

Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting

Bryan Lim*

Department of Engineering Science
University of Oxford
blim@robots.ox.ac.uk

Nicolas Loeff

Google Cloud AI
nloeff@google.com

Sercan Ö. Arık

Google Cloud AI
soarik@google.com

Tomas Pfister

Google Cloud AI
tpfister@google.com

ABSTRACT

Multi-horizon forecasting problems often contain a complex mix of inputs – including static (i.e. time-invariant) covariates, known future inputs, and other exogenous time series that are only observed historically – without any prior information on how they interact with the target. While several deep learning models have been proposed for multi-step prediction, they typically comprise black-box models which do not account for the full range of inputs present in common scenarios. In this paper, we introduce the Temporal Fusion Transformer (TFT) – a novel attention-based architecture which **combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics**. To learn temporal relationships at different scales, the TFT utilizes recurrent layers for local processing and interpretable self-attention layers for learning long-term dependencies. The TFT also uses specialized components for the judicious selection of relevant features and a series of gating layers to suppress unnecessary components, enabling high performance in a wide range of regimes. On a variety of real-world datasets, we demonstrate significant performance improvements over existing benchmarks, and showcase three practical interpretability use-cases of TFT.

KEYWORDS

Interpretable deep learning, time series forecasting, attention mechanisms.

1 INTRODUCTION

Multi-horizon forecasting, i.e the prediction of variables-of-interest at multiple future time steps, is a crucial aspect of machine learning for time series data. **In contrast to one-step-ahead predictions, multi-horizon forecasting provides decision makers access to estimates across the entire path, allowing them to optimize their course of action at multiple steps in future. One common aspect of major forecasting scenarios is the availability of different data sources – including known information about the future (e.g. holiday dates), other exogenous time series, and static metadata – without any prior knowledge on how they interact.** As such, the identification of key drivers of predictions can be important for decision makers, providing additional insights into temporal dynamics. For instance, static (i.e. time-invariant) covariates often play a key role – such as

in healthcare where genetic information can determine the expression of a disease [34]. Given the numerous real-world applications of multi-horizon forecasting, e.g. in retail [5, 10], medicine [26, 42] and economics [6], improvements in existing methods bear much significance for practitioners in many domains.

Deep neural networks have increasingly been used in multi-horizon time series forecasting, demonstrating strong performance improvements over traditional time-series models [1, 28, 31]. While many architectures have focused on recurrent neural network designs [15, 31, 39], recent improvements have considered the use of attention-based methods to **enhance the selection of relevant time-steps in the past** [13] – including Transformer-based models in [25]. However, these methods often fail to consider all different types of inputs commonly present in multi-horizon prediction problems, either assuming that all exogenous inputs are known into the future [15, 25, 31] – a common problem with autoregressive models – **or neglecting important static covariates [39] – which are simply concatenated with other time-dependent features at each step.** With many improvements in time-series models resulting from the alignment of architectures with unique data characteristics [23, 30], **similar performance gains could also be reaped by designing networks with suitable inductive biases for multi-horizon forecasting.**

Most multi-horizon prediction architectures are ‘black-box’ models, where forecasts are controlled by complex nonlinear interactions between many parameters, that render explainability challenging. In turn, **poor interpretability can make it difficult for model builders to improve the model quality, for business decision makers to trust a model’s outputs or for customers in understanding the outcomes of a product** – due to the lack of insights into what is driving its forecast. Moreover, commonly-used methods for interpretability in deep neural networks can be further limited in time series settings. Conventional **post-hoc explainability methods** (e.g. LIME [32] and SHAP [27]), for example, typically do not consider the time ordering of input features – with surrogate models independently constructed for each data-point or with features assumed to be independent of others (including those at neighboring time steps). This can potentially lead to poor quality explanations for time series data, where dependencies between time steps are typically significant. **In addition, attention-based models, such as the Transformer architecture [36], only provide insights on the relevant time-steps in their conventional form, but not into important features.**

*Completed as part of internship with Google Cloud AI Research

In this paper, we propose the Temporal Fusion Transformer (TFT) – **an attention-based architecture which combines high performance multi-horizon forecasting with interpretable insights**. For performance improvements over state-of-the-art benchmarks, we introduce several novel adjustments to align the architecture with the full range of potential inputs and temporal relationships common to multi-horizon forecasting – specifically incorporating **(1) static covariate encoders which encode context vectors for use in other parts of the network, (2) gating mechanisms throughout and sample-dependent variable selection to minimize the contributions of irrelevant inputs, (3) a sequence-to-sequence layer to locally process known and observed inputs, and 4) a temporal self-attention decoder to learn any long-term dependencies present within the dataset**. The use of specialized components also facilitates interpretability, for which we propose three use cases: **to identify (i) globally-important variables for the prediction problem, (ii) persistent temporal patterns, and (iii) significant events**. On real-world data, we show how these methods can be practically applied and the insights they bring.

2 RELATED WORKS

Deep Learning Models for Multi-horizon Forecasting. In line with traditional methods for multi-horizon forecasting [29, 35], recent deep learning methods can be categorized into iterated approaches using autoregressive models [15, 25, 31] or direct methods using sequence-to-sequence models [13, 39].

Iterated approaches utilize one-step-ahead prediction models, with multi-step predictions obtained by recursively feeding predictions into future inputs. For instance, approaches with Long Short-term Memory (LSTM) [20] networks have been considered – such as Deep AR models [15] which use 3 stacked LSTM layers to generate parameters of one-step-ahead Gaussian predictive distributions. Deep State-Space Models (DSSM) [31] adopt a similar approach, utilizing LSTMs to generate parameters of a predefined linear state-space model with predictive distributions produced via Kalman filtering – with extensions for multivariate time series data in [38]. More recently, Transformer-based architectures have been explored in [25], which propose the use of convolutional layers for local processing, and a sparse attention mechanism to increase the size of the receptive field during forecasting. Despite their simplicity, iterative methods rely on the assumption that the values of all variables excluding the target are known at forecast time – such that only the target needs to be recursively fed into future inputs. However, in many practical scenarios, numerous useful time-varying inputs exist. As they are unknown in advance, their straightforward use is limited for iterative approaches. TFTs, on the other hand, explicitly account for the diversity of inputs – naturally handling static covariates and known/unknown time-varying inputs.

Direct methods are trained to explicitly generate forecasts for multiple predefined horizons at each time step. Their architectures typically rely on sequence-to-sequence models, using LSTM encoders to summarize historical inputs, and a variety of methods to generate future predictions. The Multi-horizon Quantile Recurrent Forecaster (MQRNN) [39], for example, utilizes LSTM or convolutional encoders to generate context vectors, which are feed

into multi-layer perceptrons (MLPs) for each horizon. In [13] a multi-modal attention mechanism is used with LSTM encoders to construct context vectors for a bi-directional LSTM decoder. Despite performance gains over LSTM-based iterative methods, interpretability is still not straightforward with standard direct methods. In contrast, we show 3 use-cases for interpreting attention patterns in TFTs to produce general insights about temporal dynamics, while maintaining state-of-the-art performance on a variety of datasets.

Time Series Interpretability with Attention Weights. Attention mechanisms are used in translation [36], image classification [37] or tabular learning [3] to identify salient portions for a *specific* example – using the magnitude of attention weights to determine the importance of different locations. Recent papers have also proposed the use of attention-based mechanisms for time-series interpretability [1, 7, 25], with both LSTM-based [33] and transformer-based [25] models. However, the importance of static covariates – which may be applicable across all time-steps – may be lost with temporal importance, as these methods typically blend variables at each input. TFT alleviates this by using separate encoder-decoder attention for static features at each time step, on top of the self-attention to determine the contribution time-varying inputs.

Instance-wise Variable Importance with Deep Neural Networks. Instance-wise variable importance can be obtained with post-hoc explanation methods [27, 32, 40] and inherently-interpretable models [7, 18]. Post-hoc explanation methods, such as LIME [32], SHAP [27] and RL-LIM [40], are applied on pre-trained black-box models. They are often based on distilling into a surrogate interpretable model, or decomposing into feature attributions. These methods are not designed to take into account the time-wise ordering of inputs – i.e. they ignore sequential dependencies between the input – making it challenging to directly apply them to complex time series data. Inherently-interpretable model design approaches build components for feature selection directly into the architecture itself. For time-series forecasting specifically, they are based on explicitly quantifying time-dependent variable contributions. For example, Interpretable Multi-Variable LSTMs [18] partition the hidden state such that each variable contributes uniquely to its own memory segment, and weights memory segments to determine variable contributions. Methods combining temporal importance and variable selection have also been considered in [7], which computes a single contribution coefficient based on attention weights from each. However, in addition to modelling only one-step-ahead prediction problems, existing methods also focus on *instance-specific* interpretations of attention weights – without providing insights into general temporal dynamics. In contrast, the use-cases demonstrated in Section 7 demonstrate the capability of TFT in analyzing global temporal relationships to build insights about the data as a whole.

3 MULTI-HORIZON FORECASTING

The general problem of multi-horizon forecasting is depicted in Fig. 1. Let there be I unique entities in a given time series dataset – such as different stores for retail forecasting or patients in the medical context. Each entity i is associated with a set of static covariates $\mathbf{s}_i \in \mathbb{R}^{m_s}$, as well as inputs $\mathbf{x}_{i,t} \in \mathbb{R}^{m_x}$ and scalar targets $y_{i,t} \in \mathbb{R}$ at each time-step $t \in [0, T_i]$. In a general sense,

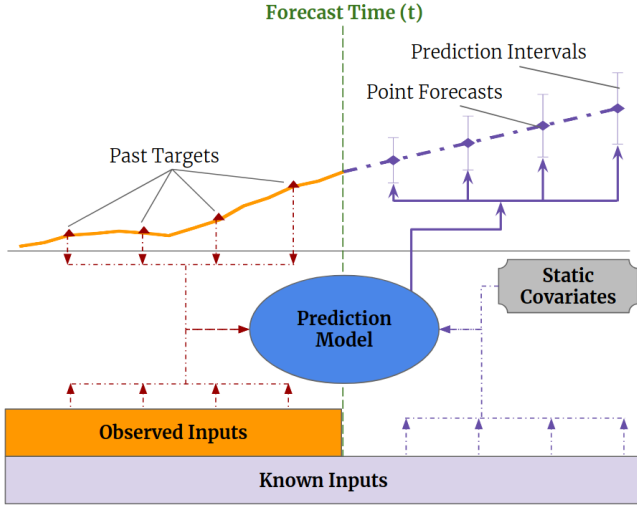


Figure 1: Illustration of Multi-horizon Forecasting with static covariates and various time-dependent inputs

time-dependent input features are subdivided into two categories $\chi_{i,t} = [z_{i,t}^T, \mathbf{x}_{i,t}^T]^T$ – i.e. observed inputs $z_{i,t} \in \mathbb{R}^{(m_z)}$ which can only be measured at each step and are unknown beforehand, and known inputs $\mathbf{x}_{i,t} \in \mathbb{R}^{m_x}$ which can be predetermined (e.g. the day-of-week at time t).

In many scenarios, the provision for prediction intervals can be useful for optimizing business decisions and risk management, by giving decision makers an indication of likely best and worst-case values that the target can take. As such, **we adopt quantile regression to our multi-horizon forecasting setting (e.g. outputting the 10th, 50th and 90th percentiles at each time step)**. Each quantile forecast takes the form:

$$\hat{y}_i(q, t, \tau) = f(q, \tau, y_{i,t-k:t}, z_{i,t-k:t}, \mathbf{x}_{i,t-k:t+\tau}, \mathbf{s}_i), \quad (1)$$

where $\hat{y}_{i,t+\tau}(q, t, \tau)$ is the predicted q^{th} sample quantile of the τ -step-ahead forecast at time t , and $f_q(\cdot)$ is a prediction model. **In line with other direct methods, our model simultaneously outputs forecasts for τ_{\max} discrete prediction horizons** – i.e. $\tau \in \{1, \dots, \tau_{\max}\}$. We incorporate all past information within a finite look-back window k , using target and known inputs only up till and including the forecast start time t (i.e. $y_{i,t-k:t} = \{y_{i,t-k}, \dots, y_{i,t}\}$) and known inputs across the entire range (i.e. $\mathbf{x}_{i,t-k:t+\tau} = \{\mathbf{x}_{i,t-k}, \dots, \mathbf{x}_{i,t}, \dots, \mathbf{x}_{i,t+\tau}\}$). For notational simplicity, we omit the subscript i through the papers unless explicitly required.

4 MODEL ARCHITECTURE

We design the Temporal Fusion Transformer (TFT) to use canonical components to efficiently build feature representations for each input type (i.e. static, known inputs, observed inputs), enabling it to obtain high forecasting performance on a wide range of problems. The major constituents of the TFT are:

- (1) **Gating Mechanisms** – **to skip over any unused components of the architecture, providing adaptive depth and network complexity to accommodate a wide range of datasets**

and scenarios. Gated Linear Units extensively are utilized throughout our architecture, and Gated Residual Network is proposed as a main building block.

- (2) **Variable Selection Networks** – to select relevant input variables at each time step.
- (3) **Static Covariate Encoders** – to integrate static features into the network, through encoding of context vectors to condition temporal dynamics.
- (4) **Temporal Processing** – to learn both long- and short-term temporal relationships, while naturally handling both observed and a priori known time-varying inputs. A sequence-to-sequence layer is employed for local feature processing, whereas long-term dependencies are captured using a novel interpretable multi-head attention block.
- (5) **Multi-Horizon Forecast Intervals Prediction** – to yield quantile forecasts produced at each prediction horizon.

Fig. 2 shows the high level architecture of the TFT, with individual components described in detail in the subsequent sections. An open-source implementation of the model can also be found on GitHub¹ for full reproducibility.

4.1 Gating Mechanisms

As previously highlighted, the precise relationship between exogenous inputs and targets is often unknown in advance, making it difficult to anticipate which variables are relevant. Moreover, it makes it difficult to determine the extent of non-linear processing required, and there may be instances where simpler models can be beneficial – e.g. when datasets are small or noisy.

To apply non-linear processing only where needed, we introduce the Gated Residual Network (GRN) as a basic building block of TFT, as shown in in Fig. 2. At the lowest level, the GRN takes in a primary input \mathbf{a} and an optional context vector \mathbf{c} and yields:

$$\text{GRN}_{\omega}(\mathbf{a}, \mathbf{c}) = \text{LayerNorm}(\mathbf{a} + \text{GLU}_{\omega}(\eta_1)), \quad (2)$$

$$\eta_1 = \mathbf{W}_{1,\omega} \eta_2 + \mathbf{b}_{1,\omega}, \quad (3)$$

$$\eta_2 = \text{ELU}(\mathbf{W}_{2,\omega} \mathbf{a} + \mathbf{W}_{3,\omega} \mathbf{c} + \mathbf{b}_{2,\omega}), \quad (4)$$

where ELU is the Exponential Linear Unit activation function [8], $\eta_1 \in \mathbb{R}^{d_{\text{model}}}$, $\eta_2 \in \mathbb{R}^{d_{\text{model}}}$ are intermediate layers, LayerNorm is standard layer normalization of [24], and ω is an index used to denote how weights are shared. **We adopt component gating layers based on Gated Linear Units (GLUs) [11] to provide the flexibility to suppress any parts of the architecture that are not required for a given dataset.** Letting $\gamma \in \mathbb{R}^{d_{\text{model}}}$ be the input, the GLU then takes the form:

$$\text{GLU}_{\omega}(\gamma) = \sigma(\mathbf{W}_{3,\omega} \gamma + \mathbf{b}_{3,\omega}) \odot (\mathbf{W}_{4,\omega} \gamma + \mathbf{b}_{4,\omega}), \quad (5)$$

where $\sigma(\cdot)$ is the sigmoid activation function, $\mathbf{W}_{(\cdot)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$, $\mathbf{b}_{(\cdot)} \in \mathbb{R}^{d_{\text{model}}}$ are the weights and biases, \odot is the element-wise Hadamard product, and d_{model} is the hidden state size (common across the TFT). GLU allows the TFT to control the extent to which the GRN contributes to the original input \mathbf{a} – potentially skipping over the layer entirely if necessary. For instances without a context vector, the GRN simply treats the context input as zero – i.e. $\mathbf{c} = 0$ in

¹GitHub URL: <https://github.com/google-research/google-research/tree/master/tft>

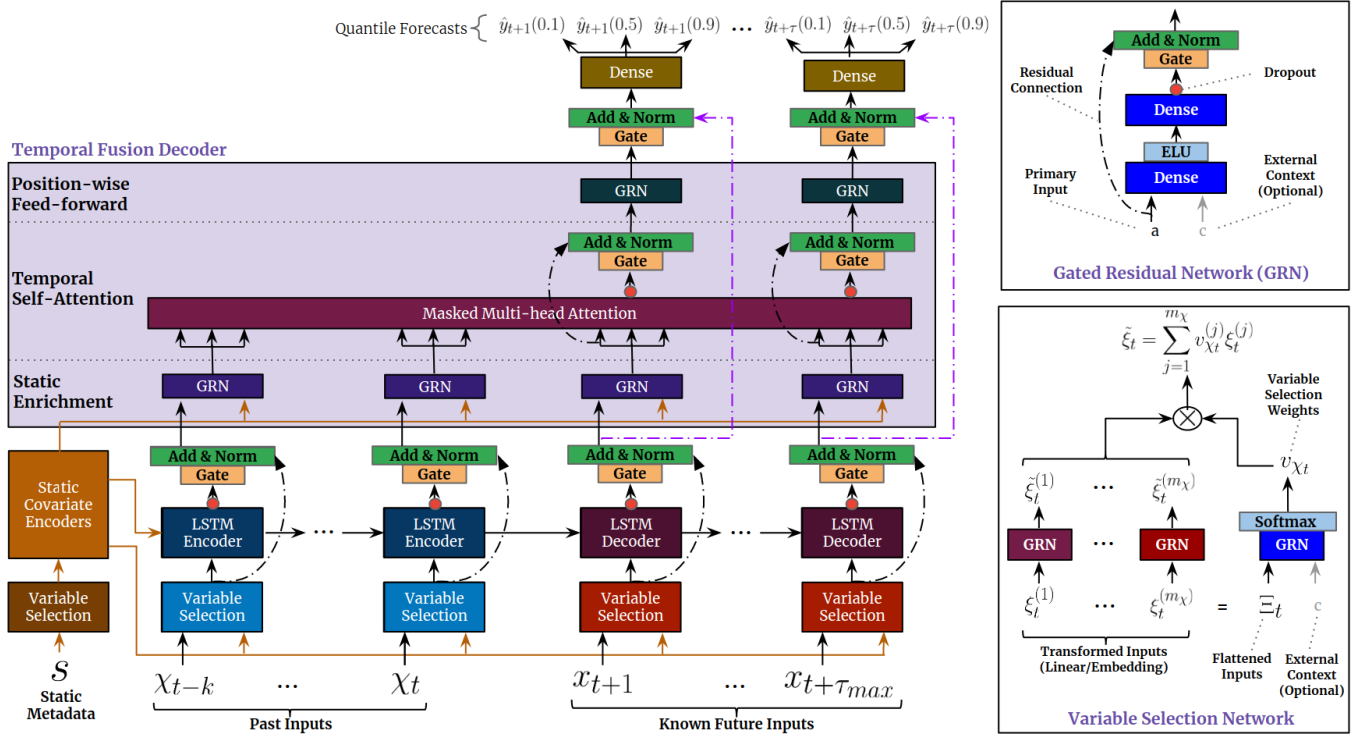


Figure 2: Temporal Fusion Transformer (TFT) architecture. TFT inputs static metadata, time-varying past inputs and time-varying a priori known future inputs. Variable Selection Network is used for judicious selection of the most salient features based on the input. Gated Residual Network blocks enable efficient information flow with skip connections and gating layers. Time-dependent processing in TFT is based on LSTMs for local processing, and multi-head attention for integrating information from any time step.

Eq. (4). During training, dropout is applied before the gating layer and layer normalization – i.e. to η_1 in Eq. (3).

4.2 Variable Selection Networks

While multiple variables may be available, their relevance and specific contribution to the output are typically unknown. The TFT is designed to provide instance-wise variable selection – through the use of variable selection networks applied to both static covariates and time-dependent covariates. Beyond providing insights into which variables are most significant for the prediction problem, variable selection also allows the TFT to remove any unnecessary noisy inputs which could negatively impact performance.

We use entity embeddings [16] for categorical variables and linear transformations for continuous variables, to transform each input variable into a (d_{model})-dimensional vector – matching the dimensions in subsequent layers for skip connections. In addition, all static, past and future inputs make use of separate variable selection networks as denoted by different colors in Fig. 2. Without loss of generality, we present the variable selection network for historical inputs – noting that those for other inputs take the same form.

Let $\xi_t^{(j)} \in \mathbb{R}^{d_{model}}$ denote the transformed input of the j -th variable at time t , with $\Xi_t = \left[\xi_t^{(1)T}, \dots, \xi_t^{(m_\chi)^T} \right]^T$ being the flattened

vector of all historical inputs at time t . Variable selection weights are generated by feeding both Ξ_t and an external context vector c_s through a GRN, followed by a Softmax layer:

$$\mathbf{v}_{\chi_t} = \text{Softmax} \left(\text{GRN}_{\chi_t}(\Xi_t, c_s) \right), \quad (6)$$

where $\mathbf{v}_{\chi_t} \in \mathbb{R}^{m_\chi}$ is a vector of variable selection weights, and c_s is obtained from a static covariate encoder (see Section 4.3). For static variables, we note that the context vector c_s is omitted – given that it already has access to static information.

At each time step, an additional layer of non-linear processing is employed by feeding each $\xi_t^{(j)}$ through its own GRN:

$$\tilde{\xi}_t^{(j)} = \text{GRN}_{\xi_t^{(j)}} \left(\xi_t^{(j)} \right), \quad (7)$$

where $\tilde{\xi}_t^{(j)}$ is the processed feature vector for variable i . We note that each variable has its own $\text{GRN}_{\xi_t^{(j)}}$, with *weights shared across all time steps t* . Processed features are then weighted by their variable selection weights and combined as below:

$$\tilde{\xi}_t = \sum_{j=1}^{m_\chi} v_{\chi_t}^{(j)} \tilde{\xi}_t^{(j)}, \quad (8)$$

where $v_{\chi_t}^{(j)}$ is the j -th element of vector \mathbf{v}_{χ_t} .

4.3 Static Covariate Encoders

To build complex representations of static metadata, we use four separate GRN encoders to produce different context vectors. These are then wired into various locations in the temporal fusion decoder (Section 4.5) where static variables play an important role in processing. Specifically, this includes contexts for 1) temporal variable selection (c_s), 2) local processing of temporal features (c_c, c_h), and 3) enriching of temporal features with static information (c_e). As an example, taking ζ to be the output of the static variable selection network, contexts for temporal variable selection would be encoded according to $c_s = \text{GRN}_{c_s}(\zeta)$.

4.4 Interpretable Multi-Head Attention

To learn long-term relationships across different time steps, TFT employs a self-attention mechanism. In a broad sense, attention mechanisms scale values $V \in \mathbb{R}^{N \times d_V}$ based on relationships between keys $K \in \mathbb{R}^{N \times d_{\text{attn}}}$ and queries $Q \in \mathbb{R}^{N \times d_{\text{attn}}}$ as below:

$$\text{Attention}(Q, K, V) = A(Q, K)V, \quad (9)$$

where $A()$ is a normalization function. A common choice is scaled dot-product attention [36]:

$$A(Q, K) = \text{Softmax}(QK^T / \sqrt{d_{\text{attn}}}). \quad (10)$$

In the canonical form used in the Transformer [36], multi-head attention uses different heads to attend to different representation subspaces, with each head applying the mechanism of Eq. (9):

$$\text{MultiHead}(Q, K, V) = [H_1, \dots, H_{m_H}] W_H, \quad (11)$$

$$H_h = \text{Attention}(Q W_Q^{(h)}, K W_K^{(h)}, V W_V^{(h)}), \quad (12)$$

where $W_Q^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$, $W_K^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$, $W_V^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_V}$ are head-specific weights for keys, queries and values, and $W_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{\text{model}}}$ linearly combines outputs concatenated from all heads H_h .

Given that different values are used in each head, analyzing attention weights alone would not be indicative of a particular feature's overall importance. As such, we modify multi-head attention to share values in each head, and employ additive aggregation of all heads at the output:

$$\text{InterpretableMultiHead}(Q, K, V) = \tilde{H} \tilde{W}_H, \quad (13)$$

$$\tilde{H} = \tilde{A}(Q, K) V W_V, \quad (14)$$

$$= \left\{ \frac{1}{H} \sum_{h=1}^{m_H} A(Q W_Q^{(h)}, K W_K^{(h)}) \right\} V W_V, \quad (15)$$

$$= \frac{1}{H} \sum_{h=1}^{m_H} \text{Attention}(Q W_Q^{(h)}, K W_K^{(h)}, V W_V) \quad (16)$$

where $W_V \in \mathbb{R}^{d_{\text{model}} \times d_V}$ are value weights shared across all heads, and $W_H \in \mathbb{R}^{d_{\text{attn}} \times d_{\text{model}}}$ is used for final linear mapping. From Eq. (15), we see that each head is able to learn different temporal patterns, while attending to a common set of input features – which can be interpreted as a simple ensemble over attention weights into combined matrix $\tilde{A}(Q, K)$ in Eq. (14). Compared to $A(Q, K)$ in Eq. (10), we can see that $\tilde{A}(Q, K)$ yields an increased representation capacity in an efficient way.

4.5 Temporal Fusion Decoder

The temporal fusion decoder uses the series of layers described below to learn temporal relationships present in the dataset:

4.5.1 Locality Enhancement with Sequence-to-Sequence Layer. Points of significance in time series data are often identified in relation to its surrounding values – such as anomalies, change-points or cyclical patterns. Leveraging local context, through the construction of features that utilize pattern information on top of point-wise values, can thus lead to performance improvements in attention-based architectures, as also highlighted in [25]. For instance, [25] adopt a single convolutional layer for locality enhancement – extracting local patterns using the same filter across all time. However, this might not be suitable for cases when observed inputs exist, due to the differing number of past and future inputs. As such, we propose the application of a sequence-to-sequence model to naturally handle these differences – feeding $\tilde{\xi}_{t-k:t}$ into the encoder and $\xi_{t+1:t+\tau_{\text{max}}}$ into the decoder. This then generates a set of uniform temporal features which serve as inputs into the temporal fusion decoder itself – denoted by $\phi(t, n) \in \{\phi(t, -k), \dots, \phi(t, \tau_{\text{max}})\}$ with n being a position index. For comparability with commonly-used sequence-to-sequence baselines, we consider the use of an LSTM encoder-decoder model – although other models can potentially be adopted as well. This also serves as a replacement for standard positional encoding, providing an appropriate inductive bias for the time ordering of the inputs. Moreover, to allow static metadata to influence local processing, we use the c_c, c_h context vectors from the static covariate encoders to initialize the cell state and hidden state respectively for the first LSTM in the layer. We also employ a gated skip connection over this layer:

$$\tilde{\phi}(t, n) = \text{LayerNorm}(\tilde{\xi}_{t+n} + \text{GLU}_{\tilde{\phi}}(\phi(t, n))), \quad (17)$$

where $n \in [-k, \tau_{\text{max}}]$ is a position index.

4.5.2 Static Enrichment Layer. As static covariates often have a significant influence on the temporal dynamics (e.g. genetic information on disease risk), we introduce a static enrichment layer that enhances temporal features with static metadata. For a given position index n , static enrichment takes the form:

$$\theta(t, n) = \text{GRN}_{\theta}(\tilde{\phi}(t, n), c_e), \quad (18)$$

where the weights of GRN_{ϕ} are shared across the entire layer, and c_e is a context vector from a static covariate encoder.

4.5.3 Temporal Self-Attention Layer. Following static enrichment, we next apply self-attention to the temporal features. All static-enriched temporal features are first grouped into a single matrix – i.e. $\Theta(t) = [\theta(t, -k), \dots, \theta(t, \tau)]^T$ – and interpretable multi-head attention (see Section 4.4) is applied at each forecast time (with $N = \tau_{\text{max}} + k + 1$):

$$B(t) = \text{InterpretableMultiHead}(\Theta(t), \Theta(t), \Theta(t)), \quad (19)$$

to yield $B(t) = [\beta(t, -k), \dots, \beta(t, \tau_{\text{max}})]$. $d_V = d_{\text{attn}} = d_{\text{model}}/m_H$ are chosen, where m_H is the number of heads. Decoder masking [25, 36] is applied to the multi-head attention layer to ensure that each temporal dimension can only attend to features preceding it. Besides preserving causal information flow via masking, the self-attention layer allows the TFT to pick up long-range dependencies

that may be challenging for RNN-based architectures to learn. Following the self-attention layer, an additional gating layer is also applied to facilitate training:

$$\delta(t, n) = \text{LayerNorm}(\theta(t, n) + \text{GLU}_\delta(\beta(t, n))). \quad (20)$$

4.5.4 Position-wise Feed-forward Layer. Lastly, we apply an additional non-linear processing to the outputs of the self-attention layer. Similar to the static enrichment layer, this makes use of a series of GRNs:

$$\psi(t, n) = \text{GRN}_\psi(\delta(t, n)), \quad (21)$$

where the weights of GRN_ψ are shared across the entire layer. As per Fig. 2, we also apply a gated residual connection which skips over the entire transformer block, providing a direct path to the sequence-to-sequence layer – yielding a simpler model if additional complexity is not required, as shown below:

$$\tilde{\psi}(t, n) = \text{LayerNorm}(\tilde{\phi}(t, n) + \text{GLU}_{\tilde{\psi}}(\psi(t, n))), \quad (22)$$

4.6 Quantile Outputs

In line with previous work [39], the TFT also generates prediction intervals on top of point forecasts. This is achieved by the simultaneous prediction of various percentiles (e.g. 10th, 50th and 90th) at each time step. Quantile forecasts are generated using linear transformation of the output from the temporal fusion decoder:

$$\hat{y}(q, t, \tau) = \mathbf{W}_q \tilde{\psi}(t, \tau) + b_q, \quad (23)$$

where $\mathbf{W}_q \in \mathbb{R}^{1 \times d}$, $b_q \in \mathbb{R}$ are linear coefficients for the specified quantile q . We note that forecasts are only generated for horizons in the future – i.e. $\tau \in \{1, \dots, \tau_{\max}\}$.

5 TRAINING PROCEDURE

As per [39], the TFT is trained by jointly minimizing the quantile loss terms summed across all quantile outputs:

$$\mathcal{L}(\Omega, \mathbf{W}) = \sum_{y_t \in \Omega} \sum_{q \in \mathcal{Q}} \sum_{\tau=1}^{\tau_{\max}} \frac{QL(y_t, \hat{y}(q, t - \tau, \tau), q)}{M\tau_{\max}}, \quad (24)$$

$$QL(y, \hat{y}, q) = q(y - \hat{y})_+ + (1 - q)(\hat{y} - y)_+, \quad (25)$$

where Ω is the domain of training data containing M samples, \mathbf{W} represents the weights of the TFT, $\mathcal{Q} = \{0.1, 0.5, 0.9\}$ is the set of output quantiles, and $(\cdot)_+ = \max(0, \cdot)$. For out-of-sample testing, we evaluate the normalized quantile losses across the entire forecasting horizon – focusing on P50 and P90 risk for consistency with previous work [15, 25, 31]:

$$q\text{-Risk} = \frac{2 \sum_{y_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{\max}} QL(y_t, \hat{y}(q, t - \tau, \tau), q)}{\sum_{y_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{\max}} |y_t|}, \quad (26)$$

where $\tilde{\Omega}$ is the domain of test samples. For additional information, full details on hyperparameter optimization and training can be found in Appendix B.

6 PERFORMANCE EVALUATION

6.1 Datasets

We choose datasets to reflect commonly observed characteristics across a wide range of challenging multi-horizon forecasting problems. To establish a baseline and position with respect to prior academic work, we first evaluate performance on the Electricity and Traffic datasets used in [15, 25, 31] – which focus on simpler univariate time series containing known inputs only alongside the target. Next, the Retail dataset helps us benchmark the model using the full range of complex inputs observed in multi-horizon prediction applications (see Section 3) – including static metadata and observed time-varying inputs. Finally, to evaluate robustness to over-fitting on smaller noisy datasets, we consider the financial application of volatility forecasting – using a dataset much smaller than others. Broad descriptions of each dataset can be found below:

Electricity. The UCI Electricity Load Diagrams Dataset contains the electricity consumption of 370 customers – aggregated on an hourly level as in [41]. In accordance with [15], we use the past week of data (i.e. 168 hours) to forecast consumption over the next day (i.e. 24 hours).

Traffic. The UCI PEM-SF Traffic Dataset describes the occupancy rate (with $y_t \in [0, 1]$) for 440 San Francisco Bay Area freeways – as in [41]. This is also aggregated on an hourly level as per the electricity dataset, with the same look back window and forecast horizon.

Retail. Favorita Grocery Sales Dataset from the Kaggle competition [14], that combines metadata for different products and the stores, along with other exogenous time-varying inputs sampled at the daily level. We forecast log product sales 30 days into the future, using 90 days of historical information.

Volatility. The OMI realized library [19] contains daily realized volatility values of 31 stock indices computed from intraday data, along with their daily returns. For our experiments, we consider forecasts over the next week (i.e. 5 business days) using information over the past year (i.e. 252 business days).

For each dataset, we partition all time series into 3 sections – a training set for network calibration, a validation set for hyperparameter optimisation, and a hold-out test set for performance evaluation. Full details on the feature engineering steps and train/test splits are provided for each dataset in Appendix A.

6.2 Benchmarks

We extensively compare the TFT to a wide range of machine learning models for multi-horizon forecasting, based on the categories described in Section 2. Hyperparameter optimization is conducted using random search over a pre-defined search space, using the same number of iterations across all benchmarks for a give dataset. Additional details on benchmark model training are also included in Appendix B for reference.

Direct methods. As the TFT falls within this class of multi-horizon models, we primarily focus comparisons on deep learning methods

Dataset	Reported Metrics From [25]						Seq2Seq	MQRNN	TFT
	ARIMA	ETS	TRMF	DeepAR	DSSM	ConvTrans			
Electricity	0.154 (+180%)	0.102 (+85%)	0.084 (+53%)	0.075 (+36%)	0.083 (51%)	0.059 (+7%)	0.067 (+22%)	0.077 (+40%)	0.055*
Traffic	0.223 (+135%)	0.236 (+148%)	0.186 (+96%)	0.161 (+69%)	0.167 (+76%)	0.122 (+28%)	0.105 (+11%)	0.117 (+23%)	0.095*

(a) P50 losses on simpler univariate datasets.

Dataset	Reported Metrics From [25]						Seq2Seq	MQRNN	TFT
	ARIMA	ETS	TRMF	DeepAR	DSSM	ConvTrans			
Electricity	0.102 (+278%)	0.077 (+185%)	-	0.040 (+48%)	0.056 (+107%)	0.034 (+26%)	0.036 (+33%)	0.036 (+33%)	0.027*
Traffic	0.137 (+94%)	0.148 (+110%)	-	0.099 (+40%)	0.113 (+60%)	0.081 (+15%)	0.075 (+6%)	0.082 (+16%)	0.070*

(b) P90 losses on simpler univariate datasets.

Dataset	CovTrans	Seq2Seq	MQRNN	TFT
Volatility	0.047 (+20%)	0.042 (+7%)	0.042 (+7%)	0.039*
Retail	0.429 (+21%)	0.411 (+16%)	0.379 (+7%)	0.354*

(c) P50 losses on datasets with rich static or observed inputs.

Dataset	CovTrans	Seq2Seq	MQRNN	TFT
Volatility	0.024 (+22%)	0.021 (+8%)	0.021 (+9%)	0.020*
Retail	0.192 (+30%)	0.157 (+7%)	0.152 (+3%)	0.147*

(d) P90 losses on datasets with rich static or observed inputs.

Table 1: P50 and P90 quantile losses on a range of real-world datasets. Percentages in brackets reflect the increase in quantile loss versus the TFT (lower q -Risk better), with the TFT outperforming across all experiments.

which directly generate prediction at future horizons. This specifically includes 1) simple sequence-to-sequence models with global contexts (Seq2Seq), and 2) the Multi-horizon Quantile Recurrent Forecaster (MQRNN) – both of which are described in [39].

Iterative methods. To position with respect to the rich body of work on iterative models, we evaluate the TFT using the same setup as [15] for the Electricity and Traffic datasets. This extends the results from [25] for 1) DeepAR models [15], 2) Deep State Space Models (DSSM) [31], and 3) the Transformer-based architecture of [25] with local convolutional processing – which refer to as ConvTrans. For more complex datasets, we focus on the ConvTrans model given its strong outperformance over other iterative models in prior work. As models in this category require knowledge of all inputs in the future to generate predictions, we accommodate this for complex datasets by imputing unknown inputs with their last available value.

6.3 Results and Discussion

Tables 1 show that the TFT significantly outperforms all benchmarks over the variety of datasets described in Section 6.1. For median forecasts, the TFT yields 7% lower P50 and 9% lower P90 losses on average compared to the next best model – demonstrating the benefits of explicitly aligning the architecture with the general multi-horizon forecasting problem. Further ablation analyses can be found in Appendix C.

Comparing direct and iterative model performance, we observe the importance of accounting for the observed inputs – noting the poorer results of ConvTrans on complex datasets where observed input imputation is required (i.e. Volatility and Retail). Furthermore, the benefits of quantile regression are also observed when targets are not modelled well by conditional Gaussians with directly method outperforming in those scenarios. This can be seen, for

example, from the Traffic dataset where target distribution is significantly skewed – with more than 90% of occupancy rates falling between 0 and 0.1, and the remainder distributed evenly until 1.0.

7 INTERPRETABILITY USE CASES

Having established the performance benefits of our model, we next demonstrate how to analyze components of the TFT to interpret the general relationships it has learned. We demonstrate three interpretability use-cases: **1) examining the importance of each input variable in prediction**, **2) visualizing persistent temporal patterns**, and **3) identifying any regimes or events that lead to significant changes in temporal dynamics**. In contrast with other examples of attention-based interpretability [1, 25, 33], which zoom in on interesting but instance-specific examples, we note that our methods focus on ways to aggregate the patterns across the entire dataset – allowing us to extract generalizable insights about temporal dynamics.

7.1 Analyzing Variable Importance

We first quantify variable importance by analyzing the variable selection weights described in Section 4.2. Concretely, we aggregate selection weights (i.e. $v_{\chi_t}^{(j)}$ in Eq. (8)) for each variable across our entire test set, recording the 10th, 50th and 90th percentiles of each sampling distribution. Given its wide range of inputs, we present results on the Retail dataset in Table 2 – with the remainder presented in Appendix D.1. Overall, the TFT focuses on only a subset of key inputs that significantly contribute to predictions.

7.2 Visualizing Persistent Temporal Patterns

The analysis of persistent temporal patterns is often key to understanding the time-dependent relationships present in a given dataset. For instance, lag models are frequently adopted to study length of time required for an intervention to take effect [12] – such

	10%	50%	90%
Item Num	0.198	0.230	0.251
Store Num	0.152	0.161	0.170
City	0.094	0.100	0.124
State	0.049	0.060	0.083
Type	0.005	0.006	0.008
Cluster	0.108	0.122	0.133
Family	0.063	0.075	0.079
Class	0.148	0.156	0.163
Perishable	0.084	0.085	0.088

(a) Static Covariates

	10%	50%	90%
Transactions	0.029	0.033	0.037
Oil	0.062	0.081	0.105
On-promotion	0.072	0.075	0.078
Day of Week	0.007	0.007	0.008
Day of Month	0.083	0.089	0.096
Month	0.109	0.122	0.136
National Hol	0.131	0.138	0.145
Regional Hol	0.011	0.014	0.018
Local Hol	0.056	0.068	0.072
Open	0.027	0.044	0.067
Log Sales	0.304	0.324	0.353

(b) Past Inputs

	10%	50%	90%
On-promotion	0.155	0.170	0.182
Day of Week	0.029	0.065	0.089
Day of Month	0.056	0.116	0.138
Month	0.111	0.155	0.240
National Hol	0.145	0.220	0.242
Regional Hol	0.012	0.014	0.060
Local Hol	0.116	0.151	0.239
Open	0.088	0.095	0.097

(c) Future Inputs

Table 2: Variable importance for the Retail dataset. The 10th, 50th and 90th percentiles of observed variable selection weights are shown, with values larger than 0.1 are highlighted in purple. For static covariates, the largest weights are attributed to variables which uniquely identify different entities (i.e. item number and store number). For historical inputs, past values of the target (i.e. log sales) are critical as expected, as the forecasts are extrapolations of past observations. For future inputs, promotion periods and national holidays have the greatest influence on sales forecasts, in line with typical periods of increased customer spending.

as the impact of a government’s increase in public expenditure on the resultant growth in Gross National Product [4]. Seasonality models are also commonly used in econometrics to identify periodic patterns in a target-of-interest [21] and measure the length of each cycle. Using the attention weights present in the self-attention layer of the temporal fusion decoder, we present a method below to identify similar persistent patterns – by measuring the contributions of features at fixed lags in the past on forecasts at various horizons.

Combining Eq. (14) and (19), we see that the self-attention layer contains a matrix of attention weights at each forecast time t – i.e. $\tilde{A}(\phi(t), \phi(t))$. As such, multi-head attention outputs at each forecast horizon τ (i.e. $\beta(t, \tau)$) can be described as an attention-weighted sum of lower level features at each position n :

$$\beta(t, \tau) = \sum_{n=-k}^{\tau_{max}} \alpha(t, n, \tau) \tilde{\theta}(t, n), \quad (27)$$

where $\alpha(t, n, \tau)$ is the (τ, n) -th element of $\tilde{A}(\phi(t), \phi(t))$, and $\tilde{\theta}(t, n)$ is a row of $\tilde{\Theta}(t) = \Theta(t)W_V$. Due to decoder masking, we also note that $\alpha(t, i, j) = 0, \forall i > j$. For each forecast horizon τ , the importance of a previous time point $n < \tau$ can hence be determined by analyzing distributions of $\alpha(t, n, \tau)$ across all time steps and entities. We present results for the Traffic dataset below, with similar findings on Electricity and Retail presented in Appendix D.2.

Fig. 3 shows the temporal patterns learned by the TFT – with the top chart recording the mean along the 10th, 50th and 90th percentiles of the attention weights for one-step-ahead forecasts (i.e. $\alpha(t, 1, \tau)$) over the test set, and the average attention weights for various horizons (i.e. $\tau \in \{5, 10, 15, 20\}$) on the bottom. Based on the regularly-spaced peaks at 24-hour intervals, we can infer that the TFT has learned a strong daily seasonal pattern for predictions – placing the largest attention on the same hour of preceding days.

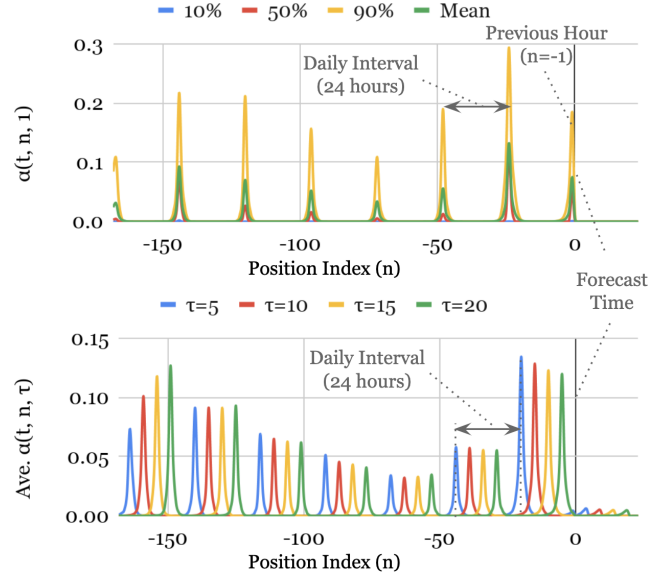


Figure 3: Visualizing persistent temporal patterns: daily seasonal patterns in Traffic dataset. Top – percentiles of attention weights for one-step-ahead forecast. Bottom – average attention weights for forecast at various horizons.

7.3 Identifying Regimes & Significant Events

Apart from persistent patterns, identifying sudden changes in temporal patterns can also be very useful, as temporary shifts can occur due to the presence of significant regimes or events. For instance, regime-switching behavior has been widely documented in financial markets [2], with returns characteristics – such as volatility – being observed to change abruptly between regimes.

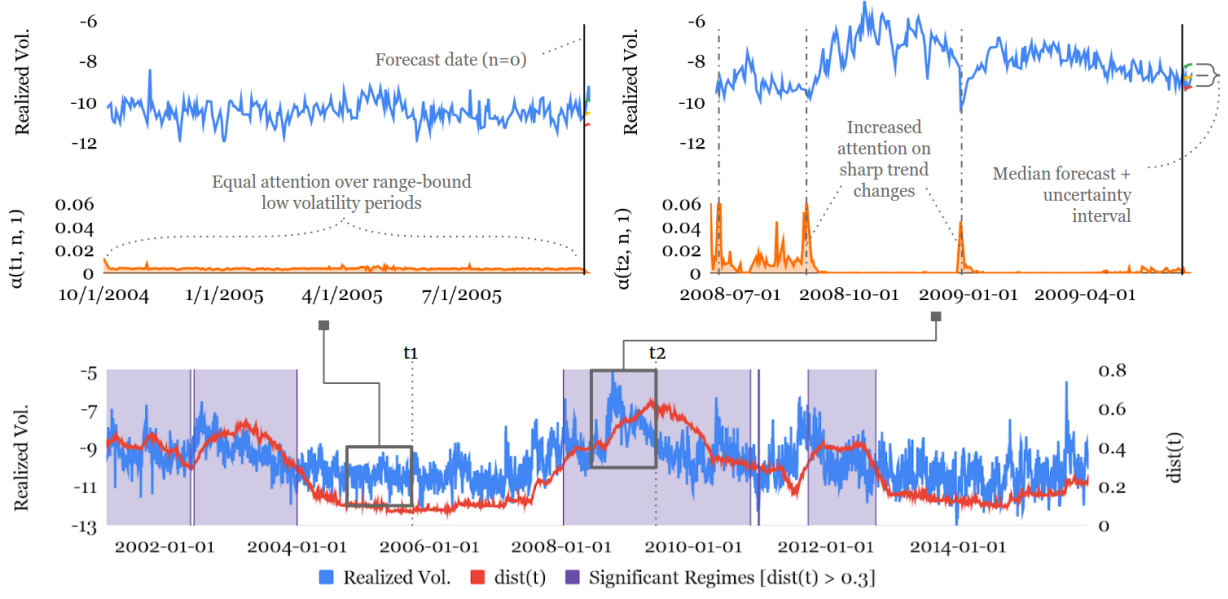


Figure 4: Regime identification for S&P 500 realized volatility. Significant deviations in attention patterns can be observed around periods of high volatility – corresponding to the peaks observed in $\text{dist}(t)$. We use a threshold of $\text{dist}(t) > 0.3$ to denote significant regimes, as highlighted in purple. Focusing on periods around the 2008 financial crisis, the top right plot visualizes $\alpha(t, n, 1)$ midway through the significant regime, compared to the normal regime on the top left.

Firstly, for a given entity, we define the average attention pattern per forecast horizon to be:

$$\bar{\alpha}(n, \tau) = \sum_{t=1}^T \alpha(t, j, \tau) / T, \quad (28)$$

and then construct $\bar{\alpha}(\tau) = [\bar{\alpha}(-k, \tau), \dots, \bar{\alpha}(\tau_{\max}, \tau)]^T$. To compare similarities between attention weight vectors, we use the distance metric proposed by [9]:

$$\kappa(\mathbf{p}, \mathbf{q}) = \sqrt{1 - \rho(\mathbf{p}, \mathbf{q})}, \quad (29)$$

where $\rho(\mathbf{p}, \mathbf{q}) = \sum_j \sqrt{p_j q_j}$ is the Bhattacharyya coefficient [22] measuring the overlap between discrete distributions – with p_j, q_j being elements of probability vectors \mathbf{p}, \mathbf{q} respectively. For each entity, significant shifts in temporal dynamics are then measured using the distance between attention vectors at each point with the average pattern, aggregated for all horizons as below:

$$\text{dist}(t) = \frac{1}{\tau_{\max}} \sum_{\tau=1}^{\tau_{\max}} \kappa(\bar{\alpha}(\tau), \alpha(t, \tau)), \quad (30)$$

where $\alpha(t, \tau) = [\alpha(t, -k, \tau), \dots, \alpha(t, \tau_{\max}, \tau)]^T$.

We use the volatility dataset as a test case for regime identification, specifically applying our distance metric to the attention patterns for the S&P 500 index over our training period from 2001 to 2015. Plotting $\text{dist}(t)$ against the target (i.e. log realized volatility) in the bottom chart of Fig. 4, significant deviations in attention patterns can be observed around periods of high volatility – corresponding to the peaks observed in $\text{dist}(t)$. From the plots, we can see that the TFT appears to alter its behaviour between regimes – placing equal attention across historical inputs when volatility

is low, while attending more to sharp trend changes during high volatility periods – suggesting differences in temporal dynamics learned in each.

8 CONCLUSIONS

We introduce the Temporal Fusion Transformer (TFT) – a novel attention-based deep neural network model for interpretable high-performance multi-horizon time series forecasting. The TFT utilizes specialized components to handle the full range of inputs typically present in multi-horizon forecasting problems (i.e. static covariates, a priori known inputs, and observed inputs). Specifically, these include: 1) sequence-to-sequence and attention based temporal processing components that capture time-varying relationships at different timescales, 2) static covariate encoders that allow the network to condition temporal forecasts on static metadata, 3) gating components that enable skipping over any parts of the network that are unnecessary for a given dataset, 4) variable selection networks that select relevant input features at each time step, and 5) quantile predictions to obtain output intervals across all prediction horizons. Through tests on a series of real-world datasets, we show that the TFT achieves state-of-the-art forecasting performance on both simple datasets that contain only known inputs, and complex datasets which encompass the full range of possible inputs. Finally, we investigate the general relationships learned by the TFT through a series of interpretability use-cases – proposing novel methods to use the TFT to 1) analyze important variables for a given prediction problem, 2) visualize persistent temporal relationships learned (e.g. seasonality), and 3) identify significant regimes present in the dataset.

ACKNOWLEDGMENTS

The authors gratefully acknowledge discussions with Yaguang Li, Maggie Wang, Jeffrey Gu and Andrew Moore that contributed to the development of this paper.

REFERENCES

- [1] A. Alaa and M. van der Schaar. 2019. Attentive State-Space Modeling of Disease Progression. In *Advances in Neural Information Processing Systems 32 (NIPS 2019)*.
- [2] Andrew Ang and Allan Timmermann. 2012. Regime Changes and Financial Markets. *Annual Review of Financial Economics* 4, 1 (2012), 313–337.
- [3] Sercan O. Arik and Tomas Pfister. 2019. TabNet: Attentive Interpretable Tabular Learning. (2019). arXiv:1908.07442
- [4] Badi Baltagi. 2008. *Distributed Lags and Dynamic Models*. Springer Berlin Heidelberg, 129–145.
- [5] Joos-Hendrik Böse et al. 2017. Probabilistic Demand Forecasting at Scale. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1694–1705.
- [6] Carlos Capistran, Christian Constandse, and Manuel Ramos-Francia. 2010. Multi-horizon inflation forecasts using disaggregated data. *Economic Modelling* 27, 3 (2010), 666 – 677.
- [7] Edward Choi et al. 2016. RETAIN: An Interpretable Predictive Model for Healthcare Using Reverse Time Attention Mechanism. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS 2016)*.
- [8] Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *International Conference on Learning Representations (ICLR 2016)*.
- [9] D. Comaniciu, V. Ramesh, and P. Meer. 2003. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 5 (2003), 564–577.
- [10] Pascal Courty and Hao Li. 1999. Timing of Seasonal Sales. *The Journal of Business* 72, 4 (1999), 545–572.
- [11] Yann Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*.
- [12] Sizhen Du, Guojie Song, Lei Han, and Haikun Hong. 2018. Temporal Causal Inference with Time Lag. *Neural Computation* 30, 1 (2018), 271–291.
- [13] Chenyou Fan et al. 2019. Multi-Horizon Time Series Forecasting with Temporal Attention Learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*.
- [14] Corporacion Favorita. 2018. Corporacion Favorita Grocery Sales Forecasting Competition. (2018). <https://www.kaggle.com/c/favorita-grocery-sales-forecasting/>
- [15] Valentin Flunkert et al. 2017. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *CoRR* abs/1704.04110 (2017). arXiv:1704.04110
- [16] Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29*.
- [17] Eleftherios Giovanis. 2014. The Turn-of-The-Month-Effect: Evidence from Periodic Generalized Autoregressive Conditional Heteroskedasticity (PGARCH) Model. *International Journal of Economic Sciences and Applied Research* 7 (12 2014), 43–61.
- [18] Tian Guo, Tao Lin, and Nino Antulov-Fantulin. 2019. Exploring interpretable LSTM neural networks over multi-variable data. In *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*.
- [19] Gerd Heber, Asger Lunde, Neil Shephard, and Kevin K. Sheppard. 2009. Oxford-Man Institute's Realized Library. (2009). <https://realized.oxford-man.ox.ac.uk/>
- [20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.
- [21] Svend Hylleberg (Ed.). 1992. *Modelling Seasonality*. Oxford University Press.
- [22] T. Kailath. 1967. The Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Transactions on Communication Technology* 15, 1 (1967), 52–60.
- [23] Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. 2014. A Clockwork RNN. In *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML 2014)*.
- [24] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv e-prints*, Article arXiv:1607.06450 (Jul 2016). arXiv:1607.06450
- [25] Shiyang Li et al. 2019. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*.
- [26] Bryan Lim, Ahmed Alaa, and Mihaela van der Schaar. 2018. Forecasting Treatment Responses Over Time Using Recurrent Marginal Structural Networks. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*.
- [27] Scott Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*.
- [28] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2020. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting* 36, 1 (2020), 54 – 74.
- [29] M. Marcellino, J Stock, and M. Watson. 2006. A Comparison of Direct and Iterated Multistep AR Methods for Forecasting Macroeconomic Time Series. *Journal of Econometrics* 135 (2006), 499–526.
- [30] Daniel Neil et al. 2016. Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS 2016)*.
- [31] Syama Sundar Rangapuram et al. 2018. Deep State Space Models for Time Series Forecasting. In *Advances in Neural Information Processing Systems 31 (NIPS 2018)*.
- [32] Marco Ribeiro et al. 2016. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In *Proceedings of the 22th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '16)*.
- [33] Huan Song et al. 2018. Attend and Diagnose: Clinical Time Series Analysis Using Attention Models (AAAI 2018).
- [34] Eichler EE Stessman HA, Bernier R. 2014. A genotype-first approach to defining the subtypes of a complex disease. *Cell* 156, 5 (2014), 872–877.
- [35] Souhaib Ben Taieb, Antti Sorjamaa, and Gianluca Bontempi. 2010. Multiple-output modeling for multi-step-ahead time series forecasting. *Neurocomputing* 73, 10 (2010), 1950 – 1957.
- [36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*.
- [37] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang. 2017. Residual Attention Network for Image Classification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [38] Yuyang Wang et al. 2019. Deep Factors for Forecasting. In *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*.
- [39] Ruofeng Wen et al. 2017. A Multi-Horizon Quantile Recurrent Forecaster. In *NIPS 2017 Time Series Workshop*.
- [40] Jinsung Yoon, Sercan O. Arik, and Tomas Pfister. 2019. RL-LIM: Reinforcement Learning-based Locally Interpretable Modeling. (2019). arXiv:cs.LG/1909.12367
- [41] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. 2016. Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction. In *Advances in Neural Information Processing Systems 29*.
- [42] J Zhang and K Nawata. 2018. Multi-step prediction for influenza outbreak by an adjusted long short-term memory. *Epidemiology and infection* 146, 7 (2018).

APPENDIX

A DATASET DESCRIPTION

Additional details on each dataset can be found below. We provide all the sufficient information on feature pre-processing and train/test splits to ensure reproducibility of our results.

Electricity. Per [15], we use 500k samples taken between 2014-01-01 to 2014-09-01 – using the first 90% for training, and the last 10% as a validation set. Testing is done over the 7 days immediately following the training set – as described in [15, 41]. Given the large differences in magnitude between trajectories, we also apply z-score normalization separately to each entity for real-valued inputs. In line with previous work, we consider the electricity usage, day-of-week, hour-of-day and a time index – i.e. the number of time-steps from the first observation – as real-valued inputs, and treat the entity identifier as a categorical variable.

Traffic. Tests on the Traffic dataset are also kept consistent with previous work, using 500k training samples taken before 2008-06-15 as per [15], and split in the same way as the Electricity dataset. For testing, we use the 7 days immediately following the training set, and z-score normalization was applied across all entities. For inputs, we also take traffic occupancy, day-of-week, hour-of-day and a time index as real-valued inputs, and the entity identifier as a categorical variable.

Retail. For the retail dataset, we treat each product number-store number pair as a separate entity, with over 135k entities present within the full dataset. For network calibration, the training set is made up of 450k samples taken between 2015-01-01 to 2015-12-01, validation set of 50k samples from the 30 days after the training set, and test set of all entities over the 30-day horizon following the validation set. We use all inputs supplied by the Kaggle competition (full list in the variable importance section of Appendix D.1 – including additional variables for the day-of-week, day-of-month, and month. Data is also resampled at regular daily intervals, imputing any missing days using the last available observation. We also include an additional 'open' flag to denote whether data is present on a given day. For holidays, we group national, regional, and local holidays as separate categorical variables. We also apply a log-transform on the sales data, and adopt z-score normalization across all entities. We consider log sales, transactions, oil to be real-valued variables – with the remainder treated as categorical inputs.

Volatility. Data is downloaded from 2000-01-03 to 2019-06-28 – with the training set consisting of data before 2016, the validation set from 2016-2017, and the test set data from 2018 onwards. For the target, we focus on 5-min sub-sampled realized volatility (i.e. the `rv5_ss` column), and add daily open-to-close returns as an extra exogenous input. Also, additional variables are included for the day-of-week, day-of-month, week-of-year, and month – along with a 'region' variable for each index (i.e. Americas, Europe or Asia). Finally, a time index is added to denote the number of days from the first day in our training set. We treat all date-related variables (i.e. day-of-week, day-of-month, week-of-year, and month) and the region input as categorical variables. A log transformation is also

applied to the target, and all inputs are z-score normalized across all entities.

B TRAINING DETAILS

Hyperparameter optimization is conducted via random search, using 240 iterations for the smaller Volatility dataset, and 60 iterations for the larger Electricity, Traffic and Retail datasets. Full search ranges for all hyperparameters are below, and the optimal TFT hyperparameters can be found in Table 3:

- **State size** – 10, 20, 40, 80, 160, 240, 320
- **Dropout rate** – 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9
- **Minibatch size** – 64, 128, 256
- **Learning rate** – 0.0001, 0.001, 0.01
- **Max. gradient norm** – 0.01, 1.0, 100.0
- **Num. heads** – 1, 4

To preserve the explainability of interpretable multi-head attention, we adopt only a single stack (i.e. temporal fusion decoder block) for the TFT itself. For ConvTrans, we adopt the same fixed stack size and number of heads used in the original paper [25] – setting them to 8 heads and 3 layers respectively. We also used the full attention model of [25], and treated kernel sizes for the CNN local processing layer as a hyperparameter (i.e. kernel size $\in \{1, 3, 6, 9\}$) – as optimal kernel sizes were observed to be dataset dependent in [25].

C ABLATION ANALYSIS

To quantify the benefits of each of our proposed architectural contribution, we perform an extensive ablation analysis – removing each component from the network as below, and quantifying the percentage increase in loss versus the original architecture:

Gating layers. The effects of gated skip connections are tested by replacing each GLU layer (Eq. (5)) with a simple linear layer passed through an ELU activation function.

Static covariate encoders. The importance of specialized static encoders are tested by setting all context vectors to zero – i.e. $\mathbf{c}_s = \mathbf{c}_e = \mathbf{c}_h = \mathbf{0}$ – and concatenating all transformed static inputs to all time-dependent past and future inputs.

Variable selection networks. The effects of instance-wise variable selection are tested by replacing the softmax outputs of Eq. 8 with a vector of trainable coefficients, and removing the networks generating the variable selection weights. We retain, however, the variable-wise GRNs (i.e. Eq. (7), maintaining the same degree non-linear processing as before.

Self-attention layers. The benefits of the self-attention layer are quantified by replacing the attention matrix used in the interpretable multi-head attention layer (Eq. 14) with a matrix of trainable parameters \mathbf{W}_A – i.e. $\hat{A}(\mathbf{Q}, \mathbf{K}) = \mathbf{W}_A$, where $\mathbf{W}_A \in \mathbb{R}^{N \times N}$. This prevents the TFT from attend to different input features at different forecast times, helping us evaluate the importance of instance-wise attention weights.

		Electricity	Traffic	Retail	Volatility
Dataset Details	Target Type	\mathbb{R}	$[0, 1]$	\mathbb{R}	\mathbb{R}
	Num. Entities	370	440	130k	41
	Num. Samples (Training + Validation)	500k	500k	500k	$\approx 100k$
Optimal Hyperparameters	Random Search Iterations	60	60	60	240
	State Size	160	320	240	160
	Dropout Rate	0.1	0.3	0.1	0.3
	Minibatch Size	64	128	128	64
	Learning Rate	0.001	0.001	0.001	0.01
	Max Gradient Norm	0.01	100	100	0.01
	Num. Heads	4	4	4	1

Table 3: Dataset details and optimal TFT configuration for each dataset.

Sequence-to-sequence layers for local processing. We evaluate the importance of local processing by removing the sequence-to-sequence layer of Section 4.5.1 – replacing this with standard positional encoding used in [36].

Ablated networks are trained across for each dataset using the hyperparameters of Table 3, with full results shown in Figure 5. From the charts, the effects on both P50 and P90 losses are found to be similar across all datasets, with all components contributing to performance improvements on the whole. In general, the components responsible for capturing temporal relationships (i.e. local processing and self-attention layers) have the largest impact on performance, with P90 loss increases of $> 6\%$ on average and $> 20\%$ on select datasets when ablated. Static encoders and variable selection have the next largest impact – increasing P90 losses by more than 2.6% on average and up to 7.9% for specific datasets. Finally, gating layer ablation also significant increases in P90 losses, with a 1.9% increase on average. This is most significantly show on the the volatility dataset (with a 4.1% P90 loss increase), demonstrating the utility of component gating for smaller, noisier datasets.

D ADDITIONAL INTERPRETABILITY RESULTS

On top of the interpretability use cases of Section 7, which highlight our most prominent findings, we also include the remaining results in this section for completeness.

D.1 Variable Importance

Table 4 shows the variable importance scores for the remaining Electricity, Traffic and Volatility datasets. Given that only one static input is present for these datasets, the network allocates full importance for the entity identifier for Electricity and Traffic, as well as for the region input for Volatility. We also observe two general types of import time-dependent inputs – those related to past values of the target as before, and those related to calendar effects. For instance, the hour-of-day plays a significant roles for Electricity and Traffic datasets, echoing the daily seasonality observed in the next section. In the Volatility dataset, the day-of-month is observe

to play a significant role in future inputs – potentially reflecting turn-of-month effects [17].

D.2 Persistent Temporal Patterns

Fig. 6 shows the attention weight patterns across all datasets, and extends the results of Section 7.2. We observe that the three datasets exhibit a seasonal pattern, with clear attention spikes at daily intervals observed for the Electricity and Traffic datasets, and a slightly weaker weekly patterns for the Retail dataset. No strong persistent patterns were observed for the Volatility datasets however, with attention weights equally distributed across all positions on average. This resembles a moving average filter at the feature level, and – given the high degree of randomness associated with the volatility process – could be useful in extracting the trend over the entire period by smoothing out high-frequency noise.

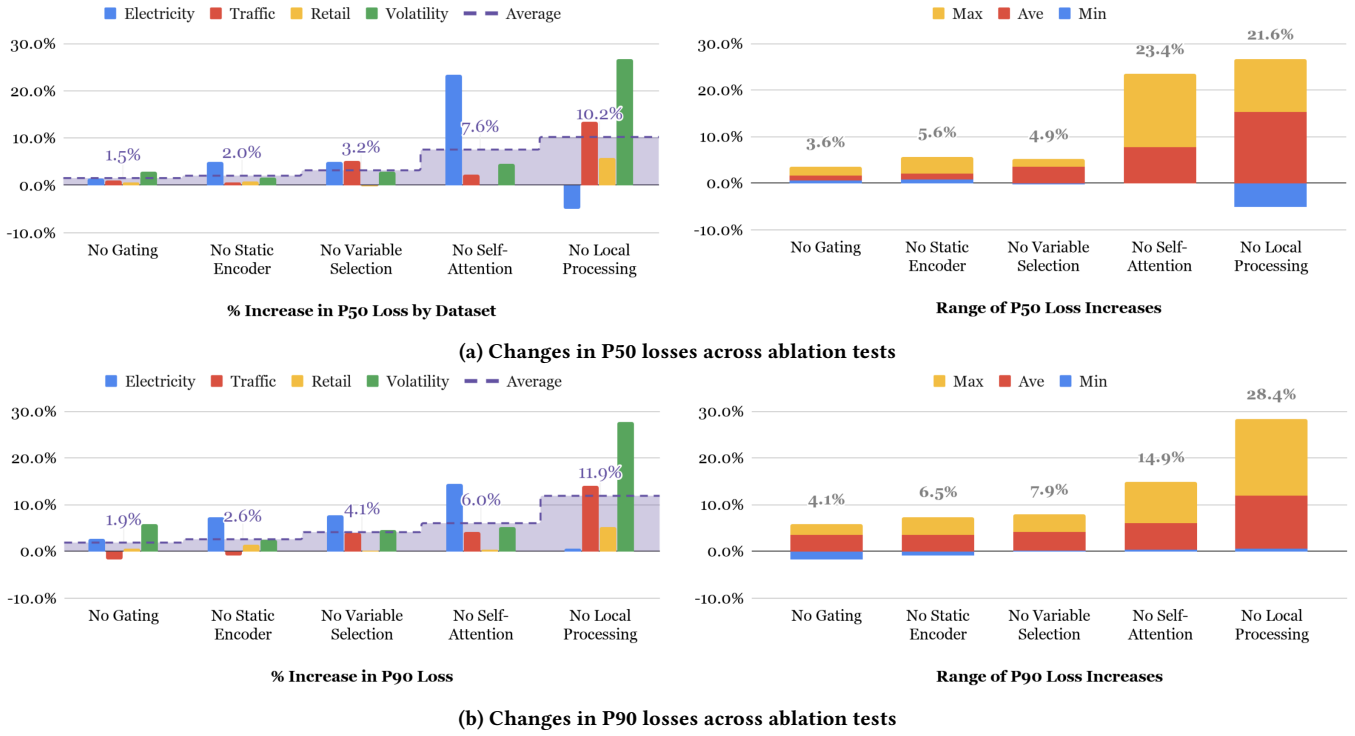


Figure 5: Results of ablation analysis. Both a) and b) show the impact of ablation on the P50 and P90 losses respectively. Results per dataset shown on the left, and the range across datasets shown on the right. While the precise importance of each is dataset-specific, all components contribute significantly on the whole – with the maximum percentage increase over all datasets ranging from 3.6% to 23.4% for P50 losses, and similarly from 4.1% to 28.4% for P90 losses.

	10%	50%	90%
Static Inputs			
ID	1.000	1.000	1.000
Past Inputs			
Hour of Day	0.437	0.462	0.473
Day of Week	0.078	0.099	0.151
Time Index	0.066	0.077	0.092
Power Usage	0.342	0.359	0.366
Future Inputs			
Hour of Day	0.718	0.738	0.739
Day of Week	0.109	0.124	0.166
Time Index	0.114	0.137	0.155

(a) Electricity

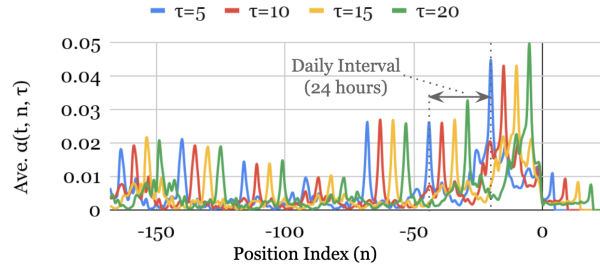
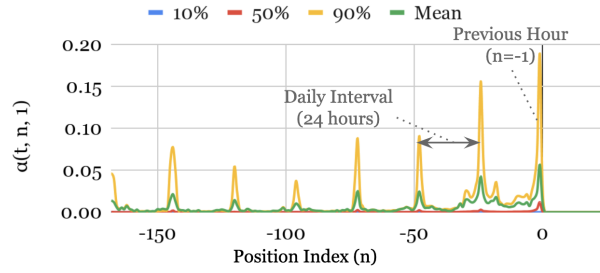
	10%	50%	90%
Static Inputs			
ID	1.000	1.000	1.000
Past Inputs			
Hour of Day	0.285	0.296	0.300
Day of Week	0.117	0.122	0.124
Time Index	0.107	0.109	0.111
Occupancy	0.471	0.473	0.483
Future Inputs			
Hour of Day	0.781	0.781	0.781
Day of Week	0.099	0.100	0.102
Time Index	0.117	0.119	0.121

(b) Traffic

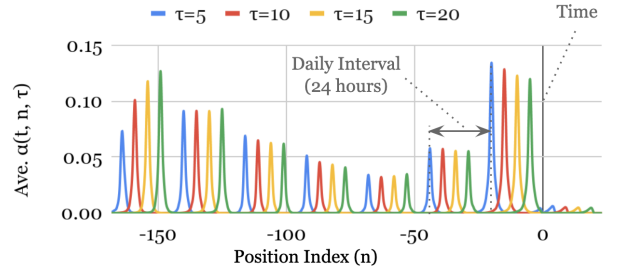
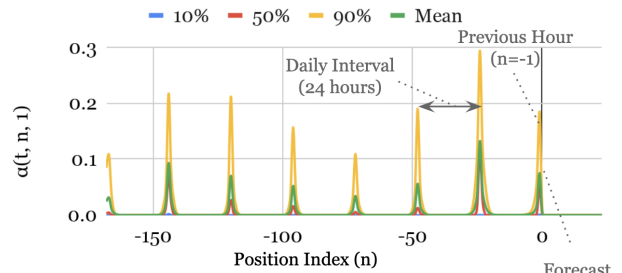
	10%	50%	90%
Static Inputs			
Region	1.000	1.000	1.000
Past Inputs			
Time Index	0.093	0.098	0.142
Day of Week	0.003	0.004	0.004
Day of Month	0.017	0.027	0.028
Week of Year	0.022	0.057	0.068
Month	0.008	0.009	0.011
Open-to-close Returns	0.078	0.158	0.178
Realised Vol	0.620	0.647	0.714
Future Inputs			
Time Index	0.011	0.014	0.024
Day of Week	0.019	0.072	0.299
Day of Month	0.069	0.635	0.913
Week of Year	0.026	0.060	0.227
Month	0.008	0.055	0.713

(c) Volatility

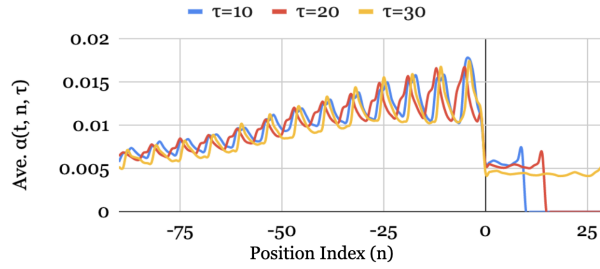
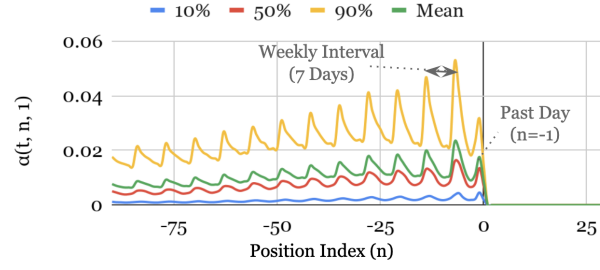
Table 4: Variable importance scores for the Electricity, Traffic and Volatility datasets. The most significant variable of each input category is highlighted in purple. As before, past values of the target play a significant role – being the top 1 or 2 most significant past input across datasets. The role of seasonality can also be seen in Electricity and Traffic, where the past and future values of the hour-of-day is important for forecasts.



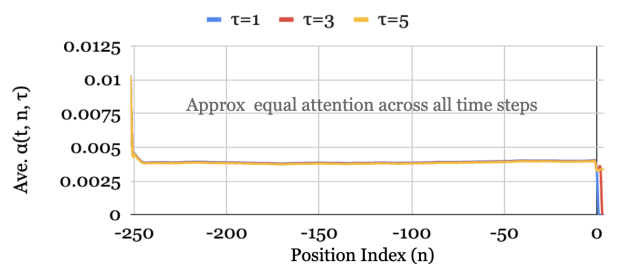
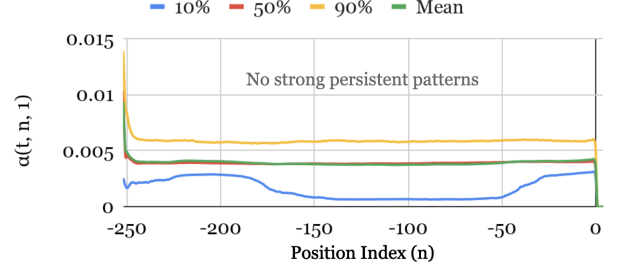
(a) Electricity



(b) Traffic



(c) Retail



(d) Volatility

Figure 6: Persistent temporal patterns across datasets. Top – percentiles of attention weights for one-step-ahead forecast. Bottom – average attention weights for forecast at various horizons. Clear seasonality observed for the Electricity, Traffic and Retail datasets, but no strong persistent patterns seen in Volatility dataset.