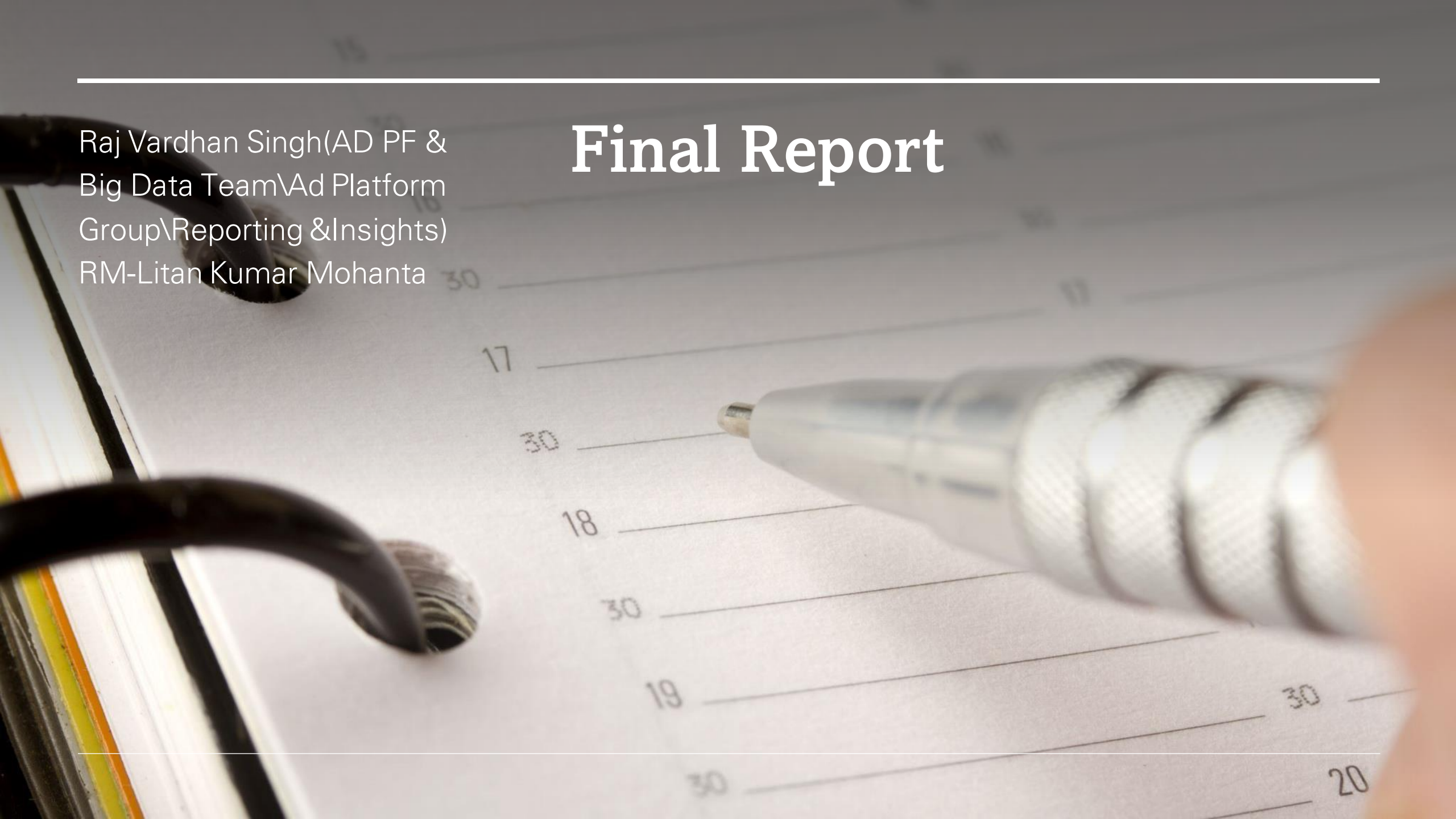


Final Report

Raj Vardhan Singh(AD PF & Big Data Team\Ad Platform Group\Reporting &Insights)
RM-Litan Kumar Mohanta

Problem Statement

Forecasting future values across multiple time steps is a challenging task with significant implications. However, existing forecasting methods encounter several obstacles, including a lack of explainability, difficulties in handling complex interactions between known and unknown inputs, and limitations in incorporating time order within traditional explainable approaches. These issues hamper the accuracy, interpretability, and usefulness of the forecasts, ultimately impacting critical decision-making processes and potential financial gains.

The primary objective of this research/problem is to develop an explainable multi-step forecasting model capable of handling complex time series data, while prioritizing accuracy, interpretability, and financial significance. By overcoming the challenges associated with existing methods, the proposed solution will empower decision-makers with reliable and actionable forecasts across various domains.

The Data Model

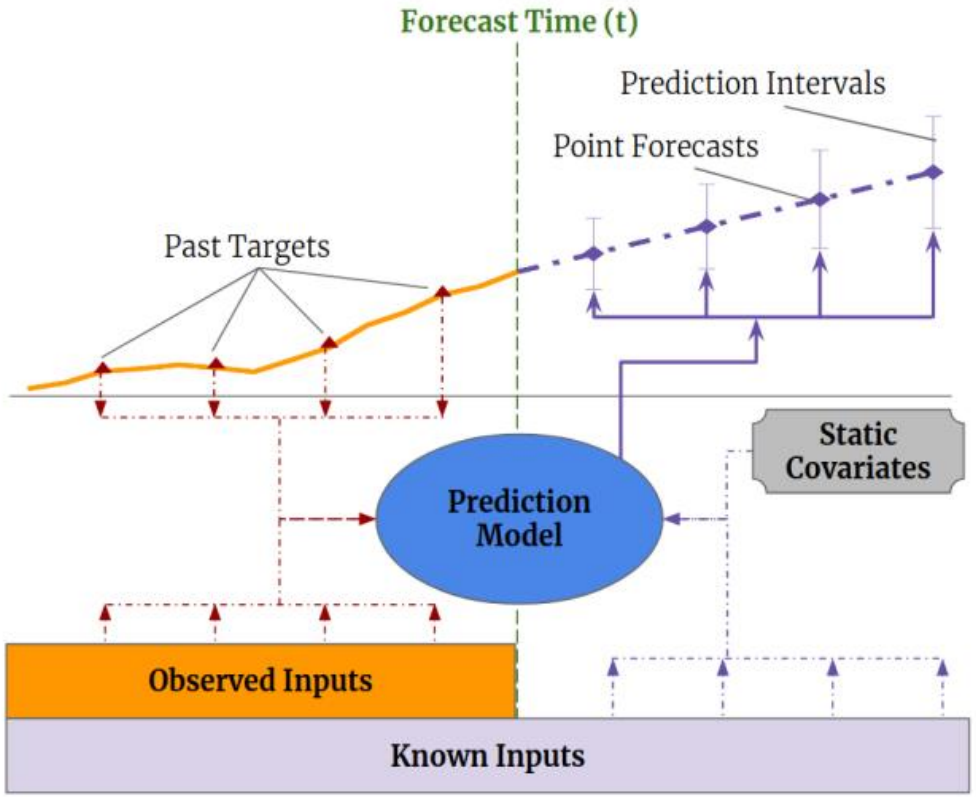


Figure 1: Illustration of multi-horizon forecasting with static covariates, past-observed and apriori-known future time-dependent inputs.

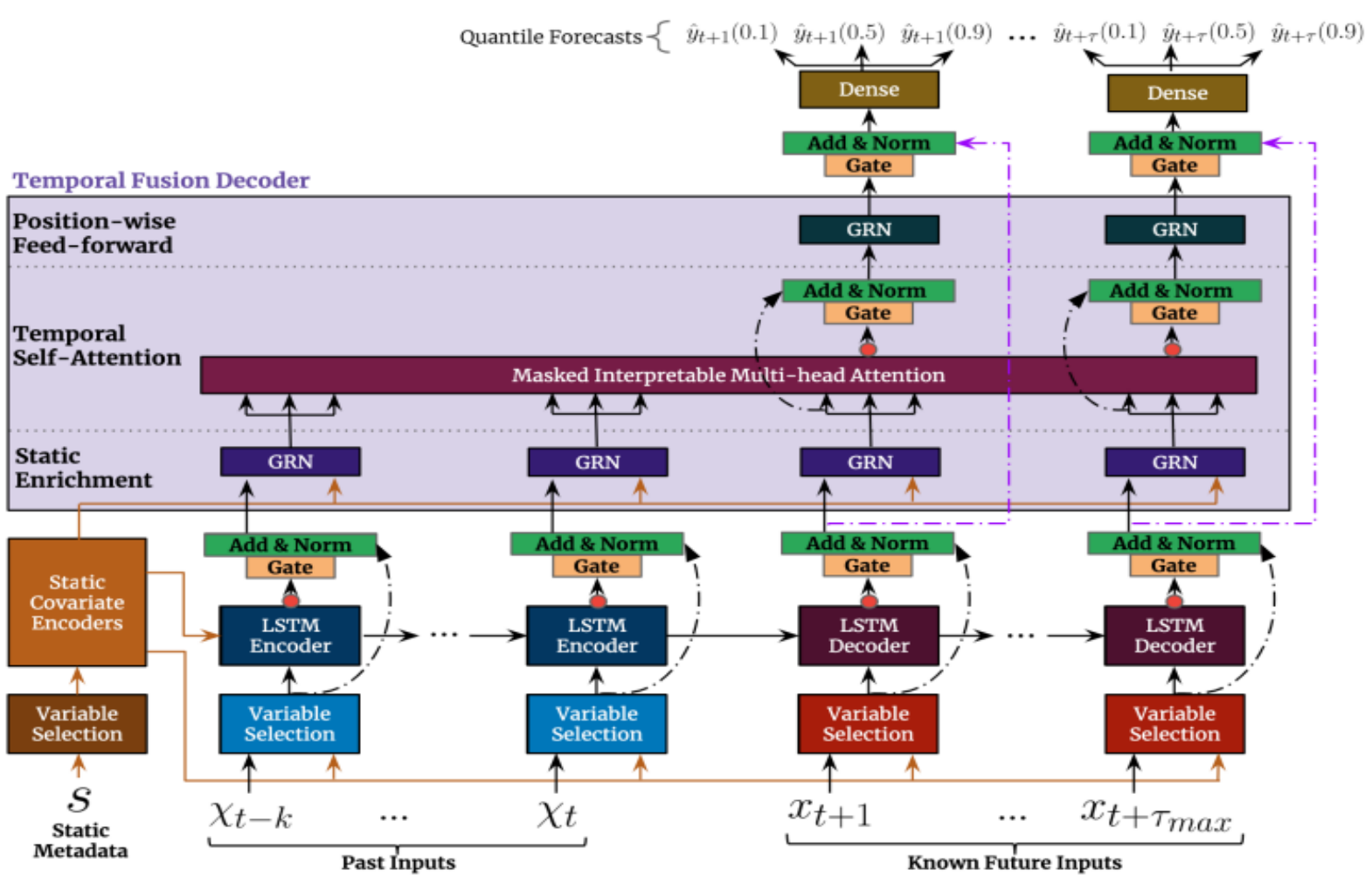
Literature Review

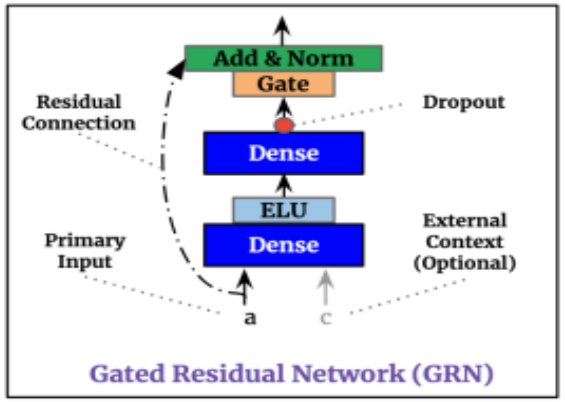
- Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting
- DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks

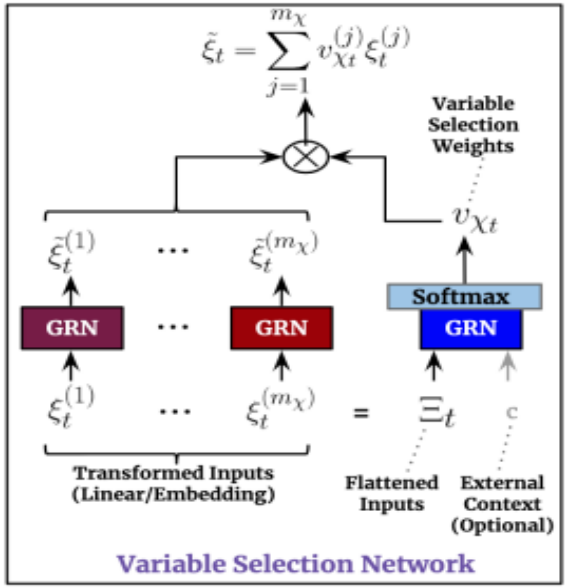
Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting

- Static covariates encoder
- Gating mechanism to allow less complex models
- A temporal self-attention mechanism
  - 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
- Explainablity gives :
  - Globally important variables
  - Persistent temporal patterns
  - Significant events

Model Architecture







Data set - FAVORITA

- The Favorita dataset, also known as the "Corporación Favorita Grocery Sales Forecasting" dataset, is a widely used dataset in the field of time series forecasting. It was part of a Kaggle competition organized by Favorita, a large Ecuadorian supermarket chain, to predict the sales of various products across their stores.
- The Favorita dataset is a retail dataset containing historical sales records from January 2013 to August 2017, with information about store locations, item types, dates, and promotions. It offers opportunities for time series forecasting, analysis of hierarchical structures, and exploration of external factors affecting sales.

Favorita Data Preprocessing

- **Data Cleaning and Filtering**: Clean and filter the data based on specific date ranges. The start_date and end_date variables are used to filter the temporal data, ensuring that only the desired date range is considered for further analysis.
- **Data Integration and Joining**: Integrate and join multiple datasets using common columns such as store number (store_nbr) and item number (item_nbr). The store_info dataset and items dataset are joined with the temporal data to enrich it with additional store and item information.
- **Handling Missing Values**: Handle missing values in the oil dataset by filling them using the forward fill method. The missing values in the oil column are filled with the last available value for each corresponding date.

- **Resampling and Time Series Transformation**: Resample the temporal data to a regular grid using daily frequency . This ensures that the data is evenly spaced and aligned, making it suitable for time series analysis. Additionally, we have done data transformations such as taking the logarithm of the sales value to normalize the data.
- **Feature Engineering and Enrichment**: In addition, we have augmented the temporal data with additional characteristics derived from the existing data. It extracts various date-related features such as day of the week, day of the month, and month. It also incorporates holiday information by merging the holidays dataset and creating new columns (national_hol, regional_hol, local_hol) to indicate the presence of holidays on specific dates.

Problem encountered in Data Preprocessing

The encountered problem during data preprocessing was the large size of the dataset, which resulted in increased memory usage. Since the temporal data was filtered for a specific date range, merging and joining additional data frames such as oil, transactions, holidays, stores, and items with the temporal data further increased memory consumption. To address this issue, several approaches were attempted.

Firstly, unnecessary data frames were deleted to free up memory. Additionally, the gc.collect() function was utilized to explicitly release memory occupied by unused objects. Filtering the temporal data frame based on specific date range. Furthermore, to reduce the dataset size, the number of stores was reduced by 50 percent. By implementing these strategies, it was possible to mitigate the memory limitations and successfully prepare the data for training.

TFT-Training

- **Training the TFT model**: trained the TFT model using the specified parameters and dataset. This involved loading and splitting the data, setting up the hyperparameter manager, and performing the training process. The model was trained for a specific number of epochs, and the best model based on validation loss was selected and saved.
- **Error analysis - P50 loss and P90 loss**: After training the TFT model, you conducted error analysis to evaluate the model's performance. You calculated the P50 loss and P90 loss for the test data. These losses measure the normalized quantile loss between the model's predictions (P50 and P90 forecasts) and the actual targets. The P50 loss represents the loss for the median forecast, while the P90 loss represents the loss for the 90th percentile forecast.
- **Graph plotting for forecasts**: In addition to error analysis, also plotted graphs for the forecasts generated by the TFT model. These graphs visually represent the model's predictions compared to the actual data. The forecasts include P50 (median) and P90 (90th percentile) forecasts.

During the model training phase, we plan to run the model for approximately 5 to 10 epochs. To monitor the model's performance and determine the optimal stopping point, we will utilize the validation loss metric. After each epoch, we will save the model to capture the progress and allow for future inference or continuation of training. To leverage the GPU resources available on Kaggle, we will ensure that the model training process is configured to utilize the GPU for accelerated computations. Additionally, we will explore and learn about data loaders in TensorFlow. Data loaders play a crucial role in efficiently loading and preprocessing data during training. By understanding the concepts and functionality of data loaders, we can optimize the data input pipeline and enhance the training process. By combining these approaches, we aim to train the model effectively, leverage the available GPU resources, and enhance the data loading process using data loaders in TensorFlow

- Trained the TFT model by optimizing its parameters to minimize the loss function during training. It then evaluates the model's performance on validation and test data.
- Hyperparameter optimization is performed to find the best set of hyperparameters for the TFT model. The code iterates over different combinations of hyperparameters, trains and evaluates the model for each combination, and selects the one with the lowest validation loss.
- The results of the training and evaluation are stored in separate CSV files. These files include the predicted values (p50 and p90 forecasts) and the actual target values for the test data. Storing the results allows for further analysis, comparison, and visualization of the model's performance.

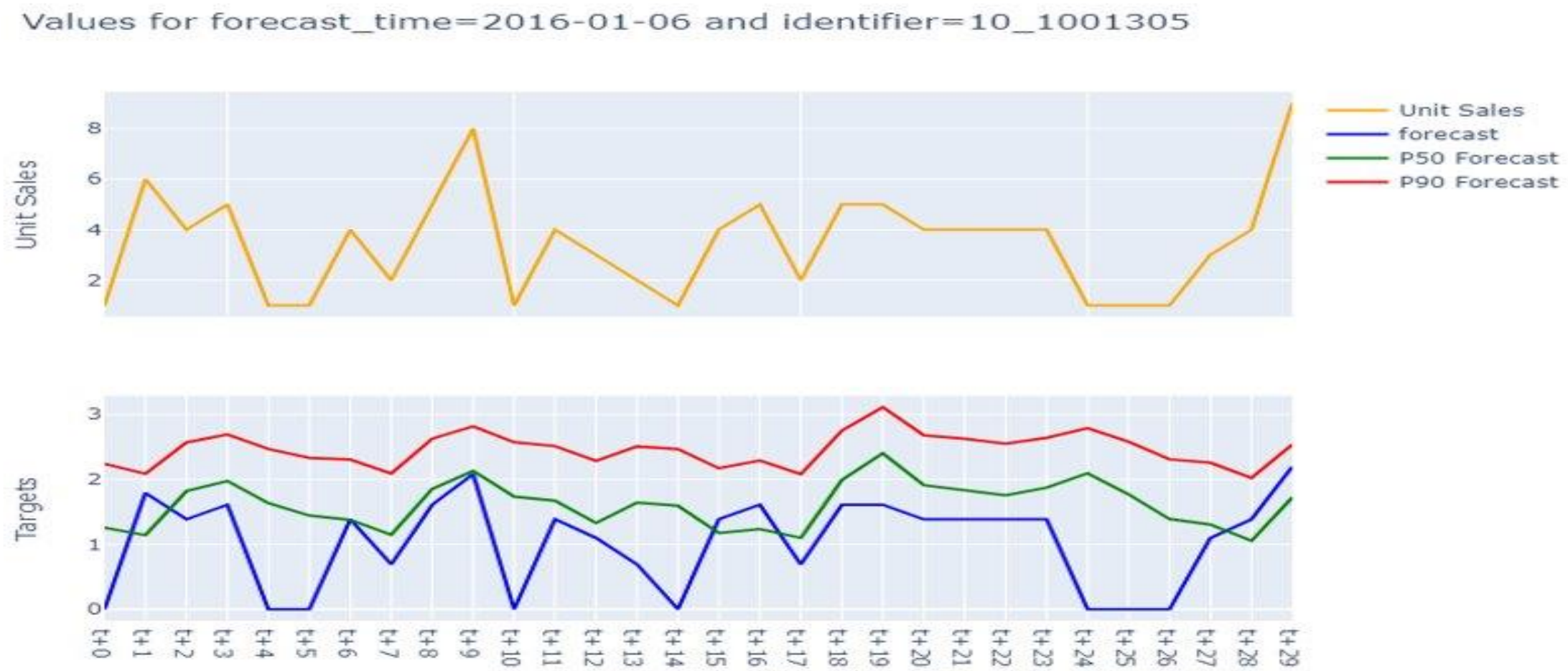
Parameters used in the model

Parameters	
dropout_rate	0.1
hidden_layer_size	100
Learning_rate	0.001
Max_gradient_norm	100
Mini_batch_size	128
Num_heads	4
Stack_size	1
Total_time_steps	120
No_encoder_steps	90
No_epochs	10
Early_stopping_patience	5

P50,P90 Loss

Normalized Quantile Loss	
P50	0.64319114653
P90	0.33425911089

Plot comparing Ground truth and Forecast



DEEP AR: PROBABILISTIC FORECASTING WITH AUTOREGRESSIVE RECURRENT NETWORKS

- Proposes an RNN architecture for probabilistic forecasting
  - Incorporating a negative Binomial likelihood for count data
  - Special treatment for the case when the magnitudes of the time series vary widely
- Demonstrate this model produces accurate probabilistic forecasts
  - Across a range of input characteristics, on several real-world data sets
  - Show effectively addressing probabilistic forecasting problem
  - Which contrasts with common belief in the field

DEEP-AR STRUCTURE

Training Encoder

Prediction Decoder

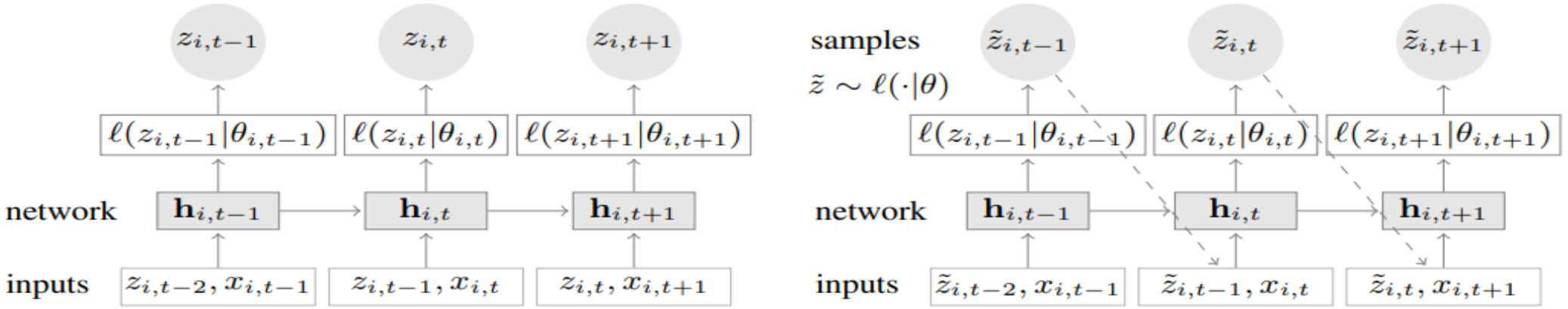
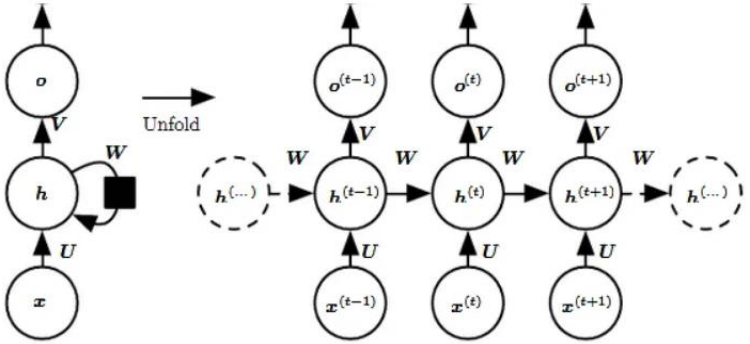


Figure 2: Summary of the model. Training (left): At each time step t, the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, as well as the previous network output $\mathbf{h}_{i,t-1}$. The network output $\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta)$ is then used to compute the parameters $\theta_{i,t} = \theta(\mathbf{h}_{i,t}, \Theta)$ of the likelihood $\ell(z|\theta)$, which is used for training the model parameters. For prediction, the history of the time series $z_{i,t}$ is fed in for $t < t_0$, then in the prediction range (right) for $t \geq t_0$ a sample $\hat{z}_{i,t} \sim \ell(\cdot|\theta_{i,t})$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + T$ generating one sample trace. Repeating this prediction process yields many traces representing the joint predicted distribution.

RNN

- A recurrent neural network is a neural network that is specialized for processing a sequence of data **x(t)= x(1), . . . , x(τ)** with the time step index ***t*** ranging from **1 to τ**
- Vanishing gradient problem->long range dependency ->LSTM

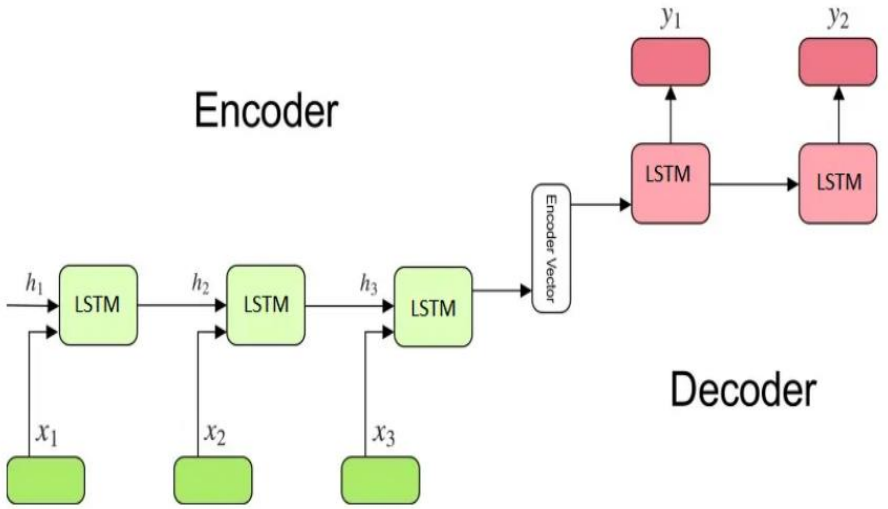


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

- **Input**: x(t) is taken as the input to the network at time step t.
- **Hidden state**: *h(t)* represents a hidden state at time t and acts as "memory" of the network. *h(t)* is calculated based on the current input and the previous time step's hidden state: **h(t)** = f(*U* **x(t)** + *W* **h(t−1)**). The function f is taken to be a non-linear transformation such as *tanh*, *ReLU.*
- **Weights**: The RNN has input-to-hidden connections U, hidden-to-hidden recurrent connections W, and hidden-to-output connections V, all of which are shared throughout time.
- **Output**: *o(t)* illustrates the output of the network.

ENCODER DECODER

- Both encoder and the decoder are typically LSTM models (or sometimes GRU models)
- Encoder reads the input sequence and summarizes the information in something called as the internal state vectors (in case of LSTM these are called as the hidden state and cell state vectors). We discard the outputs of the encoder and only preserve the internal states
- Decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM. Using these initial states, decoder starts generating the output sequence.
- The decoder behaves a bit differently during the training and inference procedure. During the training, we use a technique call teacher forcing which helps to train the decoder faster. During inference, the input to the decoder at each time step is the output from the previous time step.
- the encoder summarizes the input sequence into state vectors (sometimes also called as Thought vectors), which are then fed to the decoder which starts generating the output sequence given the Thought vectors. The decoder is just a language model conditioned on the initial states.



NOTATIONS

- Denoting the value of time series *i* at time *t* by $z_{i,t}$, our goal is to model the conditional distribution $P\left(z_{i,t_0,T} \middle| z_{i,1:t_0-1} ; x_{i,1:T} \right)$
- t_0 denotes the time point from which we assume $z_{i,t}$ to be unknown at prediction time
- Will refer to time ranges $\left[1, t_0 - 1\right]$ and $\left[t_0, T\right]$ as the **conditioning range and prediction range**
- During training, both ranges must lie in the past so that the $z_{i,t}$ are observed, but during prediction $z_{i,t}$ is only available in the conditioning range
- $x_{\mathrm{i,1:T}}$ **are covariates** that are assumed to be known for all time points

FORMULATION

- Model distribution consists of a product of likelihood factors

$$Q_\Theta(\mathbf{z}_{i,t_0:T}|\mathbf{z}_{i,1:t_0-1},\mathbf{x}_{i,1:T}) = \prod_{t=t_0}^{T} Q_\Theta(z_{i,t}|\mathbf{z}_{i,1:t-1},\mathbf{x}_{i,1:T}) = \prod_{t=t_0}^{T} \ell(z_{i,t}|\theta(\mathbf{h}_{i,t},\Theta))$$

$$\mathbf{h}_{i,t} = h\left(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta\right) \quad \text{LSTM output} \qquad (1)$$

- The model is autoregressive, in the sense that it consumes the observation at the last time step $z_{i,t-1}$ as an input, as well as recurrent, i.e., the previous output of the network $h_{i,t-1}$ is fed back as an input at the next time step
- Information about the observations in the conditioning range z_{i,t_0-1} is transferred to the prediction range through the initial state h_{i,t_0-1}.

PREDICTION PROCESS

Given the model parameters Θ, we can directly obtain joint samples $\tilde{\mathbf{z}}_{i,t_0:T} \sim Q_\Theta\left(z_{i,t_0:T} \middle| \mathbf{z}_{i,t_0-1}; \mathbf{x}_{i,1:T}\right)$ through ancestral sampling: First, we obtain h_{i,t_0-1} by computing $\mathbf{h}_{i,t} = h\left(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta\right)$ for $t = 1, \ldots\ldots, t_0$

For $t = t_0, t_0 + 1, \ldots\ldots, T$ we sample $\tilde{z}_{i,t} \sim \ell\left(\cdot \middle| \theta\left(\tilde{h}_{i,t}, \Theta\right)\right)$ where $\tilde{\boldsymbol{h}}_{i,t} = h\left(\boldsymbol{h}_{i,t-1}, \tilde{\mathbf{z}}_{i,t-1}, \boldsymbol{x}_{i,t}, \boldsymbol{\Theta}\right)$ initialized with $\tilde{\boldsymbol{h}}_{i,t_0-1} = \boldsymbol{h}_{i,t_0-1}$ and $\tilde{z}_{i,t_0-1} = z_{i,t_0-1}$. Samples from the model obtained in this way can then be used to compute quantities of interest, e.g., quantiles of the distribution of the sum of values for some time range in the future.

LIKELIHOOD MODEL

- The likelihood $\ell(z \mid \theta)$ determines the "noise model", and should be chosen to match the statistical properties of the data

$$\ell_{\mathrm{G}}(z|\mu,\sigma) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp(-(z-\mu)^2/(2\sigma^2))$$
$$\mu(\mathbf{h}_{i,t}) = \mathbf{w}_\mu^T \mathbf{h}_{i,t} + b_\mu \quad \text{and} \quad \sigma(\mathbf{h}_{i,t}) = \log(1+\exp(\mathbf{w}_\sigma^T \mathbf{h}_{i,t} + b_\sigma))$$

$$\ell_{\mathrm{NB}}(z|\mu,\alpha) = \frac{\Gamma(z+\frac{1}{\alpha})}{\Gamma(z+1)\Gamma(\frac{1}{\alpha})} \left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{z}$$
$$\mu(\mathbf{h}_{i,t}) = \log(1+\exp(\mathbf{w}_\mu^T \mathbf{h}_{i,t} + b_\mu)) \quad \text{and} \quad \alpha(\mathbf{h}_{i,t}) = \log(1+\exp(\mathbf{w}_\alpha^T \mathbf{h}_{i,t} + b_\alpha))$$

For real valued data:
We parametrize the Gaussian likelihood using its mean and standard deviation, θ = (μ, σ), where the mean is given by an affine function of the network output, and the standard deviation is obtained by applying an affine transformation followed by a SoftPlus activation to ensure σ > 0:
For positive count data:
The negative binomial distribution is a commonly used choice We parameterize the negative binomial distribution by its mean μ ∈ R + and a shape parameter α ∈ R +

TRAINING

- maximizing the log-likelihood $\mathcal{L} = \sum_{i=1}^{N} \sum_{t=t_0}^{T} \log \ell(z_{i,t}|\theta(\mathbf{h}_{i,t}))\,.$
- can be optimized directly via stochastic gradient descent by computing gradients with respect to Θ.

SCALE HANDLING

Applying the model to data that exhibits a power-law of scales presents two challenges:

- Firstly, due to the autoregressive nature of the model, both the autoregressive input $z_{i,t-1}$ as well as the output of the network (e.g., μ) directly scale with the observations $z_{i,t}$, but the nonlinearities of the network in between have a limited operating range. Without further modifications, the network thus has to learn to scale the input to an appropriate range in the input layer, and then to invert this scaling at the output.
  We address this issue by dividing the autoregressive inputs $z_{i,t}$ (or $\tilde{z}_{i,t}$) by an item-dependent scale factor ν_i, and conversely multiplying the scale-dependent likelihood parameters by the same factor

- Secondly, due to the imbalance in the data, a stochastic optimization procedure that picks training instances uniformly at random will visit the small number time series with a large scale very infrequently, which result in underfitting those time series.
  To counteract this effect, we sample the examples non-uniformly during training.

KEY ADVANTAGES

- **Minimal manual feature engineering** is needed to capture complex, group-dependent behavior.
- **Probabilistic forecasts** in the form of Monte Carlo samples that can be used to compute consistent quantile estimates for all sub-ranges in the prediction horizon.
- By learning from similar items, our method is able to provide forecasts for items with **little or no history** at all.
- Our approach does **not assume Gaussian noise**, but can incorporate a wide range of likelihood functions, allowing the user to choose one that is appropriate for the statistical properties of the data.

Conclusion

- Performance Comparison: We used TFT and DeepAR models to forecast multi-horizon time series. We thoroughly compared both models' dataset performance.
- Accuracy and Robustness: The TFT and DeepAR models predicted accurately across several time horizons. Both models accurately captured complicated time series data patterns and fluctuations.
- Model Selection: TFT or DeepAR depending on requirements and considerations. Interpretability and attention mechanisms make the TFT model useful for understanding time step and feature relevance. However, the DeepAR model captures interdependence over time and generates probabilistic forecasts well.
- In conclusion, TFT and DeepAR models are effective in multi-horizon time series forecasting. The results reveal each model's strengths and weaknesses, helping practitioners choose the best method for their needs and dataset. Advanced forecasting models will help improve time series forecasting.

Thank you

I would like to express my sincere gratitude to Samsung for providing me with the opportunity to work on this project during my internship. I am thankful to my Reporting Manager, Litan Kumar Mohanta, for guiding me throughout the project and providing valuable insights and feedback. I am also grateful to my Mentors, Jasmeen Patel and Ashish Tolambiya, for their continuous support and guidance.

During this internship, I had the privilege of working on multi-horizon time series forecasting using the TFT and DeepAR models. This project allowed me to gain a deeper understanding of time series forecasting techniques and their application in real-world scenarios. Through the guidance of my mentors and the resources provided by Samsung, I was able to successfully implement and compare the performance of the TFT and DeepAR models.