i,t}) \in \mathbb{R}^{L \times h \times p} \tag{5.5}$$

where $p = k + 1$ is the input dimension to the linear model, corresponding to the number of input time series; L is the input length, and h is the forecast horizon.

The consumer-specific weights $\mathbf{W}_{i,t}$ are then used in a linear model to produce the final forecasts. For each consumer i at time t, the input to the linear model includes both historical consumption values $x_{i,t:t+L} \in \mathbb{R}^L$ and the numerical external factors $\Phi_{t:t+L} \in \mathbb{R}^{k \times L}$.

The forecast for the next h time steps is then computed as:

$$\hat{\mathbf{y}}_{i,t:t+h} = \mathbf{W}_{i,t} \cdot \begin{pmatrix} x_{i,t:t+L} \\ \phi_{1,t:t+L} \\ \cdots \\ \phi_{k,t:t+L} \end{pmatrix} \tag{5.6}$$

Loss Function and optimization.

We jointly optimize the hypernetwork parameters θ along with all categorical feature embeddings e_i by minimizing the Mean Squared Error (MSE) between predictions and ground truth:

$$\min_{\theta,\{e_i\}} \sum_{i=1}^{N} \sum_{t \in \mathcal{T}} \|\hat{\mathbf{y}}_{i,t:t+h} - \mathbf{y}_{i,t:t+h}\|^2 \tag{5.7}$$

where N is the number of consumers, $\mathcal{T}$ is the set of time points in the training data, $\hat{\mathbf{y}}_{i,t:t+h}$ represents the predicted values, and $\mathbf{y}_{i,t:t+h}$ represents the ground truth values.

It is worth noting that unlike traditional neural networks where weights are directly optimized, in our approach, the hypernetwork parameters θ are optimized such that they can generate effective consumer-specific weights $\mathbf{W}_{i,t}$ for the linear forecasting model. This approach allows the model to dynamically adapt to different consumers' consumption patterns while leveraging shared knowledge across the entire consumer base.

5.4.3 Experimental setup

Dataset.

We use the CREOS dataset along with external factors over corresponding period (2020 and 2021):

- **Numerical features:**
  - x: Consumption data (kWh) for $N = 6,010$ households and businesses in Luxembourg;
  - $\phi_{\text{temp}}, \phi_{\text{hum}}, \phi_{\text{wind}}, \phi_{\text{sun}}$: Weather indicators — temperature (°C), humidity (%), wind speed (km/h), sunlight (minutes of sun within one hour)[2];

---

[2] For simplicity purposes, these indicators are global for all consumers (weather in Luxembourg City), as the geographical area of study is small with few local variations.

- **Categorical features:**
  - i: Consumer ID, ranging from 0 to 6,009;
  - $c_{\text{hour}}, c_{\text{dw}}, c_{\text{dm}}, c_{\text{month}}$: Timestamps data — hour of day (24 values), day of week (7), day of month (31), month of year (12);
  - $c_{\text{sh}}, c_{\text{ph}}$: School holiday indicator (boolean), public holiday (boolean);
  - $c_{\text{team}_1}, \ldots, c_{\text{team}_5}$: 5 booleans, indicating wether Luxembourg, Germany, France, Belgium or Portugal will be playing in the current day or not — which are relevant teams for the studied region.

As it is usual for electric load forecasting [GLA22], we set a forecast horizon of 1 week ($h = 168$), from an input length of 2 weeks ($L = 336$). We compare results with and without the inclusion of external factors, and run further experiments where only the consumer ID is provided in addition to electrical consumption. The dataset is partitioned chronologically into train/validation/test sets with standard 70%/10%/20% ratios following established time series forecasting protocols [whXW+21]. We preprocess the data by standardizing consumption values, temperature, and wind speed, while applying min-max normalization to humidity and sunlight variables as these represent naturally bounded quantities.

Hyperparameters.

We set the upscaling factor u to 24, which fits with the daily seasonality characteristics of electricity consumption. Given this factor and the needed sizes of the output matrices (336×168), we have to make inputs of size 14×7. To achieve this, we concatenate two 7×7 matrices, leading to 49-dimensional vectors. One reason for this choice is the flexibility this concatenation offers: one could easily change the number of weeks in the input length or forecast horizon by getting shapes of $7a \times 7b$. For consumer IDs, we allocate twice the embedding capacity ($7 \times 7 \times 2$) to capture the more complex behavioral patterns associated with individual users. These embedding tensors are concatenated along the channel dimension before being processed by the model through four residual blocks, ultimately generating weight matrices of dimension 336×168 that map input sequences to forecast horizons. Experiments are repeated 10 times to reduce randomness effects.

Baseline models.

One natural additional solution to experiment with is Mixture of Linear Experts [NLW+24], as they demonstrate strong performance in general time series forecasting. Especially, each expert can specialize in specific groups of consumers, and embeddings can simply be used to attribute expert importance. We consider three MoLE variants, MoLE_DLinear, MoLE_RLinear and MoLE_RMLP, the latter consisting in two dense layers expert models. 16 experts are used, as this setting allowed the good performance shown in [NLW+24]. When using categorical features, we use the same embeddings as for hypernetworks, which are then linearly mapped to a probability distribution vector that assigns experts importance.

Baseline models also include state-of-the-art forecasting models with a focus on multiple channels processing: iTransformer (2024 [LHZ$^+$24]), CARD (2024 [XZW$^+$24]), NHits (2023 [COO$^+$22]), PatchTST (2022 [NNS$^+$23]), RLinear (2022 [LQL$^+$23]). For completeness, we include ARIMA as a classical statistical baseline which, despite its computational complexity, often provides competitive performance for structured time series forecasting tasks. Since the baseline models are designed for continuous multivariate time series, we adapt categorical features for fair comparison. For most categorical variables, we employ one-hot encoding to create additional binary channels. However, for the high-cardinality consumer ID feature, this approach would create an impractical number of channels. Instead, we learn low-dimensional embeddings for consumer IDs and repeat these embeddings across the temporal dimension, maintaining consistent representation while controlling dimensionality. The code is available on Github[3].

Finally, we compare these results with individual RLinear models being trained for every individual consumer — not predicted by the hypernetwork — in contrast with global models cited above.

Infrastructure.

We use a Quadro RTX 8000 49GB GPU for all the experiments.

5.5 Results

Our experimental results demonstrate several key findings regarding the performance of various time series forecasting models, as shown in Table 5.1. The comparison across different input configurations yields important insights for model selection and deployment in real-world scenarios. The standard error is always $< 10^{-4}$ in the table, with two minor exceptions. More detail is provided in appendix.

5.5.1 Impact of External Factors

Perhaps the most surprising finding is that incorporating external factors generally degrades model performance across almost all architectures. This contradicts the common assumption that additional information should improve predictive accuracy. Only individual models and our hypernetwork approach exhibit improved performance when leveraging external factors, with decreases in both MSE and MAE compared to using consumer ID only or no external factors.

This exceptional behavior of hypernetworks suggests they possess a unique ability to effectively filter and use external information without introducing additional noise or complexity that harms prediction accuracy. The architecture's approach

---

[3] `https://github.com/serval-uni-lu/hypernetworks-time-series`

Table 5.1: MSE and MAE values for different models and datasets. Models denoted with an asterisk * are not meant to handle categorical features: the consumer's ID embedding is provided in additional time series channels

Model	No external factor		Consumer ID only		+ External factors	
	MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
Our method	-	-	0.1771	0.1872	<u>0.1734</u>	<u>0.1805</u>
MoLE_DLinear	0.1788	0.1899	0.1798	0.1891	0.1807	0.1904
MoLE_RLinear	0.1786	0.1836	0.1795	0.1844	0.1787	0.1839
MoLE_RMLP	0.1774	0.1820	0.1788	0.1832	0.1778	0.1826
Individual RLinears*	-	-	0.1741	0.1819	**0.1725**	**0.1792**
RLinear*	0.1806	0.1901	0.1874	0.1990	0.1888	0.2044
iTransformer*	0.1834	0.1867	0.1866	0.1894	0.1969	0.1966
CARD*	0.1759	0.1816	0.1760	0.1817	0.1765	0.1822
NHits*	0.1757	0.1849	0.1759	0.1851	0.1763	0.1854
PatchTST*	0.1759	0.1817	0.1762	0.1822	0.1768	0.1856
ARIMA	0.1780	0.1893	-	-	-	-

to handling multiple input channels appears fundamentally more effective than competing methods.

Consumer ID Embeddings. The performance when using only consumer ID embeddings as additional channels provides insights into how different models handle the introduction of this information. The MoLE models are the only global ones to improve the forecasting quality with the consumer ID provided — they are, however, with the hypernetworks, the only models designed to handle this specific input. Models not explicitly designed for this purpose always show a small degradation of performance. Despite not being optimized for categorical features, Transformer models still perform reasonably well in this scenario.

5.5.2 Performance Across Model Architectures.

The Hypernetwork architecture exhibits superior performance compared to other models by successfully *imitating* the individual models approach and getting closer to its final performance, achieving the second lowest MSE (0.1734) and MAE (0.1805) when incorporating external factors. This represents a notable improvement over traditional approaches and even other deep learning models. CARD and NHits follow closely behind, with NHits demonstrating particularly strong performance (MSE: 0.1763, MAE: 0.1854), making it a viable alternative when no external factors are available.

Interestingly, the classical Arima model (MSE: 0.1780, MAE: 0.1893) remains competitive despite being significantly less complex than the deep learning approaches. This suggests that for certain forecasting tasks, traditional statistical methods should not be dismissed outright.

5.5.3 Cost

Training time. Our hypernetwork approach achieves a favorable trade-off between computational resources and prediction accuracy. While generating consumer-specific weights introduces additional computational overhead during training compared to global models, this cost is substantially lower than training individual models for each consumer. Specifically, our approach reduces training time by 7 hours (approximately 70%) compared to individualized RLinear models.

Memory. The memory efficiency of our approach is particularly notable. The consumer embeddings require only 589K parameters (2.4MB), whereas individual linear models for all 6,010 consumers demand 3.392 billion parameters. This represents a parameter reduction factor of over 5,700×. Extrapolating to a real-world deployment with 1 million consumers, our approach would require only megabytes of storage compared to approximately 2.3TB for individual models. This dramatic reduction in model size not only decreases storage requirements but also eliminates the significant I/O overhead that would occur when loading individual models from disk during inference — a practical consideration not captured in our GPU-only-based timing experiments.

5.5.4 Generalizing consumers embeddings

As consumers might evolve over time, with new ones arriving and others leaving, embeddings often need to be updated. This can be achieved by optimizing the embeddings in order to reduce the final forecasting error. One advantage of this method is that this task can be easily parallelized, and the hypernetwork model itself doesn't necessarily need to be retrained. Figure 5.7 reveals that our hypernetwork approach, when trained on merely 8% of the consumer base (500 out of 6010 consumers), outperforms competing models across the entire dataset, given consumers' embeddings after training. This adaptive capability presents a significant advantage in dynamic real-world settings where consumer populations continually evolve, as the model maintains strong predictive performance while requiring minimal retraining.

5.5.5 Ablation studies

Inclusion of categorical features.

As several models are not explicitly designed to handle categorical features, it is important to verify that these models are not penalized by such inclusions. Results in Table 5.2 show that categorical features have overall no significant impact on

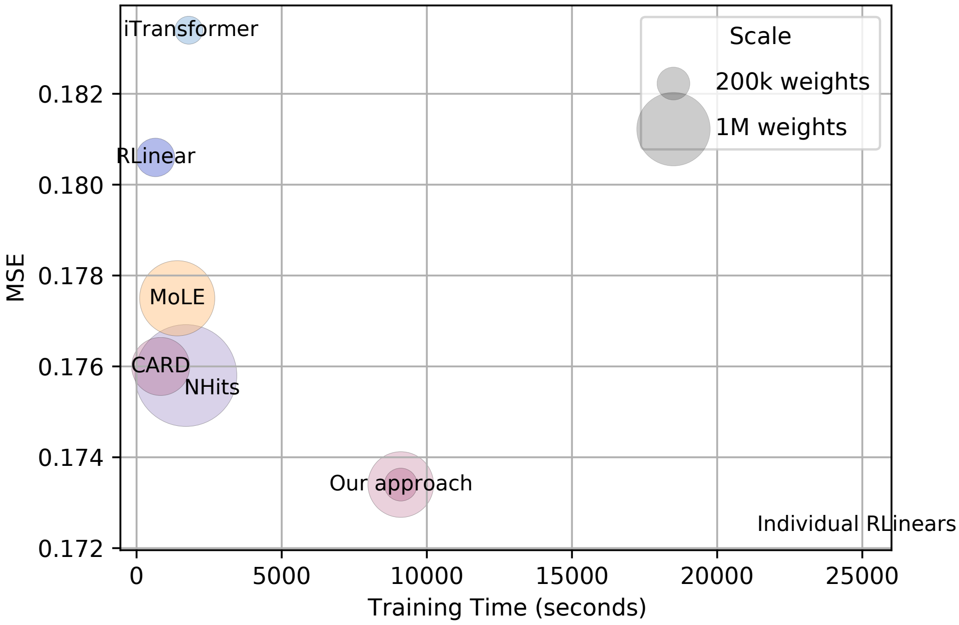


Figure 5.6: Comparison of models training time vs. the best resulting MSE. Bubble sizes refer to the number of weights. For our approach, we distinguish the size of the hypernetwork itself, and the size including all 6010 consumers embeddings. The size for individual RLinear models (bottom right) is not shown as it would fill in all the figure.

their performance, with MSE varying by at most 5×10^{-4} and MAE by at most 7×10^{-4}. Some models even show marginal improvements with categorical features (e.g., NHits exhibits lower MSE and MAE with categorical features included). This stability might suggest that the performance degradation observed in Table 5.1 is predominantly attributable to numerical features rather than categorical ones.

Importance of different external factors.

Table 5.3 demonstrates the significant contribution of each external factor to model performance. The experiment incorporating all external factors achieves the lowest error, while removing any category of factors leads to performance degradation. Temporal indicators emerge as the most critical component, with their removal causing the largest increase in error, followed by weather indicators and perhaps more interestingly football events. We also observe that including more external factors systematically decreases the standard error, thus making the performance less uncertain. Overall, these results quantitatively validate the hypernetwork's capacity to effectively integrate diverse external signals, capturing

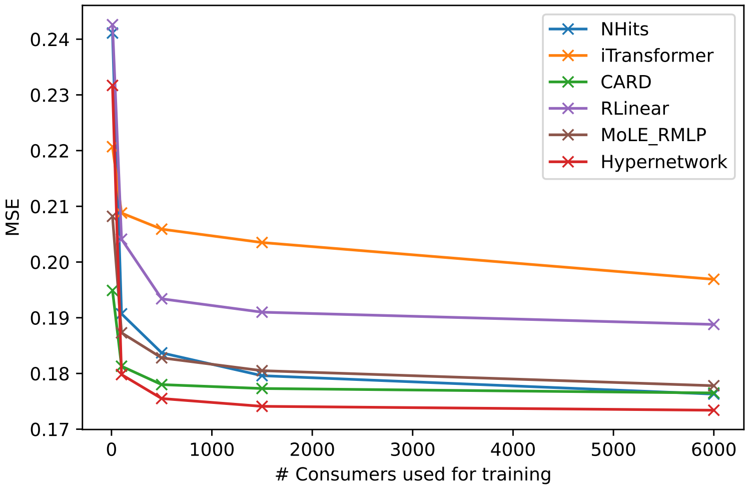


Figure 5.7: Performance evaluated on the full dataset when only a portion of the 6010 consumers is used to train models

Table 5.2: Performance (± standard error) with and without categorical features, for models that only handle numerical time series as inputs (asterisked in Table 5.1)

Model	With categorical features		Without categorical features	
	MSE	**MAE**	**MSE**	**MAE**
iTransformer	0.1969 ± 0.0013	**0.1966** ± 0.0012	**0.1966** ± 0.0014	0.1971 ± 0.0012
CARD	0.1765 ± 0.0000	0.1822 ± 0.0001	**0.1764** ± 0.0000	0.1822 ± 0.0001
NHits	**0.1763** ± 0.0002	**0.1854** ± 0.0005	0.1768 ± 0.0002	0.1861 ± 0.0005
PatchTST	0.1768 ± 0.0001	**0.1856** ± 0.0001	**0.1767** ± 0.0001	0.1857 ± 0.0001
RLinear	**0.1888** ± 0.0000	**0.2044** ± 0.0001	0.1890 ± 0.0000	0.2048 ± 0.0001

complex interdependencies between seemingly disparate factors and the target variable.

5.6 Discussion

Mixed role of external factors. The findings from our study challenge the prevailing assumption that integrating more external factors naturally enhances forecasting accuracy. Our results indicate that, for most models, the inclusion of additional external factors often leads to performance degradation. This suggests that the signal-to-noise ratio introduced by these external factors may not always be beneficial, highlighting the complexity involved in effectively leveraging such

Table 5.3: MSE and MAE values (± standard error) for the hypernetwork model with and without groups of external factors

Removed data	**MSE**	**MAE**
Weather indicators	0.1768 ± 0.0008	0.1865 ± 0.0013
Date & time indicators	0.1777 ± 0.0010	0.1875 ± 0.0018
Football events	0.1761 ± 0.0004	0.1850 ± 0.0009
∅	**0.1734** ± 0.0002	**0.1808** ± 0.0004

data.

Linearity of the forecasting process. While the end forecast is inherently linear and may not capture complex patterns directly [ZCZ^{+}23], the linear weights themselves are dynamically generated by the hypernetwork, which is nonlinear. This unique capability allows the linear model to adapt to more complex situations by tailoring weights to individual consumer behaviors, effectively making the final forecast nonlinear w.r.t. the input embeddings.

Adaptability. Hypernetworks present a notable exception by using external information without compromising performance, showcasing their capability in adapting to the varying significance of different input channels. New consumer embeddings can effectively be added over time to adapt to the demand evolution, which makes this solution suitable for real-world scenario. Encoders could be used in the future to be more effective than gradient descent in order to optimize these new embeddings.

Future work. Long time series embedding models [FBA23][WWD^{+}24] could be used to create consumers embeddings optimized to serve as hypernetwork's input. This would allow even faster profile embedding without having to apply gradient descent. More complex models than simple linear models could also be considered as for the hypernetwork's output. As already suggest with MoLE models, adding simple layers to the output model could potentially increase the performance.

5.7 Conclusion

In conclusion, our investigation into leveraging hypernetworks for electrical consumption forecasting reveals their potential as a robust alternative to traditional methods. By successfully exploiting external factors without degrading model performance for a reasonable cost, hypernetworks offer a promising direction for future research, especially in applications requiring the integration of diverse data channels. The results highlight the need for continued exploration into models that effectively balance complexity and accuracy, with improvements yet to be made to optimize new consumers embeddings, overall encouraging advancements in time series forecasting, especially with real-world applications.

6

PowerGraph-LLM: Novel Power Grid Graph Embedding and Optimization with Large Language Models

Electric load consumption forecasting can ultimately be used in power grid simulations. This chapter explores a side work by introducing the first framework designed to solve Optimal Power Flow (OPF) problems using Large Language Models, combining graph and tabular representations of power grids with specialized in-context learning and fine-tuning protocols. The framework addresses the growing need for scalable algorithms in modern power systems by leveraging LLMs to handle increasing variability and constraints while demonstrating reliable performance across different model architectures and sizes. This work has been made in close collaboration with Salah Ghamizi, then researcher at the LIST (Luxembourg Institute of Science and Technology).

Contents

6.1 Context 90
6.2 PowerGraph-LLM Framework 91
6.3 Empirical Study 93
6.4 Conclusion 96

6.1 Context

Resolving Alternating Current Optimal Power Flow (AC OPF) problems is a routine task in the operational planning of Power Systems (PS). Nevertheless, with the increasing variability, constraints, and uncertainties in today's power networks, solving problems accurately presents a significant challenge for power system engineers. Numerous strategies for addressing AC OPF challenges incorporate Machine Learning (ML) elements to mitigate the computational challenges associated with traditional optimization-based OPF approaches.

Graph Neural Networks (GNN) have recently demonstrated solid performance for various tasks in the power system. In particular, Liu et al.[LWZ23] proposed a new topology-informed GNN approach by combining grid topology and physical constraints. Ghamizi et al. [GMC+24] demonstrated that a heterogeneous graph representation combined with physical constraint losses leads to the best performances for PF and OPF tasks. Recent work such as SafePowerGraph[GBM+24] provided a standardized representation and benchmark of GNN for PF and OPF problems and identified the best architectures and design choice to solve these problems with GNN.

While these models achieve remarkable performance, they require expensive data curation — by collecting large training datasets with OPF solvers — and costly training for specific power grid sizes. Recent progress in foundation models, including LLMs (e.g. ChatGPT, LLaMa), has significantly reshaped the field of machine learning. Such models can indeed generalize to new tasks without expensive training, given the right indications. LLMs have also recently been explored to solve PS-related tasks. [YX23] proposed an LLM agent in the Deep Reinforcement Learning (DRL) training loop, directly allowing to model linguistic objectives and constraints in the OPF problem. The optimization is, however, run using traditional methods (solvers and DRL). A foundation model, developed in [HLL+24], can iteratively solve the OPF optimization problem by minimizing the cost function. The optimization only supports the toy example of a simple economic dispatch problem of units and does not fully optimize and predict the OPF variables (generation and bus variables) nor consider real world grid components (multiple loads, line operating limits). To the best of our knowledge, PowerGraph-LLM is the first embedding and optimization framework for OPF that supports realistic grid components and constraints.

This chapter presents three small contributions. We first propose novel power grid embedding for OPF to query LLMs with graph and tabular representations, followed by the development of tailored in-context learning and fine-tuning protocols for these models. We finally include an empirical study on how the architecture, size, and fine-tuning of LLMs affect their performance in solving OPF problems.

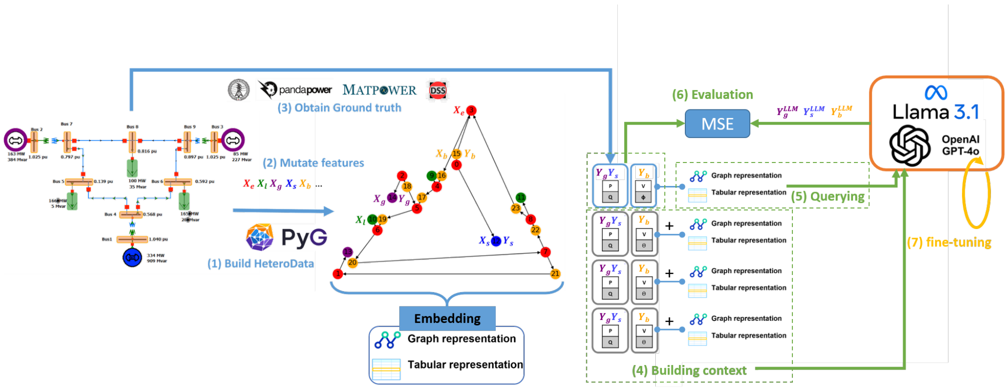


Figure 6.1: The PowerGraph-LLM framework: We generate the power grid embedding in steps 1, 2, 3 where we obtain a description of the topology of the grid, the features of its components (X_b, X_l, X_g, X_s, X_e respectively for bus, loads, generators, slack, and lines), and the OPF solution by a solver (Y_g, Y_s, Y_b). We generate thousands of power grid embeddings and we split them into a context and queries. The context (step 4) is the pairs (power grid description + OPF solutions) that will serve for the LLM model as examples, and the query (step 5) is the grid to which we expect the model to predict an OPF solution.

6.2 PowerGraph-LLM Framework

We present in Fig. 6.1 our PowerGraph-LLM framework. The first three steps consist in building the correct message format to query an LLM for OPF using appropriate table or graph embeddings. The following steps are either implemented using open-source LLMs (Llama), or remote API (OpenAI).

Power Grid Embedding (blue steps 1, 2, and 3 in Fig. 6.1) We extend SafePowerGraph [GBM+24] to build the appropriate embedding for LLM. Starting from an initial grid descriptor (in MatPower, PandaPower or OpenDSS formats), we generate a Pytorch HeteroData with each component as a distinct subgraph (Step (1)). Each node can be of type: bus, load, generator, slack, or line and is associated with its distinct set of features X_b, X_l, X_g, X_s, X_e respectively. In order to create new grids, these features are mutated in Step (2) depending on their types. For example, the mutation for loads of the active and reactive power is based on the real profile.

For each mutated grid, Step (3) runs a simulation and solver to derive the PF or OPF optimization's ground truth. This result includes the active and reactive power for every generator and the slack nodes (Y_g, Y_s), as well as the voltage magnitude and angle for each bus node (Y_b).

These features are then prepared for in-context inference in two formats. Our approach can generate LLM messages formatted as *graph representations*, including nodes features and edges connections. We can also generate LLM messages that only contain the features formatted in a tabular format, and we refer to this as the *tabular representation.* Given the context size of an LLM (i.e., how many tokens can be used as input), we can encode more examples in a table representation than in a graph representation.

LLM Inference (green steps 4, 5, and 6 in Fig. 6.1) LLMs predict the next token in a sequence by using contextual information from previously seen words to generate coherent text, facilitated by a chat interface allowing dialogue between a *user* and an *assistant*. The dialogue can be conditioned by a *system prompt*.

A prompt consists in two parts, the *context*, which consists in pairs of examples and expected answers and the *query*, which is the actual input we expect the LLM to predict. In Step (4), the context is pairs of inputs/outputs. The grids (in tabular or graph embedding from the previous step) are the input examples, and the OPF solutions $((Y_g, Y_s, Y_b))$ are the output examples. After providing a few examples, we query the LLM in step (5) with the actual grid to predict its OPF solution. The query only consists in the grid embedding either in tabular or graph representation without the OPF solutions.

The whole discussion consisting in the system prompt, the context and the query is processed in text format by the LLM. GPT and Llama models [eal24], two state-of-the-art LLMs families we consider in this chapter, share a common high-level architecture — consisting in breaking text into tokens, converting them to embeddings, and processing them through transformer blocks to capture data patterns [eal20].

LLM fine-tuning with LoRA The Low-Rank Adaptation (LoRA) [HSW+21] method allows fine-tuning LLMs efficiently by introducing low-rank matrices that capture task-specific adaptations while keeping the main model weights unchanged. Formally, instead of updating all the weights, LoRA modifies a small set by weight update ΔW as:

$$\Delta W = \frac{\alpha}{r} A \cdot B \tag{6.1}$$

where $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$, with $r \ll d, k$ and α being a chosen scaling factor. Only A and B are trained, reducing computation and memory needs, allowing for efficient adaptation to new tasks with few data and lower expenses while maintaining the LLMs' pre-existing capabilities.

We run the fine-tuning process in step (7) in yellow in Fig. 6.1. We consider each input example as a separate training sample and minimize the learning loss to its associated OPF solution.

Table 6.1: Errors in OPF estimation after fine-tuning GPT4o-mini and Llama-8b.

		Before fine-tuning			
Case	LLM	MSE_{GEN}	MSE_{SLACK}	MSE_{BUS}	VALID
9-bus	GPT4o-mini (graph)	0.205558	0.177675	0.000806	97.7%
	Llama-8b (graph)	8461773.4068	2213.2375	0.074695	11.5%
	Llama-8b (table)	3.172312	1.957585	0.001922	32.1%
30-bus	Llama-8b (graph)	-	-	-	0%

		After fine-tuning			
Case	LLM	MSE_{GEN}	MSE_{SLACK}	MSE_{BUS}	VALID
9-bus	GPT4o-mini (graph)	0.018740	0.010669	0.000269	93.1%
	Llama-8b (graph)	0.006417	0.061813	0.000629	99.7%
	Llama-8b (table)	0.054423	0.052320	0.001846	98.4%
30-bus	Llama-8b (graph)	0.01807	0.016839	0.000513	89.5%

6.3 Empirical Study

6.3.1 Experimental Protocol

In-context inference To investigate the capability of LLMs in generalizing from examples presented within the context window, we conduct an assessment using in-context inference. Four models were tested: OpenAI's *gpt-4o-mini*, OpenAI's *gpt-4o*, *Llama-3.1-8B-Instruct*, and *Llama-3.1-70B-Instruct* [eal24], the latter being respectively renamed *llama-8b* and *llama-70b* for the sake of brevity.

Sequences provided for in-context inference are made as follows: after an initial system prompt, a total of 65 pairs of example requests and solutions are provided using the JSON format. A context of 65 examples maximizes the utilization of context windows across all models. An additional, 66th example is used to evaluate the model's generalization abilities, with its response benchmarked against an expected solution. The overall sequence constructed is illustrated in Table 6.2.

If no JSON object can be read from the LLM's response, or if the values returned by the LLM are invalid (missing or invalid values), the output is deemed *INVALID*. This evaluation process was repeated 1,000 times, resulting in the generation of a total of 65,000 pairs for context and 1,000 pairs for evaluation across the assessments.

We run these inference tasks on one NVIDIA RTX 8000 48GB for Llama 8B (two for Llama 70B), using Q4_K_M quantization to optimize performance and resources.

Table 6.2: Sequence schema provided for LLM inference

system: You are a power grid operator running an Optimal Power Flow simulation, and you need to return a JSON-formatted response based on the provided input JSON. The input is the description of the components of the grid, including the buses, generators, loads, lines, and external grid. The output is the solution to the optimal power flow problem. You will get a few examples of Input and Output JSON. You need to return the correct Output for the last given Input.
user: Example Input JSON: <embedding input #1>
assistant: Example Output JSON: <OPF solution #1>
. . .
user: Query Input JSON: <embedding input #66>

Fine-tuning Subsequently, we apply fine-tuning to the *Llama-3.1-8B-Instruct* model, as shown on Figure 6.1, using the 65,000 pairs of context developed earlier. Each fine-tuning sample includes the system prompt, the specific OPF problem being addressed, and its corresponding solution. We follow a similar protocol to fine-tune OpenAI's *gpt-4o-mini* with API.

The fine-tuning process of the *Llama* model was conducted on an NVIDIA RTX 8000 48GB, using a LoRA configuration. The LoRA setup was defined with a rank of 8 ($r = 8$) and a scaling factor of 16 ($\alpha = 16$).

Evaluation In all experiments, we evaluate the mean squared error (MSE) for the active and reactive powers of the generators and the slack bus, as well as the voltage magnitude and phase angle of the buses. Each test grid is an IEEE 9-bus grid where the loads are mutated following a uniform distribution (+/- 20% variations). We also report the percentage of test grids where the output of the LLM was *invalid*, i.e. no JSON could be parsed from the output. This typically happens when the assistant tries to explain how to solve the problem instead of providing the solution itself. Reported values include MSE for generators (GEN), slack buses (SLACK), and buses (BUS) values in the grid.

6.3.2 Effectiveness of pre-trained LLMs

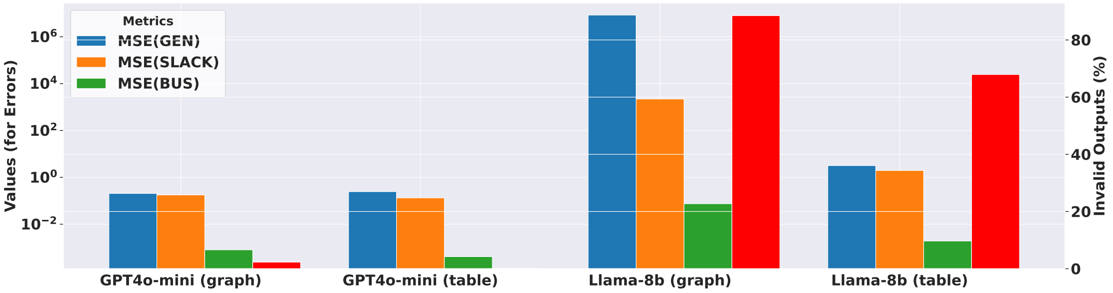


Figure 6.2: Errors in OPF estimations for *gpt-4o-mini* and *llama-8b*. The red bars represent the proportion of invalid inputs.

The results, presented in Figure 6.2, clearly show an important discrepancy between *gpt-4o-mini* and *llama-8b*. While *gpt-4o-mini* consistently showcases GEN and SLACK MSE values below 0.5 and BUS MSE below 10^{-3} for both graph and table formats with minimal invalid outputs, *llama-8b* exhibits higher MSEs by multiple magnitudes (min. 100 times more) for all criteria and a large majority of invalid outputs. The usage of the tabular representation here results in lower errors than graphs for both models, although it is less pronounced for the GEN and SLACK with *gpt-4o-mini*. Table representations also lead to less invalid outputs by 23%, supporting their efficiency.

6.3.3 Impact of the size of the model

As it is the case for natural language tasks [eal24], bigger models result in better performance (Figure 6.3). This improvement is especially prominent for *llama-70b*, getting closer to *gpt-4o-mini*'s performance in terms of GEN and SLACK MSE.

The MSE of BUS decreases when increasing the size of the LLM using table representation but increases significantly with graph representation. Our results confirm that larger LLMs lead to better performance for given representations, and both the size and the representation parameters should be considered together in future assessments.

6.3.4 Impact of fine-tuning

We report in Table 6.1 the impact of fine-tuning LLMs to solve the OPF problem. The percentage of invalid outputs of the LLM decreases to less than 2% for the Llama models and marginally increases for the GPT4o model. The error across all the components decreases, in particular for the Llama models. They become as effective as the proprietary model.

Contrary to the earlier vanilla models, graph representation is more effective than tabular to query LLMs after fine-tuning.

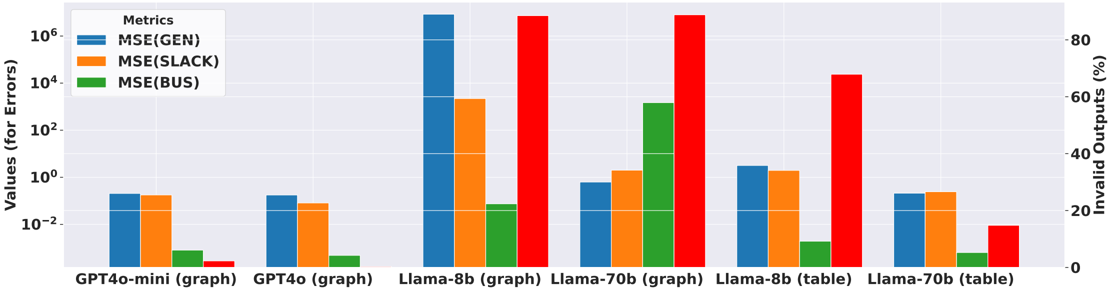


Figure 6.3: Errors in OPF estimations for graph representations using bigger *gpt-4o* and *llama* models. The red bars represent the proportion of invalid inputs.

In Fig. 6.4, we compare the predicted active power of the generator and the ground truth. After fine-tuning, the model achieves faithful results within large ranges of perturbations. Similarly, Fig. 6.5 shows that the fine-tuned LLM leads to closer results while achieving only rare violations (Bus 2).

We focus on the graph representation to evaluate the generalization of larger grids. Our results on the 30-bus grid in Table 6.1 demonstrate that off-shelf LLMs completely fail to generate valid OPF solutions, but that LLMs can be fine-tuned to achieve competitive results with graph embedding.

6.4 Conclusion

We introduced PowerGraph-LLM, a novel framework for solving OPF problems using LLMs. We propose a new power grid embedding combining graph and tabular representations, LLM fine-tuning protocols for OPF, and an empirical analysis of LLM performance. Our results show that larger models perform better, with fine-tuning significantly improving accuracy and reducing invalid outputs. Graph representations become more effective than tabular ones after fine-tuning. These findings highlight the potential of LLMs in power system optimization and open new avenues for more efficient and accurate OPF solutions for complex power grids.

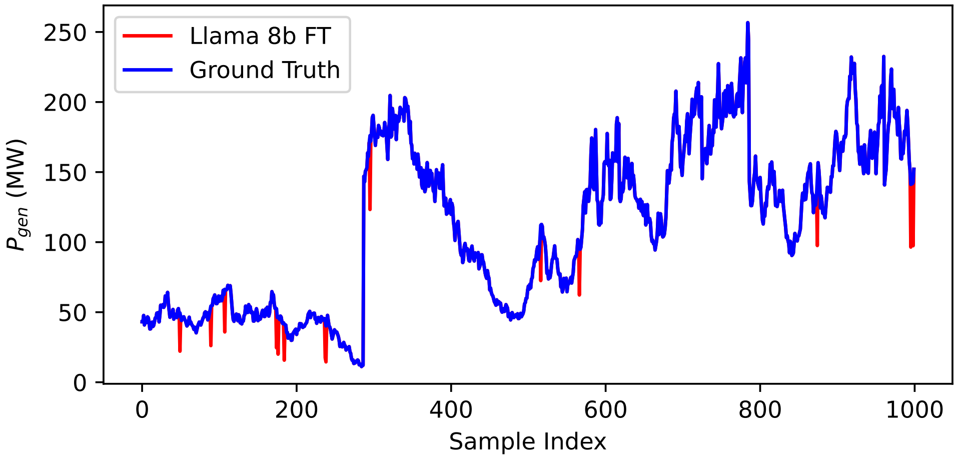


Figure 6.4: Predicted and real value (Gen 1) with Llama-8b.

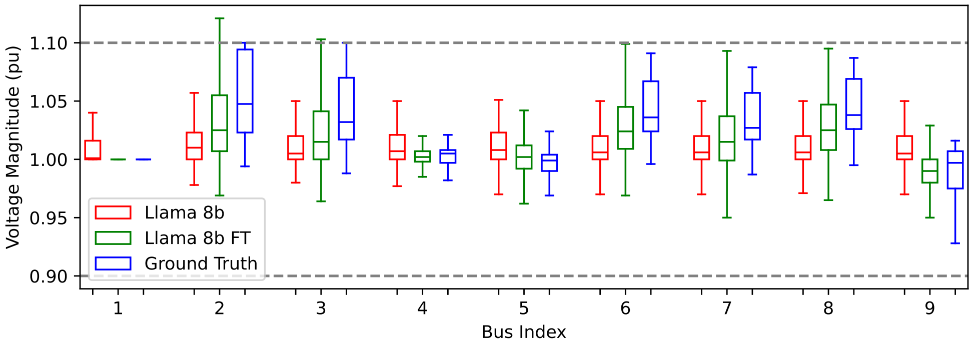


Figure 6.5: Predicted and real voltage of buses with Llama-8b.

7

Conclusion

This chapter proposes the overall conclusion of this dissertation and suggests potential future research direction.

Contents

7.1 Summary of Contributions 100
7.2 Limitations 101
7.3 Perspectives for Future Work 101
7.4 Concluding Remarks 102

7.1 Summary of Contributions

The objective of this thesis was to advance *energy–demand forecasting at scale* under the constraints of (i) ever-growing data volumes, (ii) rapidly changing usage patterns fostered by the energy transition, and (iii) narrow computational budgets typically available to distribution system operators (DSOs). To this end we have proposed, implemented and validated three complementary research strands:

1. **Transplit** – a light-weight, CPU–friendly transformer that leverages *season tokenisation* (Slice–to–Vector–to–Slice, SVS) to compress daily/weekly electrical load curves into a handful of vectors, reducing training time by up to $9.40 \times 10^2 \times$ while retaining state-of-the-art accuracy.
2. **DeDiPeak** – a diffusion based *deblurring* plug-in which rehabilitates the shy peaks produced by classical MSE-optimised predictors. Together with two novel peak-aware metrics ($P3_E$ and $P3_{SW}$) DeDiPeak improves peak forecasts by up to $3.6 \times 10^1\,\%$ without sacrificing base-line accuracy.
3. **Exogenous Factor Analysis** – an extensive ablation study over meteorological, socio-economic and event-driven signals which clarifies when and why external variables become beneficial and, conversely, when they unnecessarily inflate model complexity.

All contributions are released as reproducible open-source artefacts, and more specifically, lead to a coherent framework for time series forecasting.[1]

Across eight public benchmarks and two industrial data sets Transplit matches or surpasses deep transformers (Autoformer, FEDformer, PatchTST, *etc.*) while:

- dividing the sequence length inside the encoder/decoder by the period length T ($\to \approx 24-96$ fold shorter sequences);
- requiring $\leq 1.30 \times 10^2$ k – two orders of magnitude fewer than the closest competitors;
- training in minutes on a laptop CPU, hence enabling *daily* retraining cycles.

Standard losses (MAE, MSE) penalise temporal misalignment harsher than under-estimated magnitude. Consequently, most neural forecasters *flatten* critical peaks. DeDiPeak addresses this by:

introducing metrics that decouple amplitude and timing tolerance ($P3_E$, $P3_{SW}$);

applying inverse heat-dissipation diffusion to sharpen forecasts only where necessary.

The result is a systematic reduction of operational peak errors, a key lever for grid congestion management.

---

[1] `https://github.com/serval-uni-lu/transplit-framework`

While temperature and holidays reliably improve long-term error, fine-grained features (news sentiment, football matches, real-time prices) yield diminishing returns once seasonality is explicitly embedded by SVS. This emphasises the importance of feature cost/benefit analysis in frugal machine learning.

7.2 Limitations

Notwithstanding the encouraging results, several limitations remain:

- **Seasonality Assumption**: Transplit presumes a dominant, well-defined period T. Its performance degrades on weakly or non-seasonal series (e.g. exchange rates).
- **Static Filter Bank**: SVS uses shared convolutional filters learned *offline*. Adapting to concept drift currently requires full retraining.
- **Selective Deblurring**: DeDiPeak enhances *existing* peaks but struggles to resurrect completely missing ones when the base model fails to indicate them.
- **Computational Trade-off**: Although vastly cheaper than classical DDPMs, diffusion inference still incurs a fixed overhead (15 backward steps), which may be non-negligible for sub-second operational deadlines.

7.3 Perspectives for Future Work

Multi-Scale Season Tokenisation. Extending SVS towards a hierarchical dictionary (hour–day–week–year) would allow simultaneous exploitation of multiple seasonalities. Dynamic programming could stitch together variable-length slices, generalising beyond the fixed period T.

Probabilistic and Physics-aware Forecasting. Integrating quantile (or energy-based) objectives inside Transplit's decoder and coupling it with physical constraints (e.g. feeder capacity, conservation laws) would deliver forecasts that are not only accurate but *actionable* for grid operation.

Online and Federated Learning. Edge deployment at smart meters invites on-device fine-tuning under strict privacy budgets. Adapting Transplit to a federated setting – possibly leveraging split-SVS where only slice embeddings travel across the network – constitutes an exciting research avenue.

Adaptive Diffusion Schedules. Replacing the fixed 15-step deblurring with an *adaptive* scheduler conditioned on peak uncertainty could cut inference time by another order of magnitude while preserving sharpness.

Causal Exogenous Variable Selection. We have shown that not all external factors are beneficial. Coupling Granger-causality or attention-based attribution with automated feature pruning could yield a lean yet explanatory pipeline.

7.4 Concluding Remarks

The accelerating electrification of mobility, heating and industry dramatically amplifies peak loads and volatility on distribution grids. Accurate, *scalable* and *frugal* forecasting therefore becomes a cornerstone of the sustainable energy transition.

This thesis demonstrates that **efficiency and accuracy are not mutually exclusive.**

By re-thinking time-series representation (SVS), error metrics (P3 family) and post-processing (deblurring diffusion), we reconcile deep learning with the stringent reality of grid-edge devices and CPU-only data centres. We hope that the ideas and open-source artefacts presented here will help practitioners deploy load-aware services faster, researchers design ever more incisive models, and, ultimately, societies manage their energy resources more wisely.

8

Appendices

Appendices and side explorations

Contents

8.1 Time series token prediction with adaptive neural decision trees 104

8.1 Time series token prediction with adaptive neural decision trees

We have seen in the introduction of chapter 4 that the time series forecast regression task can be turned in a *classification* task, by converting a time series to tokens using a VQ-VAE. An interesting observation from Figure 4.1 is that some time series have very easily predictable patterns (bottom), while others are more chaotic, seem more random and need deeper analysis to forecast the future consumption. However, while a human brain will take more time in processing more complex tasks, deep learning models take the same time in processing both, despite the huge difficulty discrepancy.

We ran a small experiment stacking simple layers iteratively to build a decision tree: the idea is that some branches become expert of tasks, and obvious predictions can be classified early. The idea of having diffentiable decision trees is not new [Bal17]. Before moving to a specific branch, each layer comes along with a classification head (dense layer) which ultimately allows us to know in which branch to move for the current sample. The overall process is better described visually along with the captions of figures 8.1, 8.2, 8.3 and 8.4.

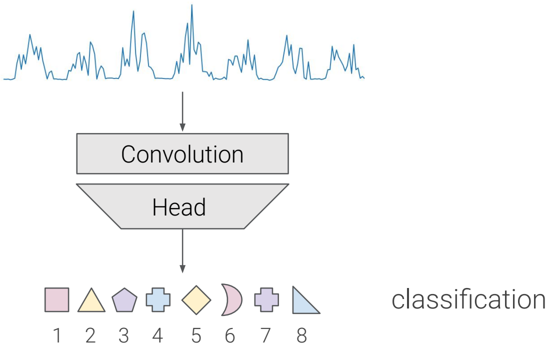


Figure 8.1: Initial stage: a 1D-convolutional layer trains for the classification among all possible tokens.

This however resulted in even longer predictions. The batch management was especially challenging as every sample of the batch goes to different individual layers / branches, which ultimately erases the essential point of working with batches, as this splits all tensor operations. The usage of the classification head at every layer also adds significant computational overhead. In total, 10 seconds were required to do the forecast of 32 samples, which is nearly 50,000 times longer than Transplit. Outcome: better work with a fast, time-constant neural network than a slow, variable decision tree.

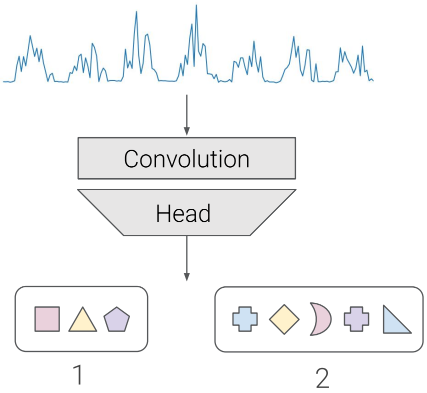


Figure 8.2: Splitting stage: the tokens are put in two groups so that the accuracy is maximal when the task is reduced to the classification among those two groups only.

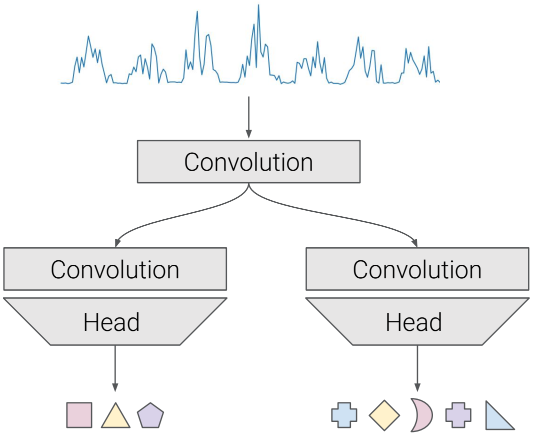


Figure 8.3: Repeat the process with two subsequent branches. Each branch now focuses on the classification task over the tokens from their given group. Note: if the group consists in only one token, the branch can stop here without training any new layer.

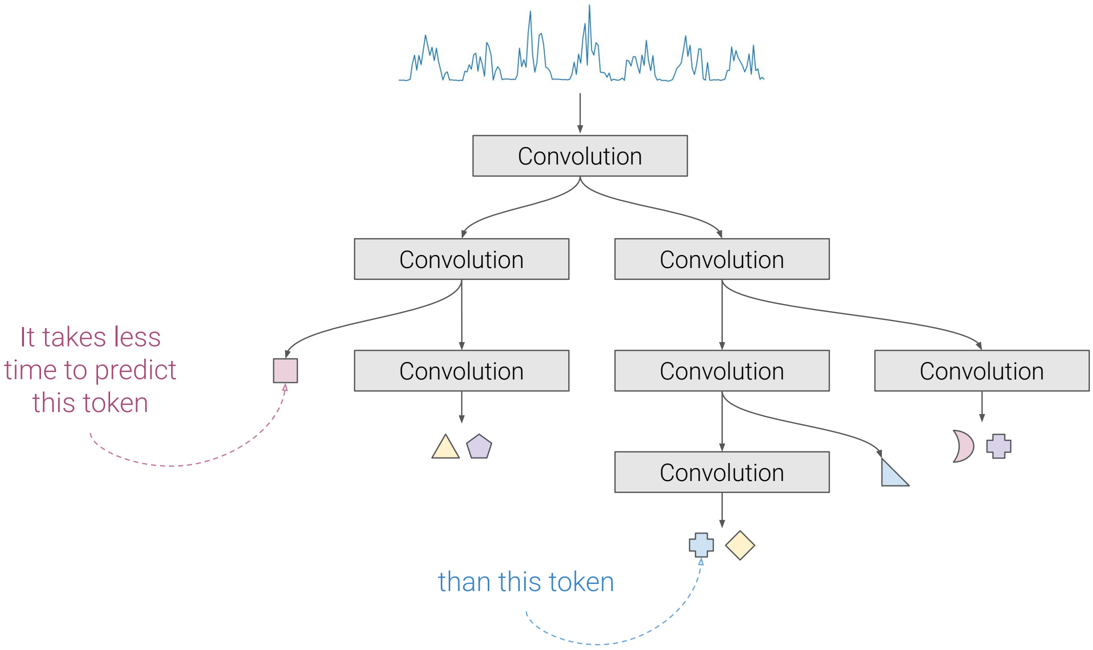


Figure 8.4: When all tokens have been isolated in one branch or if we reach the maximum depth, the training phase is over. All classification heads are still used for inference, to decide the branch to move in.

AI Assistance Disclosure

I hereby acknowledge that parts of the text in this thesis were refined with the assistance AI language models. The tools were used solely for language polishing and rewriting purposes; all ideas, analysis, and conclusions are my own.

List of publications and tools

Papers included in the dissertation

- Fabien Bernier et al. Transplit: Faster and Cheaper Energy Demand Forecasting at Scale. *Under review.*
- Fabien Bernier et al. Peak Forecasting Enhancement with Deblurring Diffusion Models. *Under review.*
- Fabien Bernier et al. Leveraging External Factors in Household-Level Electrical Consumption Forecasting using Hypernetworks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases, ECML-PKDD 2025, Applied Data Science Track*, 2025
- Fabien Bernier et al. SafePowerGraph-LLM: Novel Power Grid Graph Embedding and Optimization with Large Language Models. In *IEEE Power Engineering Letters*, 2025

Papers not included in the dissertation

- Pantelis Dogoulis et al. Constraint-Guided Prediction Refinement via Deterministic Diffusion Trajectories. *Under review.*

Tools included in the dissertation

- Overall forecasting framework with demo: `https://github.com/serval-uni-lu/transplit-framework`

List of figures

1.1 Important localized overconsumption can lead to network overloads, which threaten the stability of the power grid. 2

3.1 Simple, yet efficient architecture of Transplit: (1) the input time series is split into seasons; (2) Slice2Vec converts each cycle into a vector; (3) the transformer forecasts vectors (4) which are converted back to a time series with Vec2Slice. 14
3.2 The Slice2Vec process 16
3.3 The Vec2Slice process 16
3.4 Predictions of Transplit (orange) and Reformer (green), showing that both models are able to recognize various profiles at different scales, while being trained only once. 24
3.5 Test loss variation when varying the seasonality parameter on the CREOS and Traffic datasets. The Fourier decomposition is shown below as reference (period on the x-axis). 28
3.6 Transplit performance on the *electricity* dataset, w.r.t. sequence length provided in input 29
3.7 Transplit performance on the *traffic* dataset, w.r.t. sequence length provided in input 29
3.8 Average time taken by Transplit to forecast 1 week from different input sequence lengths. 30
3.9 The model is trained on the electricity dataset, with a loss being a mix of a range of trade-offs between MSE and PMSE. $t = 0$ means that Transplit is only trained on the MSE loss, $t = 1$ means that the loss is fully defined by the PMSE. The variation of MSE, MAE, PMSE and PMAE is reported. 30

4.1 Looks like Transplit fits well with different consumption patterns. That’s great. But what about peak amplitudes (highlighted in red)? Even the obvious ones (bottom)? That’s exactly what this chapter is about. 32

4.2 Example of a time series y and two different approximations $\hat{y}_1$ and $\hat{y}_2$, with an MSE of 0.62 and 0.26, respectively. 34
4.3 A VQ-VAE model converts time series into discrete tokens, enabling prediction in the token space rather than directly on time series values, potentially avoiding point-wise metric limitations. 37
4.4 MinGPT architecture for time series forecasting: a GPT-like model trained to predict subsequent tokens in tokenized consumption sequences. 38
4.5 Example of generation with a DDPM with difference guidance factors, from the same initial noise. 40
4.6 Illustrations of DDPM (denoising) and IHDM (deblurring, or *desmoothing*) and their effects on time series. 44
4.7 Illustration of $P3_{sw}$ and $P3_{\mathcal{E}}$ metrics, respectively. 48
4.8 Visualization of peaks within a consumption time series. The dashed grey line corresponds to the average amplitude. 48
4.9 Figure illustrating our new framework DeDiPeak. 49
4.10 Example of improved peaks by the diffusion model taken from the *traffic* dataset. 52
4.11 Dependence of the IHDM predictive performance (MSE in blue, $P3_{sw}$ in orange, $P3_{\mathcal{E}}$ in violet) applied on top of the FiLM base model for the ETTm2 dataset on the variance of the sampling noise δ^2 and of the training noise σ^2. 54
4.12 Comparison between a stand-alone deblurring model and DeDiPeak 55
4.13 Definition of a residual block in the model. Each residual block in the U-Net is conditioned both on the deblurring step t, and the input time series c. 59
4.14 FEDformer on the *Traffic* dataset 60
4.15 FiLM on the *Weather* dataset 61
4.16 NHits on the *Electricity* dataset 61
4.17 Performer on the *ETTm1* dataset 62
4.18 Dependence of the IHDM's predictive performance (MSE in blue, $P3_{sw}$ in orange, $P3_{\mathcal{E}}$ in violet) applied on top of the FiLM and FEDformer base models for the *weather* and *ETTm1* datasets on the variance of the sampling noise δ^2 and of the training noise σ^2. 63
4.19 Impact of different values of sampling noise. Deblurring from the FiLM model on ETTm2. 63
4.20 $P3_{\mathcal{E}}$ evolution when varying the ratio α/β 65

5.1 Difference between global and individual models, and the proposed in-between solution using hypernetworks. 73

5.2 Weighting the different external factors with the consumer embedding in the custom attention mechanism. Each consumer embedding $\mathbf{c}$ is used to compute attention weights α_i for external factors through scalar products, creating a personalized representation of environmental influences. 75
5.3 Overview of the whole forecasting pipeline, with the initial idea of adapting Transplit to handle external factors w.r.t. a consumer embedding. 76
5.4 Overview of the hypernetwork pipeline 78
5.5 Illustration of example embeddings. Each consumer ID and other known categorical features are transformed to embeddings, which are reshaped and stacked together to form the Hypernetwork input. 79
5.6 Comparison of models training time vs. the best resulting MSE. Bubble sizes refer to the number of weights. For our approach, we distinguish the size of the hypernetwork itself, and the size including all 6010 consumers embeddings. The size for individual RLinear models (bottom right) is not shown as it would fill in all the figure. 85
5.7 Performance evaluated on the full dataset when only a portion of the 6010 consumers is used to train models 86

6.1 The PowerGraph-LLM framework: We generate the power grid embedding in steps 1, 2, 3 where we obtain a description of the topology of the grid, the features of its components (X_b, X_l, X_g, X_s, X_e respectively for bus, loads, generators, slack, and lines), and the OPF solution by a solver (Y_g, Y_s, Y_b). We generate thousands of power grid embeddings and we split them into a context and queries. The context (step 4) is the pairs (power grid description + OPF solutions) that will serve for the LLM model as examples, and the query (step 5) is the grid to which we expect the model to predict an OPF solution. 91
6.2 Errors in OPF estimations for *gpt-4o-mini* and *llama-8b*. The red bars represent the proportion of invalid inputs. 95
6.3 Errors in OPF estimations for graph representations using bigger *gpt-4o* and *llama* models. The red bars represent the proportion of invalid inputs. 96
6.4 Predicted and real value (Gen 1) with Llama-8b. 97
6.5 Predicted and real voltage of buses with Llama-8b. 97

8.1 Initial stage: a 1D-convolutional layer trains for the classification among all possible tokens. 104

8.2 Splitting stage: the tokens are put in two groups so that the accuracy is maximal when the task is reduced to the classification among those two groups only. 105
8.3 Repeat the process with two subsequent branches. Each branch now focuses on the classification task over the tokens from their given group. Note: if the group consists in only one token, the branch can stop here without training any new layer. 105
8.4 When all tokens have been isolated in one branch or if we reach the maximum depth, the training phase is over. All classification heads are still used for inference, to decide the branch to move in. 106

List of tables

3.1 Number of trainable parameters and **GPU** training time for each model and dataset used, for a 720 → 720 values forecast. 21
3.2 **CPU** training time for each model and dataset used, for a 720 → 720 values forecast. 21
3.3 Performances on seasonal datasets for 1 week forecasting 22
3.4 Case study: Error metrics for the electricity and CREOS datasets (kWh scale) 23

4.1 Comparison of the MSE and $P3$ metrics for several models on five datasets. Each pair of columns compares the score of the baseline model (on the left) and its update after deblurring (on the right). The $P3$ results are put in bold when improved by the deblurring process. 57
4.2 Comparison between DeDiPeak and EVL – EVT model, input length 336, forecast length 168. 58
4.3 Comparison between DeDiPeak and DILATE – Seq2Seq model, input length 168, forecast length 24. 58
4.4 *Traffic* dataset: average variation of the $P3_{sw}$ in %, after applying the diffusion model on FiLM predictions. $P3_{sw}$ is expressed with different values of T and n. 66
4.5 Computation time comparison on the weather dataset, performed on an NVIDIA RTX A2000 Laptop GPU 66
4.6 Comparison of the MSE and $P3$ metrics between the D3VAE model and Dedipeak/NHits on five datasets. 67
4.7 Proportion of peaks successfully predicted by different baseline models within 5 neighbors, regardless of their amplitude. 68

5.1 MSE and MAE values for different models and datasets. Models denoted with an asterisk * are not meant to handle categorical features: the consumer's ID embedding is provided in additional time series channels 83

5.2 Performance (± standard error) with and without categorical features, for models that only handle numerical time series as inputs (asterisked in Table 5.1) 86
5.3 MSE and MAE values (± standard error) for the hypernetwork model with and without groups of external factors 87

6.1 Errors in OPF estimation after fine-tuning GPT4o-mini and Llama-8b. 93
6.2 Sequence schema provided for LLM inference 94

Bibliography

[ABC^{+}05] Torben G Andersen, Tim Bollerslev, Peter Christoffersen, and Francis X Diebold. Volatility forecasting, 2005 (cited on page 3).

[AS22] Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based time series imputation and forecasting with structured state space models. *arXiv preprint arXiv:2208.09399*, 2022 (cited on page 48).

[Bal17] Randall Balestriero. Neural decision trees, 2017. arXiv: `1702.07360 [stat.ML]`. URL: `https://arxiv.org/abs/1702.07360` (cited on page 104).

[Bar19] Jonathan T Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4331–4339, 2019 (cited on pages 33, 35, 46).

[BJ76] George EP Box and Gwilym M Jenkins. Time series analysis: forecasting and control san francisco. *Calif: Holden-Day*, 1976 (cited on page 8).

[BMR^{+}20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020 (cited on page 38).

[Che23] Ting Chen. On the importance of noise scheduling for diffusion models. *arXiv preprint arXiv:2301.10972*, 2023 (cited on page 43).

[CLD^{+}20] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020 (cited on page 50).

[COO$^+$22] Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza, Max Mergenthaler-Canseco, and Artur Dubrawski. N-hits: neural hierarchical interpolation for time series forecasting. *arXiv preprint arXiv:2201.12886*, 2022 (cited on pages 50, 82).

[CRC$^+$20] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020 (cited on page 36).

[CXC$^+$17] Yongbao Chen, Peng Xu, Yiyi Chu, Weilin Li, Yuntao Wu, Lizhou Ni, Yi Bao, and Kun Wang. Short-term electrical load forecasting using the support vector regression (svr) model to calculate the demand response baseline for office buildings. *Applied Energy*, 195:659–670, 2017. ISSN: 0306-2619. DOI: `https://doi.org/10.1016/j.apenergy.2017.03.034`. URL: `https://www.sciencedirect.com/science/article/pii/S0306261917302581` (cited on page 72).

[Dat14] ECL Dataset. ECL Dataset. `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`, 2014. [Online; accessed 17-May-2023] (cited on pages 4, 50).

[DB16] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *Advances in neural information processing systems*, 29, 2016 (cited on page 33).

[DHZ$^+$23] Wenying Duan, Xiaoxi He, Lu Zhou, Lothar Thiele, and Hong Rao. Combating distribution shift for accurate time series forecasting via hypernetworks. In *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 900–907. IEEE, 2023 (cited on page 77).

[DMD$^+$21] Shuang Dai, Fanlin Meng, Hongsheng Dai, Qian Wang, and Xizhong Chen. Electrical peak demand forecasting: A review. Technical report, August 2021. DOI: `https://doi.org/10.48550/arXiv.2108.01393` (cited on pages 3, 33, 39).

[DN21a] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021 (cited on page 36).

[DN21b] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021 (cited on page 41).

[DZ14] Giuseppe Da Prato and Jerzy Zabczyk. *Stochastic equations in infinite dimensions*. Cambridge university press, 2014 (cited on page 42).

[DZP+19] Daizong Ding, Mi Zhang, Xudong Pan, Min Yang, and Xiangnan He. Modeling extreme events in time series prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1114–1122, 2019 (cited on pages 41, 50, 53).

[eal20] Tom B. Brown et al. Language models are few-shot learners, 2020. arXiv: `2005.14165 [cs.CL]`. URL: `https://arxiv.org/abs/2005.14165` (cited on page 92).

[eal24] Aaron Grattafiori et al. The llama 3 herd of models, 2024. arXiv: `2407.21783 [cs.AI]`. URL: `https://arxiv.org/abs/2407.21783` (cited on pages 92, 93, 95).

[ERO21] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021 (cited on pages 33, 36).

[EVH+21] Mikhail Evchenko, Joaquin Vanschoren, Holger H. Hoos, Marc Schoenauer, and Michèle Sebag. Frugal machine learning, 2021. arXiv: `2111.03731 [cs.LG]`. URL: `https://arxiv.org/abs/2111.03731` (cited on pages 3, 12).

[FBA23] Archibald Fraikin, Adrien Bennetot, and Stéphanie Allassonnière. T-rep: representation learning for time series using time-embeddings. *arXiv preprint arXiv:2310.04486*, 2023 (cited on page 87).

[FSE+22] Elizabeth Fons, Alejandro Sztrajman, Yousef El-Laham, Alexandros Iosifidis, and Svitlana Vyetrenko. Hypertime: implicit neural representation for time series. *arXiv preprint arXiv:2208.05836*, 2022 (cited on page 77).

[Gar85] Everette S Gardner Jr. Exponential smoothing: the state of the art. *Journal of forecasting*, 4(1):1–28, 1985 (cited on page 8).

[GBM+24] Salah Ghamizi, Aleksandar Bojchevski, Aoxiang Ma, and Jun Cao. Safepowergraph: safety-aware evaluation of graph neural networks for transmission power grids. *arXiv preprint arXiv:2407.12421*, 2024 (cited on pages 90, 91).

[GHC+19] Alfonso González-Briones, Guillermo Hernández, Juan M. Corchado, Sigeru Omatu, and Mohd Saberi Mohamad. Machine learning models for electricity consumption forecasting: a review. In *ICCAIS'19*, 2019. DOI: `10.1109/CAIS.2019.8769508` (cited on page 13).

[GLA22] Alberto Gasparin, Slobodan Lukovic, and Cesare Alippi. Deep learning for time series forecasting: the electric load case. *CAAI Transactions on Intelligence Technology*, 7(1):1–25, 2022 (cited on page 81).

[GMC+24] Salah Ghamizi, Aoxiang Ma, Jun Cao, and Pedro Rodriguez Cortes. Opf-hgnn: generalizable heterogeneous graph neural networks for ac optimal power flow. In *2024 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5, 2024. DOI: `10.1109/PESGM51994.2024.10688560` (cited on page 90).

[GPM+14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014 (cited on page 35).

[HDL16] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016 (cited on pages 73, 77).

[HF16] Tao Hong and Shu Fan. Probabilistic electric load forecasting: a tutorial review. *International Journal of Forecasting*, 32(3):914–938, 2016. ISSN: 0169-2070. DOI: `https://doi.org/10.1016/j.ijforecast.2015.11.011` (cited on pages 2, 72).

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020 (cited on pages 36, 39, 46).

[HLL+24] Chenghao Huang, Siyang Li, Ruohong Liu, Hao Wang, and Yize Chen. Large foundation models for power systems. In *2024 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2024 (cited on page 90).

[HS22a] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022 (cited on pages 39, 51).

[HS22b] Emiel Hoogeboom and Tim Salimans. Blurring diffusion models. *arXiv preprint arXiv:2209.05557*, 2022 (cited on pages 4, 34, 46).

[HSW+21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: low-rank adaptation of large language models, 2021. arXiv: `2106.09685 [cs.CL]`. URL: `https://arxiv.org/abs/2106.09685` (cited on page 92).

[JAF16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016 (cited on page 35).

[JJN$^+$91] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991 (cited on page 77).

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *ICLR'15*, 2015 (cited on pages 20, 50).

[KGE$^+$19] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: learning a vector representation of time, 2019. DOI: `10.48550/ARXIV.1907.05321`. URL: `https://arxiv.org/abs/1907.05321` (cited on page 13).

[KKL20] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: the efficient transformer, 2020. DOI: `10.48550/ARXIV.2001.04451`. URL: `https://arxiv.org/abs/2001.04451` (cited on pages 8, 13, 18, 24, 50).

[KSP$^+$21] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021 (cited on page 46).

[LB19] Jinxiang Liu and Laura E Brown. Prediction of hour of coincident daily peak load. In *2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5. IEEE, 2019 (cited on page 39).

[LCY$^+$18] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *SIGIR '18*, 2018. ISBN: 9781450356572. DOI: `10.1145/3209978.3210006` (cited on pages 8, 19, 33).

[LHZ$^+$24] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. Itransformer: inverted transformers are effective for time series forecasting, 2024. arXiv: `2310.06625 [cs.LG]`. URL: `https://arxiv.org/abs/2310.06625` (cited on pages 72, 77, 82).

[LJX$^+$19] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *NIPS'19*, volume 32, 2019 (cited on page 13).

[LLW$^+$22] Yan Li, Xinjiang Lu, Yaqing Wang, and Dejing Dou. Generative time series forecasting with diffusion, denoise, and disentanglement. *Advances in Neural Information Processing Systems*, 35:23009–23022, 2022 (cited on pages 41, 43, 48, 67).

[LPP20] Ioannis E Livieris, Emmanuel Pintelas, and Panagiotis Pintelas. A cnn–lstm model for gold price time-series forecasting. *Neural computing and applications*, 32:17351–17360, 2020 (cited on page 33).

[LQL+23] Zhe Li, Shiyi Qi, Yiduo Li, and Zenglin Xu. Revisiting long-term time series forecasting: an investigation on linear mapping, 2023. arXiv: `2305.10721 [cs.LG]`. URL: `https://arxiv.org/abs/2305.10721` (cited on page 82).

[LT19] Vincent Le Guen and Nicolas Thome. Shape and time distortion loss for training deep time series forecasting models. *Advances in neural information processing systems*, 32, 2019 (cited on pages 41, 50, 53).

[LWZ23] Shaohui Liu, Chengyang Wu, and Hao Zhu. Topology-Aware Graph Neural Networks for Learning Feasible and Adaptive AC-OPF Solutions. en. *IEEE Transactions on Power Systems*, 38(6):5660–5670, November 2023. ISSN: 0885-8950, 1558-0679. DOI: `10.1109/TPWRS.2022.3230555`. URL: `https://ieeexplore.ieee.org/document/9992121/` (visited on 10/27/2023) (cited on page 90).

[MLK21] Alex Mallen, Henning Lange, and J. Nathan Kutz. Deep probabilistic koopman: long-term time-series forecasting under periodic uncertainties, 2021. DOI: `10.48550/ARXIV.2106.06033`. URL: `https://arxiv.org/abs/2106.06033` (cited on page 13).

[MM18] Rishabh Madan and Partha Sarathi Mangipudi. Predicting computer network traffic: a time series forecasting approach using dwt, arima and rnn. In *2018 Eleventh International Conference on Contemporary Computing (IC3)*, pages 1–5, 2018. DOI: `10.1109/IC3.2018.8530608` (cited on page 33).

[MS22] Sidra Mehtab and Jaydip Sen. Analysis and forecasting of financial time series using cnn and lstm-based deep learning models. In *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2021*, pages 405–423. Springer, 2022 (cited on pages 2, 33).

[NBS19] Meike Nauta, Doina Bucur, and Christin Seifert. Causal discovery with attention-based convolutional neural networks. *Machine Learning and Knowledge Extraction*, 1(1):312–340, 2019. ISSN: 2504-4990. DOI: `10.3390/make1010019`. URL: `https://www.mdpi.com/2504-4990/1/1/19` (cited on page 49).

[NLW$^+$24] Ronghao Ni, Zinan Lin, Shuaiqi Wang, and Giulia Fanti. Mixture-of-linear-experts for long-term time series forecasting. In *International Conference on Artificial Intelligence and Statistics*, pages 4672–4680. PMLR, 2024 (cited on pages 77, 81).

[NNS$^+$23] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: long-term forecasting with transformers, 2023. arXiv: `2211.14730 [cs.LG]` (cited on pages 8, 15, 77, 82).

[NTN$^+$20] Isaac Kofi Nti, Moses Teimeh, Owusu Nyarko-Boateng, and Adebayo Felix Adekoya. Electricity load forecasting: a systematic review. *Journal of Electrical Systems and Information Technology*, 7(1), 2020 (cited on page 2).

[PG21] Fotios Petropoulos and Yael Grushka-Cockayne. Fast and frugal time series forecasting, February 2021. DOI: `10.48550/arXiv.2102.13209` (cited on pages 3, 12).

[PKA$^+$21] A. Parrado-Duque, S. Kelouwani, K. Agbossou, S. Hosseini, N. Henao, and F. Amara. A comparative analysis of machine learning methods for short-term load forecasting systems. In *SmartGridComm'21*, 2021. DOI: `10.1109/SmartGridComm51999.2021.9632002` (cited on page 13).

[PKD$^+$16] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016 (cited on page 35).

[RBL$^+$22a] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. arXiv: `2112.10752 [cs.CV]` (cited on pages 39, 41).

[RBL$^+$22b] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022 (cited on page 36).

[RHS22] Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation. *arXiv preprint arXiv:2206.13397*, 2022 (cited on pages 4, 34, 42, 43, 53).

[RPG+21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021 (cited on pages 36, 38).

[RSS+21] Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, pages 8857–8868. PMLR, 2021 (cited on pages 2, 33, 41, 43, 48).

[RVV19] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019 (cited on page 36).

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019 (cited on page 38).

[SAM+22] Amin Shabani, Amir Abdi, Lili Meng, and Tristan Sylvain. Scaleformer: iterative multi-scale refining transformers for time series forecasting. *arXiv preprint arXiv:2206.04038*, 2022 (cited on pages 18, 33, 41, 46, 50).

[SCS+22] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022 (cited on page 39).

[SFF22] Md Jamal Ahmed Shohan, Md Omar Faruque, and Simon Y. Foo. Forecasting of electric load using a hybrid lstm-neural prophet model. *Energies*, 15(6), 2022. ISSN: 1996-1073. DOI: `10.3390/en15062158` (cited on page 13).

[SO21] Marco Savi and Fabrizio Olivadese. Short-term energy consumption forecasting at the edge: a federated learning approach. *IEEE Access*, 9:95949–95969, 2021. DOI: `10.1109/ACCESS.2021.3094089` (cited on page 72).

[SSK+20] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020 (cited on pages 36, 39, 45).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014 (cited on page 51).

[SWM$^+$15] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015 (cited on page 41).

[THP$^+$21] Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Nikolay Laptev, Christoph Bergmeir, and Ram Rajagopal. NeuralProphet: Explainable Forecasting at Scale, 2021. DOI: `10.48550/arXiv.2111.15397` (cited on page 13).

[tra20] traffic. traffic. `https://pems.dot.ca.gov/`, 2020. [Online; accessed 17-May-2023] (cited on page 50).

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS'17*, 2017 (cited on pages 2, 8, 14, 18, 33, 36).

[VV$^+$17a] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017 (cited on pages 17, 78).

[VV$^+$17b] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017 (cited on pages 36, 37).

[WCH$^+$19] Yi Wang, Qixin Chen, Tao Hong, and Chongqing Kang. Review of smart meter data analytics: applications, methodologies, and challenges. *IEEE Transactions on Smart Grid*, 10(3):3125–3148, May 2019. ISSN: 1949-3061. DOI: `10.1109/tsg.2018.2818167`. URL: `http://dx.doi.org/10.1109/TSG.2018.2818167` (cited on page 72).

[WCK$^+$16] Yi Wang, Qixin Chen, Chongqing Kang, and Qing Xia. Clustering of electricity consumption behavior dynamics toward big data applications. *IEEE transactions on smart grid*, 7(5):2437–2447, 2016 (cited on page 72).

[wet20] wetter. wetter. `https://www.bgc-jena.mpg.de/wetter/`, 2020. [Online; accessed 17-May-2023] (cited on page 50).

[whXW$^+$21] haixu wu haixu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: decomposition transformers with auto-correlation for long-term series forecasting. In *NIPS'21*, 2021 (cited on pages 2, 8, 13, 18, 20, 33, 50, 81).

[WWD$^+$24] Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Guo Qin, Haoran Zhang, Yong Liu, Yunzhong Qiu, Jianmin Wang, and Mingsheng Long. Timexer: empowering transformers for time series forecasting with exogenous variables. *arXiv preprint arXiv:2402.19072*, 2024 (cited on page 87).

[WZZ$^+$22] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: a survey, 2022. DOI: `10.48550/ARXIV.2202.07125`. URL: `https://arxiv.org/abs/2202.07125` (cited on page 18).

[XXW12] Jiliang Xue, Zhenyuan Xu, and Junzo Watada. Building an integrated hybrid model for short-term and mid-term load forecasting with genetic optimization. *International Journal of Innovative Computing, Information and Control*, 8(10):7381–7391, 2012 (cited on pages 39, 41).

[XZW$^+$24] Wang Xue, Tian Zhou, Qingsong Wen, Jinyang Gao, Bolin Ding, and Rong Jin. Card: channel aligned robust blend transformer for time series forecasting, 2024. arXiv: `2305.12095 [cs.LG]`. URL: `https://arxiv.org/abs/2305.12095` (cited on pages 72, 77, 82).

[YSL$^+$19] Bailin Yang, Shulin Sun, Jianyuan Li, Xianxuan Lin, and Yan Tian. Traffic flow prediction using lstm with feature enhancement. *Neurocomputing*, 332, 2019. ISSN: 0925-2312 (cited on pages 2, 8, 33).

[YX23] Ziming Yan and Yan Xu. Real-time optimal power flow with linguistic stipulations: integrating gpt-agent and deep reinforcement learning. *IEEE Transactions on Power Systems*, 2023 (cited on page 90).

[ZCZ$^+$23] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37 of number 9, pages 11121–11128, 2023 (cited on page 87).

[ZHB$^+$24] Kawthar Zaraket, Hassan Harb, Ismail Bennis, Ali Jaber, and Abedalhafid Abouaissa. Hyper-flophet: a neural prophet-based model for traffic flow forecasting in transportation systems. *Simulation Modelling Practice and Theory*, 134:102954, 2024. ISSN: 1569-190X. DOI: `https://doi.org/10.1016/j.simpat.2024.102954`.

URL: `https://www.sciencedirect.com/science/article/pii/S1569190X24000686` (cited on page 13).

[ZM98] Jun Zhang and K.F. Man. Time series prediction using rnn in multi-dimension embedding phase space. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, volume 2, 1868–1873 vol.2, 1998. DOI: `10.1109/ICSMC.1998.728168` (cited on pages 2, 33).

[ZMW$^+$22a] Tian Zhou, Ziqing Ma, Qingsong Wen, Liang Sun, Tao Yao, Wotao Yin, Rong Jin, et al. Film: frequency improved legendre memory model for long-term time series forecasting. *Advances in Neural Information Processing Systems*, 35:12677–12690, 2022 (cited on page 50).

[ZMW$^+$22b] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pages 27268–27286. PMLR, 2022 (cited on page 50).

[ZZL$^+$17] Chu Zhang, Jianzhong Zhou, Chaoshun Li, Wenlong Fu, and Tian Peng. A compound structure of elm based on feature selection and parameter optimization using hybrid backtracking search algorithm for wind speed forecasting. *Energy Conversion and Management*, 143:360–376, 2017. ISSN: 0196-8904. DOI: `https://doi.org/10.1016/j.enconman.2017.04.007`. URL: `https://www.sciencedirect.com/science/article/pii/S0196890417303126` (cited on page 72).

[ZZP$^+$21] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: beyond efficient transformer for long sequence time-series forecasting. In *AAAI'21*, 2021 (cited on pages 2, 8, 13, 18–20, 33, 50).