Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting

Bryan Lim[a,1,*], Sercan Ö. Arık[b], Nicolas Loeff[b], Tomas Pfister[b]

[a] *University of Oxford, UK*
[b] *Google Cloud AI, USA*

Abstract

Multi-horizon forecasting often contains a complex mix of inputs – including static (i.e. time-invariant) covariates, known future inputs, and other exogenous time series that are only observed in the past – without any prior information on how they interact with the target. Several deep learning methods have been proposed, but they are typically 'black-box' models which do not shed light on how they use the full range of inputs present in practical 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, TFT uses recurrent layers for local processing and interpretable self-attention layers for long-term dependencies. TFT utilizes specialized components to select relevant features and a series of gating layers to suppress unnecessary components, enabling high performance in a wide range of scenarios. 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: Deep learning, Interpretability, Time series, Multi-horizon forecasting, Attention mechanisms, Explainable AI.

1. Introduction

Multi-horizon forecasting, i.e. the prediction of variables-of-interest at multiple future time steps, is a crucial problem within time series machine learning. In contrast to one-step-ahead predictions, multi-horizon forecasts provide users with access to estimates across the entire path, allowing them to optimize their actions at multiple steps in future (e.g. retailers optimizing the inventory for

*Corresponding authors
Email addresses: blim@robots.ox.ac.uk (Bryan Lim), soarik@google.com (Sercan Ö. Arık), nloeff@google.com (Nicolas Loeff), tpfister@google.com (Tomas Pfister)
[1] Completed as part of internship with Google Cloud AI Research.

the entire upcoming season, or clinicians optimizing a treatment plan for a patient). Multi-horizon forecasting has many impactful real-world applications in retail [1, 2], healthcare [3, 4] and economics [5]) – performance improvements to existing methods in such applications are highly valuable.

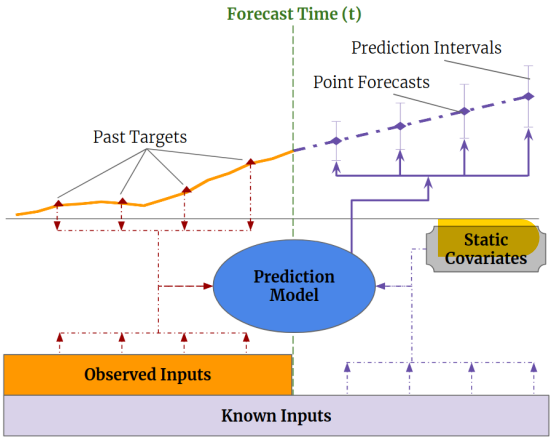


Figure 1: Illustration of multi-horizon forecasting with static covariates, past-observed and apriori-known future time-dependent inputs.

Practical multi-horizon forecasting applications commonly have access to a variety of data sources, as shown in Fig. 1, including known information about the future (e.g. upcoming holiday dates), other exogenous time series (e.g. historical customer foot traffic), and static metadata (e.g. location of the store) – without any prior knowledge on how they interact. This heterogeneity of data sources together with little information about their interactions makes multi-horizon time series forecasting particularly challenging.

Deep neural networks (DNNs) have increasingly been used in multi-horizon forecasting, demonstrating strong performance improvements over traditional time series models [6, 7, 8]. While many architectures have focused on variants of recurrent neural network (RNN) architectures [9, 6, 10], recent improvements have also used attention-based methods to enhance the selection of relevant time steps in the past [11] – including Transformer-based models [12]. However, these often fail to consider the different types of inputs commonly present in multi-horizon forecasting, and either assume that all exogenous inputs are known into the future [9, 6, 12] – a common problem with autoregressive models – or neglect important static covariates [10] – which are simply concatenated with other time-dependent features at each step. Many recent improvements in time series models have resulted from the alignment of architectures with unique data characteristics [13, 14]. We argue and demonstrate that similar performance gains can also be reaped by designing networks with suitable inductive biases for multi-horizon forecasting.

In addition to not considering the heterogeneity of common multi-horizon forecasting inputs, most current architectures are 'black-box' models where fore-

casts are controlled by complex nonlinear interactions between many parameters. This makes it difficult to explain how models arrive at their predictions, and in turn makes it challenging for users to trust a model's outputs and model builders to debug it. Unfortunately, commonly-used explainability methods for DNNs are not well-suited for applying to time series. In their conventional form, post-hoc methods (e.g. LIME [15] and SHAP [16]) do not consider the time ordering of input features. For example, for LIME, surrogate models are independently constructed for each data-point, and for SHAP, features are considered independently for neighboring time steps. Such post-hoc approaches would lead to poor explanation quality as dependencies between time steps are typically significant in time series. On the other hand, some attention-based architectures are proposed with inherent interpretability for sequential data, primarily language or speech – such as the Transformer architecture [17]. The fundamental caveat to apply them is that multi-horizon forecasting includes many different types of input features, as opposed to language or speech. In their conventional form, these architectures can provide insights into relevant time steps for multi-horizon forecasting, but they cannot distinguish the importance of different features at a given timestep. Overall, in addition to the need for new methods to tackle the heterogeneity of data in multi-horizon forecasting for high performance, new methods are also needed to render these forecasts interpretable, given the needs of the use cases.

In this paper we propose the Temporal Fusion Transformer (TFT) – an attention-based DNN architecture for multi-horizon forecasting that achieves high performance while enabling new forms of interpretability. To obtain significant performance improvements over state-of-the-art benchmarks, we introduce multiple novel ideas 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 these specialized components also facilitates interpretability; in particular, we show that TFT enables three valuable interpretability use cases: helping users identify (i) globally-important variables for the prediction problem, (ii) persistent temporal patterns, and (iii) significant events. On a variety of real-world datasets, we demonstrate how TFT can be practically applied, as well as the insights and benefits it provides.

2. Related Work

DNNs for Multi-horizon Forecasting: Similarly to traditional multi-horizon forecasting methods [18, 19], recent deep learning methods can be categorized into iterated approaches using autoregressive models [9, 6, 12] or direct methods based on sequence-to-sequence models [10, 11].

Iterated approaches utilize one-step-ahead prediction models, with multi-step predictions obtained by recursively feeding predictions into future inputs. Approaches with Long Short-term Memory (LSTM) [20] networks have been considered, such as Deep AR [9] which uses stacked LSTM layers to generate parameters of one-step-ahead Gaussian predictive distributions. Deep State-Space Models (DSSM) [6] 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 [21]. More recently, Transformer-based architectures have been explored in [12], which proposes 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, with many unknown in advance. Their straightforward use is hence limited for iterative approaches. TFT, on the other hand, explicitly accounts for the diversity of inputs – naturally handling static covariates and (past-observed and future-known) time-varying inputs.

In contrast, 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, e.g. LSTM encoders to summarize past inputs, and a variety of methods to generate future predictions. The Multi-horizon Quantile Recurrent Forecaster (MQRNN) [10] uses LSTM or convolutional encoders to generate context vectors which are fed into multi-layer perceptrons (MLPs) for each horizon. In [11] a multi-modal attention mechanism is used with LSTM encoders to construct context vectors for a bi-directional LSTM decoder. Despite performing better than LSTM-based iterative methods, interpretability remains challenging for such standard direct methods. In contrast, we show that by interpreting attention patterns, TFT can provide insightful explanations about temporal dynamics, and do so while maintaining state-of-the-art performance on a variety of datasets.

Time Series Interpretability with Attention: Attention mechanisms are used in translation [17], image classification [22] or tabular learning [23] to identify salient portions of input for each instance using the magnitude of attention weights. Recently, they have been adapted for time series with interpretability motivations [7, 12, 24], using LSTM-based [25] and transformer-based [12] architectures. However, this was done without considering the importance of static covariates (as the above methods 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 DNNs: Instance (i.e. sample)-wise variable importance can be obtained with post-hoc explanation methods [15, 16, 26] and inherently intepretable models [27, 24]. Post-hoc explanation methods, e.g. LIME [15], SHAP [16] and RL-LIM [26], are applied on pre-

trained black-box models and often based on distilling into a surrogate interpretable model, or decomposing into feature attributions. They are not designed to take into account the time ordering of inputs, limiting their use for complex time series data. Inherently-interpretable modeling approaches build components for feature selection directly into the architecture. For time series forecasting specifically, they are based on explicitly quantifying time-dependent variable contributions. For example, Interpretable Multi-Variable LSTMs [27] partitions 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 [24], which computes a single contribution coefficient based on attention weights from each. However, in addition to the shortcoming of modelling only one-step-ahead forecasts, existing methods also focus on *instance-specific* (i.e. sample-specific) interpretations of attention weights – without providing insights into global temporal dynamics. In contrast, the use cases in Sec. 7 demonstrate that TFT is able to analyze global temporal relationships and allows users to interpret global behaviors of the model on the whole dataset – specifically in the identification of any persistent patterns (e.g. seasonality or lag effects) and regimes present.

3. Multi-horizon Forecasting

Let there be I unique entities in a given time series dataset – such as different stores in retail or patients in healthcare. Each entity i is associated with a set of static covariates $\boldsymbol{s}_i \in \mathbb{R}^{m_s}$, as well as inputs $\boldsymbol{\chi}_{i,t} \in \mathbb{R}^{m_\chi}$ and scalar targets $y_{i,t} \in \mathbb{R}$ at each time-step $t \in [0, T_i]$. Time-dependent input features are subdivided into two categories $\boldsymbol{\chi}_{i,t} = \left[\boldsymbol{z}_{i,t}^T, \boldsymbol{x}_{i,t}^T\right]^T$ – observed inputs $\boldsymbol{z}_{i,t} \in \mathbb{R}^{(m_z)}$ which can only be measured at each step and are unknown beforehand, and known inputs $\boldsymbol{x}_{i,t} \in \mathbb{R}^{m_x}$ 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 decisions and risk management by yielding 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 10^{th}, 50^{th} and 90^{th} percentiles at each time step). Each quantile forecast takes the form:

$$\hat{y}_i(q, t, \tau) = f_q\left(\tau, y_{i,t-k:t}, \boldsymbol{z}_{i,t-k:t}, \boldsymbol{x}_{i,t-k:t+\tau}, \boldsymbol{s}_i\right), \tag{1}$$

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 $f_q(.)$ is a prediction model. In line with other direct methods, we simultaneously output forecasts for τ_{max} time steps – i.e. $\tau \in \{1, \ldots, \tau_{max}\}$. 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. $y_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$) and known inputs across the entire

range (i.e. $\boldsymbol{x}_{i,t-k:t+\tau} = \{\boldsymbol{x}_{i,t-k}, \ldots, \boldsymbol{x}_{i,t}, \ldots, \boldsymbol{x}_{i,t+\tau}\}$). [2]

4. Model Architecture

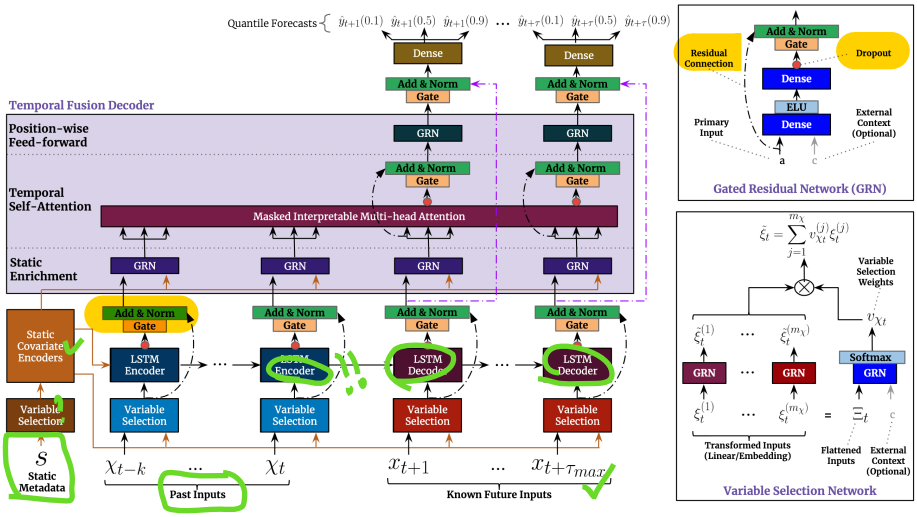


Figure 2: TFT architecture. TFT inputs static metadata, time-varying past inputs and time-varying a priori known future inputs. Variable Selection is used for judicious selection of the most salient features based on the input. Gated Residual Network blocks enable efficient information flow with skip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

We design TFT to use canonical components to efficiently build feature representations for each input type (i.e. static, known, observed inputs) for high forecasting performance on a wide range of problems. The major constituents of 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.
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 from both observed and known time-varying inputs. A sequence-to-sequence layer is employed for local processing, whereas long-term dependencies are captured using a novel interpretable multi-head attention block.

[2] For notation simplicity, we omit the subscript i unless explicitly required.

5. **Prediction intervals** via quantile forecasts to determine the range of likely target values at each prediction horizon.

Fig. 2 shows the high level architecture of Temporal Fusion Transformer (TFT), with individual components described in detail in the subsequent sections.

4.1. Gating Mechanisms

The precise relationship between exogenous inputs and targets is often unknown in advance, making it difficult to anticipate which variables are relevant. Moreover, it is difficult to determine the extent of required non-linear processing, and there may be instances where simpler models can be beneficial – e.g. when datasets are small or noisy. With the motivation of giving the model the flexibility to apply non-linear processing only where needed, we propose Gated Residual Network (GRN) as shown in in Fig. 2 as a building block of TFT. The GRN takes in a primary input $\boldsymbol{a}$ and an optional context vector $\boldsymbol{c}$ and yields:

$$\text{GRN}_{\omega}\left(\boldsymbol{a}, \boldsymbol{c}\right) = \text{LayerNorm}\left(\boldsymbol{a} + \text{GLU}_{\omega}(\boldsymbol{\eta}_1)\right), \tag{2}$$

$$\boldsymbol{\eta}_1 = \boldsymbol{W}_{1,\omega}\ \boldsymbol{\eta}_2 + \boldsymbol{b}_{1,\omega}, \tag{3}$$

$$\boldsymbol{\eta}_2 = \text{ELU}\left(\boldsymbol{W}_{2,\omega}\ \boldsymbol{a} + \boldsymbol{W}_{3,\omega}\ \boldsymbol{c} + \boldsymbol{b}_{2,\omega}\right), \tag{4}$$

where ELU is the Exponential Linear Unit activation function [28], $\boldsymbol{\eta}_1 \in \mathbb{R}^{d_{model}}, \boldsymbol{\eta}_2 \in \mathbb{R}^{d_{model}}$ are intermediate layers, LayerNorm is standard layer normalization of [29], and ω is an index to denote weight sharing. When $\boldsymbol{W}_{2,\omega}\ \boldsymbol{a} + \boldsymbol{W}_{3,\omega}\ \boldsymbol{c} + \boldsymbol{b}_{2,\omega} >> 0$, the ELU activation would act as an identity function and when $\boldsymbol{W}_{2,\omega}\ \boldsymbol{a} + \boldsymbol{W}_{3,\omega}\ \boldsymbol{c} + \boldsymbol{b}_{2,\omega} << 0$, the ELU activation would generate a constant output, resulting in linear layer behavior. We use component gating layers based on Gated Linear Units (GLUs) [30] to provide the flexibility to suppress any parts of the architecture that are not required for a given dataset. Letting $\boldsymbol{\gamma} \in \mathbb{R}^{d_{model}}$ be the input, the GLU then takes the form:

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

where $\sigma(.)$ is the sigmoid activation function, $\boldsymbol{W}_{(.)} \in \mathbb{R}^{d_{model} \times d_{model}}$, $\boldsymbol{b}_{(.)} \in \mathbb{R}^{d_{model}}$ are the weights and biases, $\odot$ is the element-wise Hadamard product, and d_{model} is the hidden state size (common across TFT). GLU allows TFT to control the extent to which the GRN contributes to the original input $\boldsymbol{a}$ – potentially skipping over the layer entirely if necessary as the GLU outputs could be all close to 0 in order to surpress the nonlinear contribution. For instances without a context vector, the GRN simply treats the contex input as zero – i.e. $\boldsymbol{c} = 0$ in Eq. (4). During training, dropout is applied before the gating layer and layer normalization – i.e. to $\boldsymbol{\eta}_1$ 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. 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 TFT to remove any unnecessary noisy inputs which could negatively impact performance. Most real-world time series datasets contain features with less predictive content, thus variable selection can greatly help model performance via utilization of learning capacity only on the most salient ones.

We use entity embeddings [31] for categorical variables as feature representations, and linear transformations for continuous variables – transforming each input variable into a (d_{model})-dimensional vector which matches the dimensions in subsequent layers for skip connections. 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 past inputs – noting that those for other inputs take the same form.

Let $\boldsymbol{\xi}_t^{(j)} \in \mathbb{R}^{d_{model}}$ denote the transformed input of the j-th variable at time t, with $\boldsymbol{\Xi}_t = \left[\boldsymbol{\xi}_t^{(1)^T}, \ldots, \boldsymbol{\xi}_t^{(m_\chi)^T}\right]^T$ being the flattened vector of all past inputs at time t. Variable selection weights are generated by feeding both $\boldsymbol{\Xi}_t$ and an external context vector $\boldsymbol{c}_s$ through a GRN, followed by a Softmax layer:

$$\boldsymbol{v}_{\chi_t} = \text{Softmax}\left(\text{GRN}_{v_\chi}(\boldsymbol{\Xi}_t, \boldsymbol{c}_s)\right), \tag{6}$$

where $\boldsymbol{v}_{\chi_t} \in \mathbb{R}^{m_\chi}$ is a vector of variable selection weights, and $\boldsymbol{c}_s$ is obtained from a static covariate encoder (see Sec. 4.3). For static variables, we note that the context vector $\boldsymbol{c}_s$ 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 $\boldsymbol{\xi}_t^{(j)}$ through its own GRN:

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

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

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

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

4.3. Static Covariate Encoders

In contrast with other time series forecasting architectures, the TFT is carefully designed to integrate information from static metadata, using separate GRN encoders to produce four different context vectors, $\boldsymbol{c}_s$, $\boldsymbol{c}_e$, $\boldsymbol{c}_c$, and $\boldsymbol{c}_h$. These contect vectors are wired into various locations in the temporal fusion

decoder (Sec. 4.5) where static variables play an important role in processing. Specifically, this includes contexts for (1) temporal variable selection ($\boldsymbol{c}_s$), (2) local processing of temporal features ($\boldsymbol{c}_c, \boldsymbol{c}_h$), and (3) enriching of temporal features with static information ($\boldsymbol{c}_e$). As an example, taking $\boldsymbol{\zeta}$ to be the output of the static variable selection network, contexts for temporal variable selection would be encoded according to $\boldsymbol{c}_s = GRN_{c_s}(\boldsymbol{\zeta})$.

4.4. Interpretable Multi-Head Attention

The TFT employs a self-attention mechanism to learn long-term relationships across different time steps, which we modify from multi-head attention in transformer-based architectures [17, 12] to enhance explainability. In general, attention mechanisms scale values $\boldsymbol{V} \in \mathbb{R}^{N \times d_V}$ based on relationships between keys $\boldsymbol{K} \in \mathbb{R}^{N \times d_{attn}}$ and queries $\boldsymbol{Q} \in \mathbb{R}^{N \times d_{attn}}$ as below:

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

where $A()$ is a normalization function. A common choice is scaled dot-product attention [17]:

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

To improve the learning capacity of the standard attention mechanism, multi-head attention is proposed in [17], employing different heads for different representation subspaces:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_{m_H}]\, \boldsymbol{W}_H, \tag{11}$$

$$\boldsymbol{H}_h = \text{Attention}(\boldsymbol{Q}\, \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\, \boldsymbol{W}_K^{(h)}, \boldsymbol{V}\, \boldsymbol{W}_V^{(h)}), \tag{12}$$

where $\boldsymbol{W}_K^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $\boldsymbol{W}_Q^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $\boldsymbol{W}_V^{(h)} \in \mathbb{R}^{d_{model} \times d_V}$ are head-specific weights for keys, queries and values, and $\boldsymbol{W}_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{model}}$ linearly combines outputs concatenated from all heads $\boldsymbol{H}_h$.

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

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

$$\tilde{\boldsymbol{H}} = \tilde{A}(\boldsymbol{Q}, \boldsymbol{K})\, \boldsymbol{V}\, \boldsymbol{W}_V, \tag{14}$$

$$= \left\{1/H \sum_{h=1}^{m_H} A\left(\boldsymbol{Q}\, \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\, \boldsymbol{W}_K^{(h)}\right)\right\} \boldsymbol{V}\, \boldsymbol{W}_V, \tag{15}$$

$$= 1/H \sum_{h=1}^{m_H} \text{Attention}(\boldsymbol{Q}\, \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\, \boldsymbol{W}_K^{(h)}, \boldsymbol{V}\, \boldsymbol{W}_V), \tag{16}$$

where $\boldsymbol{W}_V \in \mathbb{R}^{d_{model} \times d_V}$ are value weights shared across all heads, and $\boldsymbol{W}_H \in \mathbb{R}^{d_{attn} \times d_{model}}$ is used for final linear mapping. From Eq. (15), we see that each

head can 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 $\tilde{A}(\boldsymbol{Q}, \boldsymbol{K})$ in Eq. (14). Compared to $A(\boldsymbol{Q}, \boldsymbol{K})$ in Eq. (10), $\tilde{A}(\boldsymbol{Q}, \boldsymbol{K})$ 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

In time series data, points of significance are often identified in relation to their 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. For instance, [12] adopts 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 $\tilde{\boldsymbol{\xi}}_{t-k:t}$ into the encoder and $\tilde{\boldsymbol{\xi}}_{t+1:t+\tau_{max}}$ into the decoder. This then generates a set of uniform temporal features which serve as inputs into the temporal fusion decoder itself – denoted by $\boldsymbol{\phi}(t, n) \in \{\boldsymbol{\phi}(t, -k), \ldots, \boldsymbol{\phi}(t, \tau_{max})\}$ with n being a position index. For comparability with commonly-used sequence-to-sequence baselines, we consider the use of an LSTM encoder-decoder – 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 $\boldsymbol{c}_c, \boldsymbol{c}_h$ 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:

$$\tilde{\boldsymbol{\phi}}(t, n) = \text{LayerNorm}\left(\tilde{\boldsymbol{\xi}}_{t+n} + \text{GLU}_{\tilde{\phi}}(\boldsymbol{\phi}(t, n))\right), \tag{17}$$

where $n \in [-k, \tau_{max}]$ 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:

$$\boldsymbol{\theta}(t, n) = \text{GRN}_{\theta}\left(\tilde{\boldsymbol{\phi}}(t, n), \boldsymbol{c}_e\right), \tag{18}$$

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

4.5.3. Temporal Self-Attention Layer

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

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

to yield $\boldsymbol{B}(t) = [\boldsymbol{\beta}(t,-k), \ldots, \boldsymbol{\beta}(t,\tau_{max})]$. $d_V = d_{attn} = d_{model}/m_H$ are chosen, where m_H is the number of heads. Decoder masking [17, 12] 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 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:

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

4.5.4. Position-wise Feed-forward Layer

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 GRNs:

$$\boldsymbol{\psi}(t,n) = \text{GRN}_{\psi}\left(\boldsymbol{\delta}(t,n)\right), \tag{21}$$

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:

$$\tilde{\boldsymbol{\psi}}(t,n) = \text{LayerNorm}\left(\tilde{\boldsymbol{\phi}}(t,n) + \text{GLU}_{\tilde{\psi}}(\boldsymbol{\psi}(t,n))\right), \tag{22}$$

4.6. Quantile Outputs

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

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

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

5. Loss Functions

TFT is trained by jointly minimizing the quantile loss [10], summed across all quantile outputs:

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

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

where Ω is the domain of training data containing M samples, $\boldsymbol{W}$ represents the weights of TFT, $\mathcal{Q}$ is the set of output quantiles (we use $\mathcal{Q} = \{0.1, 0.5, 0.9\}$ in our experiments, and $(.)_+ = \max(0, .)$. 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 [9, 6, 12]:

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

where $\tilde{\Omega}$ is the domain of test samples. Full details on hyperparameter optimization and training can be found in Appendix A.

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 [9, 6, 12] – 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 Sec. 3) – including rich 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, containing the electricity consumption of 370 customers – aggregated on an hourly level as in [32]. In accordance with [9], we use the past week (i.e. 168 hours) to forecast over the next 24 hours.
- **Traffic:** The UCI PEM-SF Traffic Dataset describes the occupancy rate (with $y_t \in [0, 1]$) of 440 SF Bay Area freeways – as in [32]. It 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 [33], 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 past information.
- **Volatility (or Vol.):** The OMI realized library [34] 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).

6.2. Training Procedure

For each dataset, we partition all time series into 3 parts – a training set for learning, a validation set for hyperparameter tuning, and a hold-out test set for performance evaluation. Hyperparameter optimization is conducted via random search, using 240 iterations for Volatility, and 60 iterations for others. Full search ranges for all hyperparameters are below, with datasets and optimal model parameters listed in Table 1.

- **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 explainability, we adopt only a single interpretable multi-head attention layer. For ConvTrans [12], we use the same fixed stack size (3 layers) and number of heads (8 heads) as in [12]. We keep the same attention model, and treat kernel sizes for the convolutional processing layer as a hyperparameter ($\in \{1, 3, 6, 9\}$) – as optimal kernel sizes are observed to be dataset dependent [12]. An open-source implementation of the TFT on these datasets can be found on GitHub[3] for full reproducibility.

6.3. Computational Cost

Across all datasets, each TFT model was also trained on a single GPU, and can be deployed without the need for extensive computing resources. For instance, using a NVIDIA Tesla V100 GPU, our optimal TFT model (for Electricity dataset) takes just slightly over 6 hours to train (each epoch being roughly 52 mins). The batched inference on the entire validation dataset (consisting 50,000 samples) takes 8 minutes. TFT training and inference times can be further reduced with hardware-specific optimizations.

Table 1: Information on dataset and optimal TFT configuration.

	Electricity	**Traffic**	**Retail**	**Vol.**
Dataset Details				
Target Type	$\mathbb{R}$	[0, 1]	$\mathbb{R}$	$\mathbb{R}$
Number of Entities	370	440	130k	41
Number of Samples	500k	500k	500k	$\sim$100k
Network Parameters				
k	168	168	90	252
τ_{max}	24	24	30	5
Dropout Rate	0.1	0.3	0.1	0.3
State Size	160	320	240	160
Number of Heads	4	4	4	1
Training Parameters				
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

6.4. Benchmarks

We extensively compare TFT to a wide range of models for multi-horizon forecasting, based on the categories described in Sec. 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 are included in Appendix A.

Direct methods: As TFT falls within this class of multi-horizon models, we primarily focus comparisons on deep learning models which directly generate prediction at future horizons, including: 1) simple sequence-to-sequence models with global contexts (Seq2Seq), and 2) the Multi-horizon Quantile Recurrent Forecaster (MQRNN) [10].

Iterative methods: To position with respect to the rich body of work on iterative models, we evaluate TFT using the same setup as [9] for the Electricity and Traffic datasets. This extends the results from [12] for 1) DeepAR [9], 2) DSSM [6], and 3) the Transformer-based architecture of [12] 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, and DeepAR due to its popularity among practitioners. 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.

For simpler univariate datasets, we note that the results for ARIMA, ETS, TRMF, DeepAR, DSSM and ConvTrans have been reproduced from [12] in

[3]URL: `https://github.com/google-research/google-research/tree/master/tft`

Table 2: P50 and P90 quantile losses on a range of real-world datasets. Percentages in brackets reflect the increase in quantile loss versus TFT (lower q-Risk better), with TFT outperforming competing methods across all experiments, improving on the next best alternative method (underlined) between 3% and 26%.

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

(a) P50 losses on simpler univariate datasets.

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

(b) P90 losses on simpler univariate datasets.

	DeepAR	**CovTrans**	**Seq2Seq**	**MQRNN**	**TFT**
Vol.	0.050 (*+28%*)	0.047 (*+20%*)	0.042 (*+7%*)	0.042 (*+7%*)	**0.039***
Retail	0.574 (*+62%*)	0.429 (*+21%*)	0.411 (*+16%*)	0.379 (*+7%*)	**0.354***

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

	DeepAR	**CovTrans**	**Seq2Seq**	**MQRNN**	**TFT**
Vol.	0.024 (*+21%*)	0.024 (*+22%*)	0.021 (*+8%*)	0.021 (*+9%*)	**0.020***
Retail	0.230 (*+56%*)	0.192 (*+30%*)	0.157 (*+7%*)	0.152 (*+3%*)	**0.147***

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

Table 2 for consistency.

6.5. Results and Discussion

Table 2 shows that TFT significantly outperforms all benchmarks over the variety of datasets described in Sec. 6.1. For median forecasts, 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.

Comparing direct and iterative models, 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 captured well by Gaussian distributions with direct models 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.

6.6. 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:** We ablate by replacing each GLU layer (Eq. (5)) with a simple linear layer followed by ELU.
- **Static covariate encoders:** We ablate by setting all context vectors to zero – i.e. $\boldsymbol{c}_s{=}\boldsymbol{c}_e{=}\boldsymbol{c}_c{=}\boldsymbol{c}_h{=}\boldsymbol{0}$ – and concatenating all transformed static inputs to all time-dependent past and future inputs.
- **Instance-wise variable selection networks:** We ablate by replacing the softmax outputs of Eq. 6 with trainable coefficients, and removing the networks generating the variable selection weights. We retain, however, the variable-wise GRNs (see Eq. (7)), maintaining a similar amount of non-linear processing.
- **Self-attention layers:** We ablate by replacing the attention matrix of the interpretable multi-head attention layer (Eq. 14) with a matrix of trainable parameters $\boldsymbol{W}_A$ – i.e. $\tilde{A}(\boldsymbol{Q}, \boldsymbol{K}) = \boldsymbol{W}_A$, where $\boldsymbol{W}_A \in \mathbb{R}^{N \times N}$. This prevents TFT from attending to different input features at different times, helping evaluation of the importance of instance-wise attention weights.
- **Sequence-to-sequence layers for local processing:** We ablate by replacing the sequence-to-sequence layer of Sec. 4.5.1 with standard positional encoding used in [17].

Ablated networks are trained across for each dataset using the hyperparameters of Table 1. Fig. 3 shows that the effects on both P50 and P90 losses are similar across all datasets, with all components contributing to performance improvements on the whole.

In general, the components responsible for capturing temporal relationships, 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. The diversity across time series datasets can also be seen from the differences in the ablation impact of the respective temporal components. Concretely, while local processing is critical in Traffic, Retail and Volatility, lower post-ablation P50 losses indicate that it can be detrimental in Electricity – with the self-attention layer playing a more vital role. A possible explanation is that persistent daily seasonality appears to dominate other temporal relationships in the Electricity dataset. For this dataset, Table B.4 of Appendix B also shows that the hour-of-day has the largest variable importance score across all temporal inputs, exceeding even the target (i.e. Power Usage) itself. In contrast to other dataset where past target observations are more significant (e.g. Traffic), direct attention to previous days seem to help learning daily seasonal patterns in Electricity – with local processing between adjacent time steps being less necessary. We can account for this by treating the sequence-to-sequence architecture in the temporal fusion decoder as a hyperparameter to tune, including an option for simple positional encoding without any local processing.

Static covariate encoders and instance-wise variable selection have the next largest impact – increasing P90 losses by more than 2.6% and 4.1% on average. The biggest benefits of these are observed for electricity dataset, where some of the input features get very low importance.

Finally, gating layer ablation also shows increases in P90 losses, with a 1.9% increase on average. This is the most significant on the volatility (with a 4.1% P90 loss increase), underlying the benefit of component gating for smaller and noisier datasets.

7. Interpretability Use Cases

Having established the performance benefits of our model, we next demonstrate how our model design allows for analysis of its individual components 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 to other examples of attention-based interpretability [25, 12, 7] which zoom in on interesting but instance-specific examples, our methods focus on ways to aggregate the patterns across the entire dataset – extracting generalizable insights about temporal dynamics.

7.1. Analyzing Variable Importance

We first quantify variable importance by analyzing the variable selection weights described in Sec. 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 10^{th}, 50^{th} and 90^{th} percentiles of each sampling distribution. As the Retail dataset

Table 3: Variable importance for the Retail dataset. The 10^{th}, 50^{th} and 90^{th} percentiles of the variable selection weights are shown, with values larger than 0.1 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 past inputs, past values of the target (i.e. log sales) are critical as expected, as forecasts are extrapolations of past observations. For future inputs, promotion periods and national holidays have the greatest influence on sales forecasts, in line with periods of increased customer spending.

	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

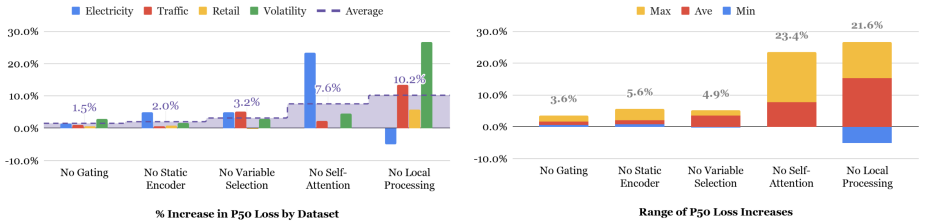
(a) Static Covariates

	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**

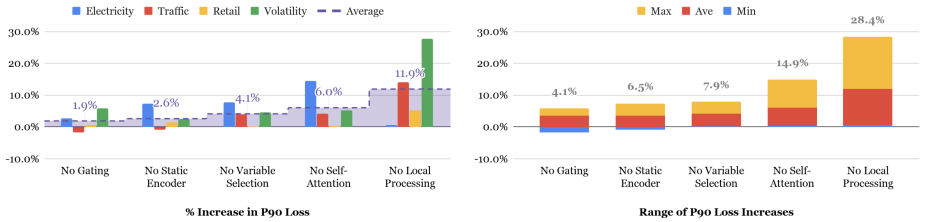
(b) Past Inputs

	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



(a) Changes in P50 losses across ablation tests



(b) Changes in P90 losses across ablation tests

Figure 3: 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.

contains the full set of available input types (i.e. static metadata, known inputs, observed inputs and the target), we present the results for its variable importance analysis in Table 3. We also note similar findings in other datasets, which are documented in Appendix B.1 for completeness. On the whole, the results show that the TFT extracts only a subset of key inputs that intuitively play a significant role in predictions. 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 [35] – such as the impact of a government's increase in public expenditure on the resultant growth in Gross National Product [36]. Seasonality models are also commonly used in econometrics to identify periodic patterns in a target-of-interest [37] and measure the length of each cycle. From a practical standpoint, model builders can use these insights to further improve the forecasting model – for instance by increasing the receptive field to incorporate more history if attention peaks are observed at the start of the lookback window, or by engineering features to directly incorporate seasonal effects. As such, using the attention weights present in the self-attention layer of the temporal fusion decoder, we present a method 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. $\tilde{A}(\boldsymbol{\phi}(t), \boldsymbol{\phi}(t))$. Multi-head attention outputs at each forecast horizon τ

(i.e. $\boldsymbol{\beta}(t,\tau)$) can then be described as an attention-weighted sum of lower level features at each position n:

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

where $\alpha(t,n,\tau)$ is the (τ,n)-th element of $\tilde{A}(\boldsymbol{\phi}(t),\boldsymbol{\phi}(t))$, and $\tilde{\boldsymbol{\theta}}(t,n)$ is a row of $\tilde{\boldsymbol{\Theta}}(t) = \boldsymbol{\Theta}(t)\boldsymbol{W}_V$. 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.

7.2. Visualizing Persistent Temporal Patterns

Attention weight patterns can be used to shed light on the most important past time steps that the TFT model bases its decisions on. In contrast to other traditional and machine learning time series methods, which rely on model-based specifications for seasonality and lag analysis, the TFT can learn such patterns from raw training data.

Fig. 4 shows the attention weight patterns across all our test datasets – with the upper graph plotting the mean along with the 10^{th}, 50^{th} and 90^{th} percentiles of the attention weights for one-step-ahead forecasts (i.e. $\alpha(t,1,\tau)$) over the test set, and the bottom graph plotting the average attention weights for various horizons (i.e. $\tau \in \{5,10,15,20\}$). We observe that the three datasets exhibit a seasonal pattern, with clear attention spikes at daily intervals observed for Electricity and Traffic, and a slightly weaker weekly patterns for Retail. For Retail, we also observe the decaying trend pattern, with the last few days dominating the importance.

No strong persistent patterns were observed for the Volatility – 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.

TFT learns these persistent temporal patterns from the raw training data without any human hard-coding. Such capability is expected to be very useful in building trust with human experts via sanity-checking. Model developers can also use these towards model improvements, e.g. via specific feature engineering or data collection.

7.3. Identifying Regimes & Significant Events

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 [38], with returns characteristics – such as volatility – being observed to change abruptly between regimes. As such, identifying such regime changes provides strong insights into the underlying problem which is useful for identification of the significant events.

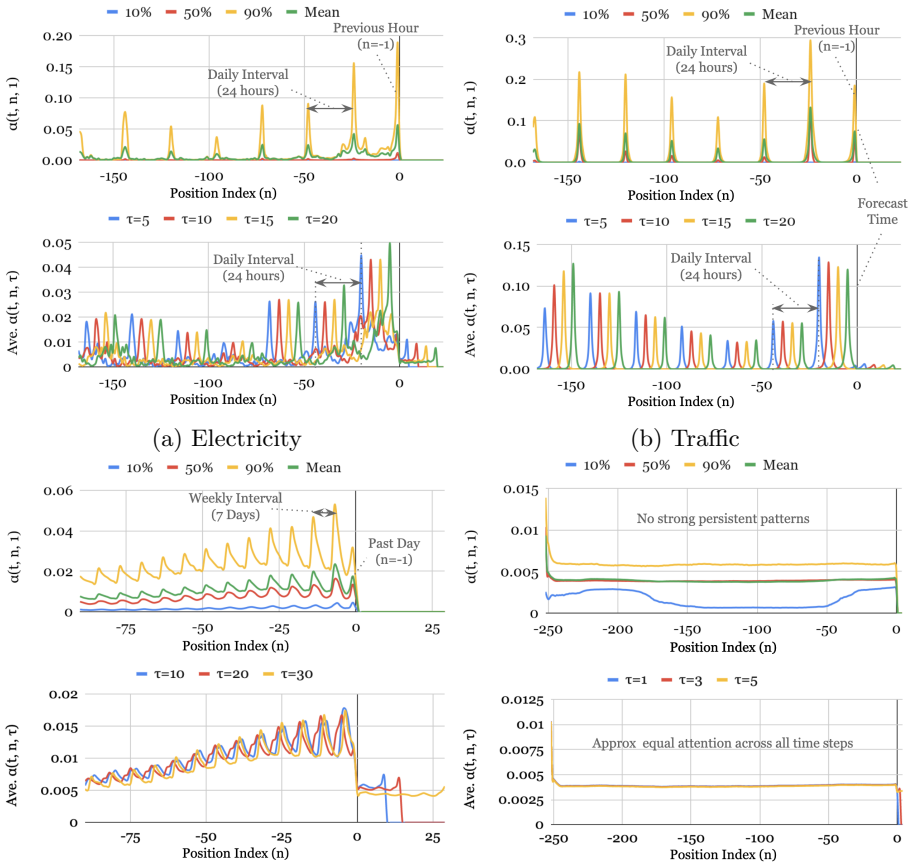


Figure 4: Persistent temporal patterns across datasets. Clear seasonality observed for the Electricity, Traffic and Retail datasets, but no strong persistent patterns seen in Volatility dataset. Upper plot – percentiles of attention weights for one-step-ahead forecast. Lower plot – average attention weights for forecast at various horizons.

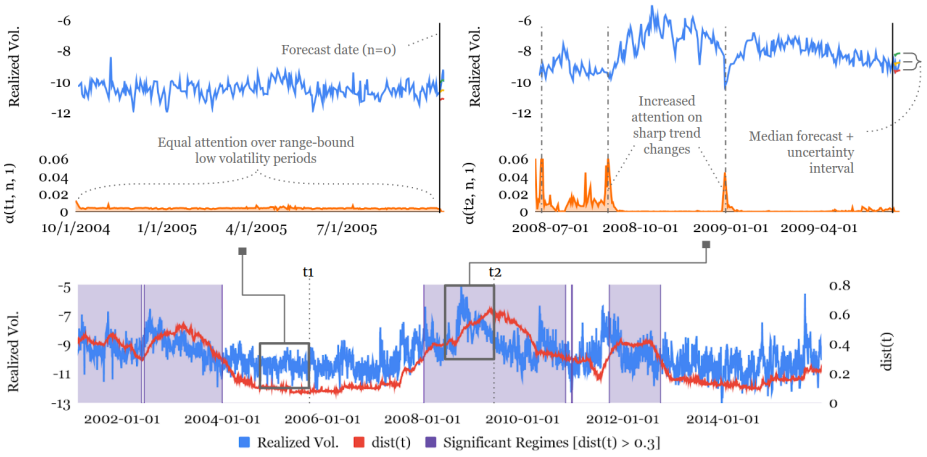


Figure 5: Regime identification for S&P 500 realized volatility. Significant deviations in attention patterns can be observed around periods of high volatility – corresponding to the peaks observed in dist(t). We use a threshold of dist(t) > 0.3 to denote significant regimes, as highlighted in purple. 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 as:

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

and then construct $\bar{\boldsymbol{\alpha}}(\tau) = [\bar{\alpha}(-k, \tau), \ldots, \bar{\alpha}(\tau_{max}, \tau)]^T$. To compare similarities between attention weight vectors, we use the distance metric proposed by [39]:

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

where $\rho(\boldsymbol{p}, \boldsymbol{q}) = \sum_j \sqrt{p_j q_j}$ is the Bhattacharya coefficient [40] measuring the overlap between discrete distributions – with p_j, q_j being elements of probability vectors $\boldsymbol{p}, \boldsymbol{q}$ 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:

$$\text{dist}(t) = \sum_{\tau=1}^{\tau_{max}} \kappa\big(\bar{\boldsymbol{\alpha}}(\tau), \boldsymbol{\alpha}(t, \tau)\big)/\tau_{max}, \tag{30}$$

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

Using the volatility dataset, we attempt to analyse regimes by applying our distance metric to the attention patterns for the S&P 500 index over our training period (2001 to 2015). Plotting dist(t) against the target (i.e. log realized volatility) in the bottom chart of Fig. 5, significant deviations in attention patterns can be observed around periods of high volatility (e.g. the 2008 financial crisis) – corresponding to the peaks observed in dist(t). From the plots, we can

see that TFT appears to alter its behaviour between regimes – placing equal attention across past inputs when volatility is low, while attending more to sharp trend changes during high volatility periods – suggesting differences in temporal dynamics learned in each of these cases.

8. Conclusions

We introduce TFT, a novel attention-based deep learning model for interpretable high-performance multi-horizon forecasting. To handle static covariates, a priori known inputs, and observed inputs effectively across wide range of multi-horizon forecasting datasets, TFT uses specialized components. 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 unnecessary parts of the network, (4) variable selection to pick relevant input features at each time step, and (5) quantile predictions to obtain output intervals across all prediction horizons. On a wide range of real-world tasks – on both simple datasets that contain only known inputs and complex datasets which encompass the full range of possible inputs – we show that TFT achieves state-of-the-art forecasting performance. Lastly, we investigate the general relationships learned by TFT through a series of interpretability use cases – proposing novel methods to use TFT to (i) analyze important variables for a given prediction problem, (ii) visualize persistent temporal relationships learned (e.g. seasonality), and (iii) identify significant regimes changes.

9. Acknowledgements

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

References

[1] J.-H. Böse, et al., Probabilistic demand forecasting at scale, Proc. VLDB Endow. 10 (12) (2017) 1694–1705.

[2] P. Courty, H. Li, Timing of seasonal sales, The Journal of Business 72 (4) (1999) 545–572.

[3] B. Lim, A. Alaa, M. van der Schaar, Forecasting treatment responses over time using recurrent marginal structural networks, in: NeurIPS, 2018.

[4] J. Zhang, K. Nawata, Multi-step prediction for influenza outbreak by an adjusted long short-term memory, Epidemiology and infection 146 (7) (2018).

[5] C. Capistran, C. Constandse, M. Ramos-Francia, Multi-horizon inflation forecasts using disaggregated data, Economic Modelling 27 (3) (2010) 666 – 677.

[6] S. S. Rangapuram, et al., Deep state space models for time series forecasting, in: NIPS, 2018.

[7] A. Alaa, M. van der Schaar, Attentive state-space modeling of disease progression, in: NIPS, 2019.

[8] S. Makridakis, E. Spiliotis, V. Assimakopoulos, The m4 competition: 100,000 time series and 61 forecasting methods, International Journal of Forecasting 36 (1) (2020) 54 – 74.

[9] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, DeepAR: Probabilistic forecasting with autoregressive recurrent networks, International Journal of Forecasting (2019).

[10] R. Wen, et al., A multi-horizon quantile recurrent forecaster, in: NIPS 2017 Time Series Workshop, 2017.

[11] C. Fan, et al., Multi-horizon time series forecasting with temporal attention learning, in: KDD, 2019.

[12] S. Li, et al., Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, in: NeurIPS, 2019.

[13] J. Koutník, K. Greff, F. Gomez, J. Schmidhuber, A clockwork rnn, in: ICML, 2014.

[14] D. Neil, et al., Phased lstm: Accelerating recurrent network training for long or event-based sequences, in: NIPS, 2016.

[15] M. Ribeiro, et al., "why should i trust you?" explaining the predictions of any classifier, in: KDD, 2016.

[16] S. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: NIPS, 2017.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: NIPS, 2017.

[18] S. B. Taieb, A. Sorjamaa, G. Bontempi, Multiple-output modeling for multi-step-ahead time series forecasting, Neurocomputing 73 (10) (2010) 1950 – 1957.

[19] M. Marcellino, J. Stock, M. Watson, A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series, Journal of Econometrics 135 (2006) 499–526.

[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.

[21] Y. Wang, et al., Deep factors for forecasting, in: ICML, 2019.

[22] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, Residual attention network for image classification, in: CVPR, 2017.

[23] S. O. Arik, T. Pfister, Tabnet: Attentive interpretable tabular learning (2019). arXiv:1908.07442.

[24] E. Choi, et al., Retain: An interpretable predictive model for healthcare using reverse time attention mechanism, in: NIPS, 2016.

[25] H. Song, et al., Attend and diagnose: Clinical time series analysis using attention models, 2018.

[26] J. Yoon, S. O. Arik, T. Pfister, Rl-lim: Reinforcement learning-based locally interpretable modeling (2019). arXiv:1909.12367.

[27] T. Guo, T. Lin, N. Antulov-Fantulin, Exploring interpretable LSTM neural networks over multi-variable data, in: ICML, 2019.

[28] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), in: ICLR, 2016.

[29] J. Lei Ba, J. R. Kiros, G. E. Hinton, Layer Normalization, arXiv:1607.06450 (Jul 2016). arXiv:1607.06450.

[30] Y. Dauphin, A. Fan, M. Auli, D. Grangier, Language modeling with gated convolutional networks, in: ICML, 2017.

[31] Y. Gal, Z. Ghahramani, A theoretically grounded application of dropout in recurrent neural networks, in: NIPS, 2016.

[32] H.-F. Yu, N. Rao, I. S. Dhillon, Temporal regularized matrix factorization for high-dimensional time series prediction, in: NIPS, 2016.

[33] C. Favorita, Corporacion favorita grocery sales forecasting competition (2018).
URL https://www.kaggle.com/c/favorita-grocery-sales-forecasting/

[34] G. Heber, A. Lunde, N. Shephard, K. K. Sheppard, Oxford-man institute's realized library (2009).
URL https://realized.oxford-man.ox.ac.uk/

[35] S. Du, G. Song, L. Han, H. Hong, Temporal causal inference with time lag, Neural Computation 30 (1) (2018) 271–291.

[36] B. Baltagi, Distributed Lags and Dynamic Models, 2008, pp. 129–145.

[37] S. Hylleberg (Ed.), Modelling Seasonality, Oxford University Press, 1992.

[38] A. Ang, A. Timmermann, Regime changes and financial markets, Annual Review of Financial Economics 4 (1) (2012) 313–337.

[39] D. Comaniciu, V. Ramesh, P. Meer, Kernel-based object tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (5) (2003) 564–577.

[40] T. Kailath, The divergence and bhattacharyya distance measures in signal selection, IEEE Transactions on Communication Technology 15 (1) (1967) 52–60.

[41] E. Giovanis, The turn-of-the-month-effect: Evidence from periodic generalized autoregressive conditional heteroskedasticity (pgarch) model, International Journal of Economic Sciences and Applied Research 7 (2014) 43–61.

APPENDIX

Appendix A. Dataset and Training Details

We provide all the sufficient information on feature pre-processing and train/test splits to ensure reproducibility of our results.

Electricity: Per [9], 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 [9, 32]. 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 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 [9], 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 and a time index as real-valued inputs, and the entity identifier as a categorical variable.

Retail: We treat each product number-store number pair as a separate entity, with over 135k entities in total. 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 from the Kaggle competition. Data is resampled at regular daily intervals, imputing any missing days using the last available observation. We include an additional 'open' flag to denote whether data is present on a given day. We group national, regional, and local holidays into separate variables. We 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 and the rest to be categorical.

Volatility: We use the data 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. 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 the training set. We treat all date-related variables (i.e. day-of-week, day-of-month, week-of-year, and month) and the region as categorical inputs. A log transformation is applied to the target, and all inputs are z-score normalized across all entities.

Appendix B. Interpretability Results

Apart from Sec. 7, which highlights our most prominent findings, we present the remaining results here for completeness.

Appendix B.1. Variable Importance

Table B.4 shows the variable importance scores for the remaining Electricity, Traffic and Volatility datasets. As these datasets only have one static input, the network allocates full weight to the entity identifier for Electricity and Traffic, along with the region input for Volatility. We also observe two types of important 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 observed to play a significant role in future inputs – potentially reflecting turn-of-month effects [41].

Table B.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.

	10%	50%	90%
Static			
ID	**1.000**	**1.000**	**1.000**
Past			
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			
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			
ID	**1.000**	**1.000**	**1.000**
Past			
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			
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			
Region	**1.000**	**1.000**	**1.000**
Past			
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			
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