Transformer Meets Time-Series

Temporal Fusion Transformer zur Vorhersage von Wärmebedarfen bei Iqony

Our Journey

1. A brief introduction to Iqony
2. Getting familar with the Use Case
3. Let´s talk about the initial situation and ARIMA models
4. Let´s talk about Recurrent Neural Network
5. Let´s talk about Temporal Fusion Transformers
6. Summary & Outlook
7. Discussion

AN ADVENTURE?

LET´S ALPACA OUR BAGS!

Brief intoduction to Iqony

Operator of multiple decentralized energy systems:

einer der größten **Fernwärme-versorger** und **Contracting-Anbieter,** besonders für Industrie und große Liegenschaften in Deutschland

Vorreiter in effizienter Erzeugung mit Erfahrung mit unterschiedlichen Erzeugungstechnologien

Betreiber von über 100 dezentralen Anlagen (Strom, Wärme, Kälte, Druckluft, Dampf), auch in Verbindung mit Erneuerbaren Energien für die Industrie und Kommunen

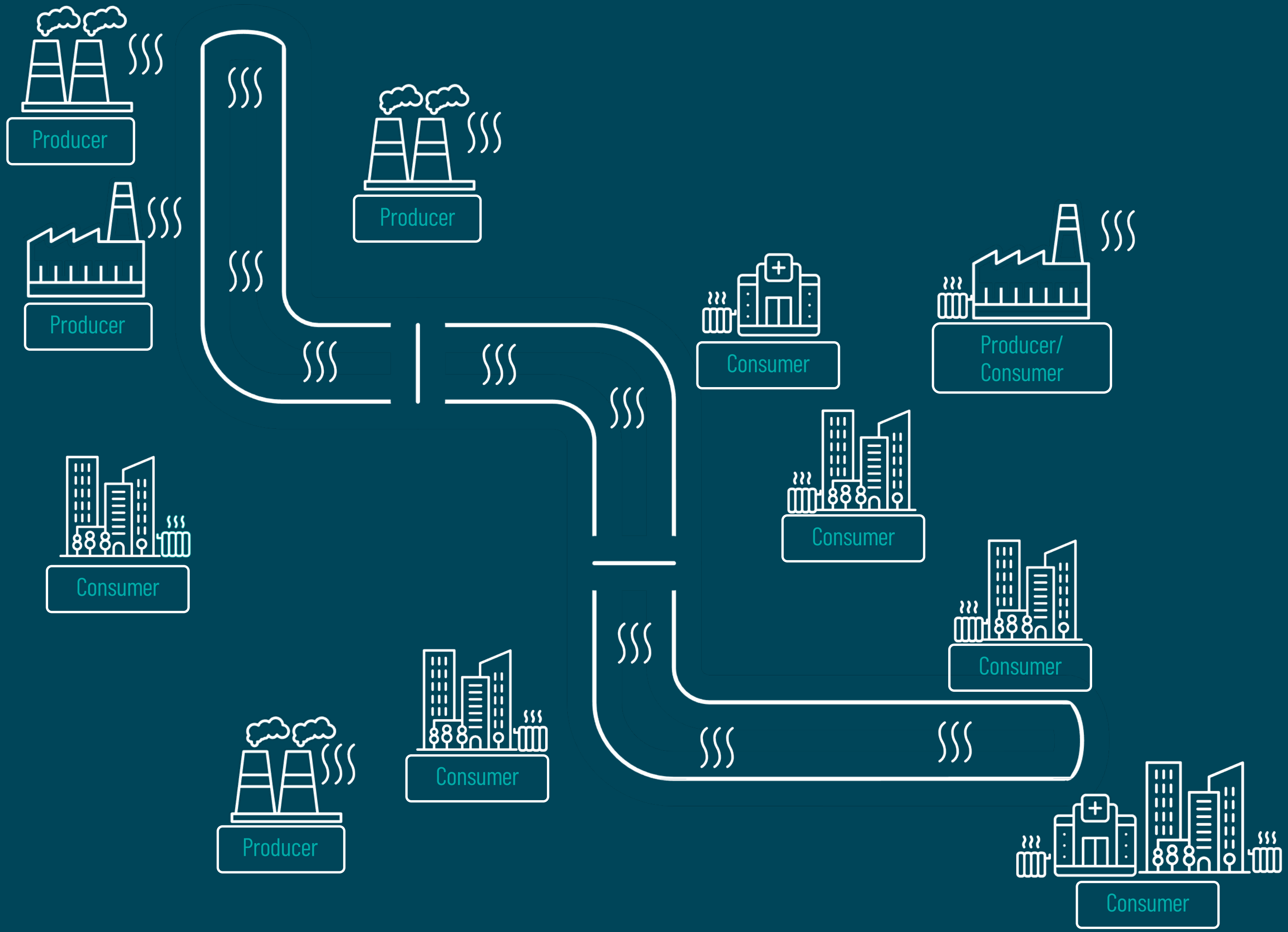
Betreiber von Windenergie-anlagen in Deutschland, Frankreich und Polen

Betreiber von 39 Wärme-versorgungen inkl. der Fernwärmeschiene Saar

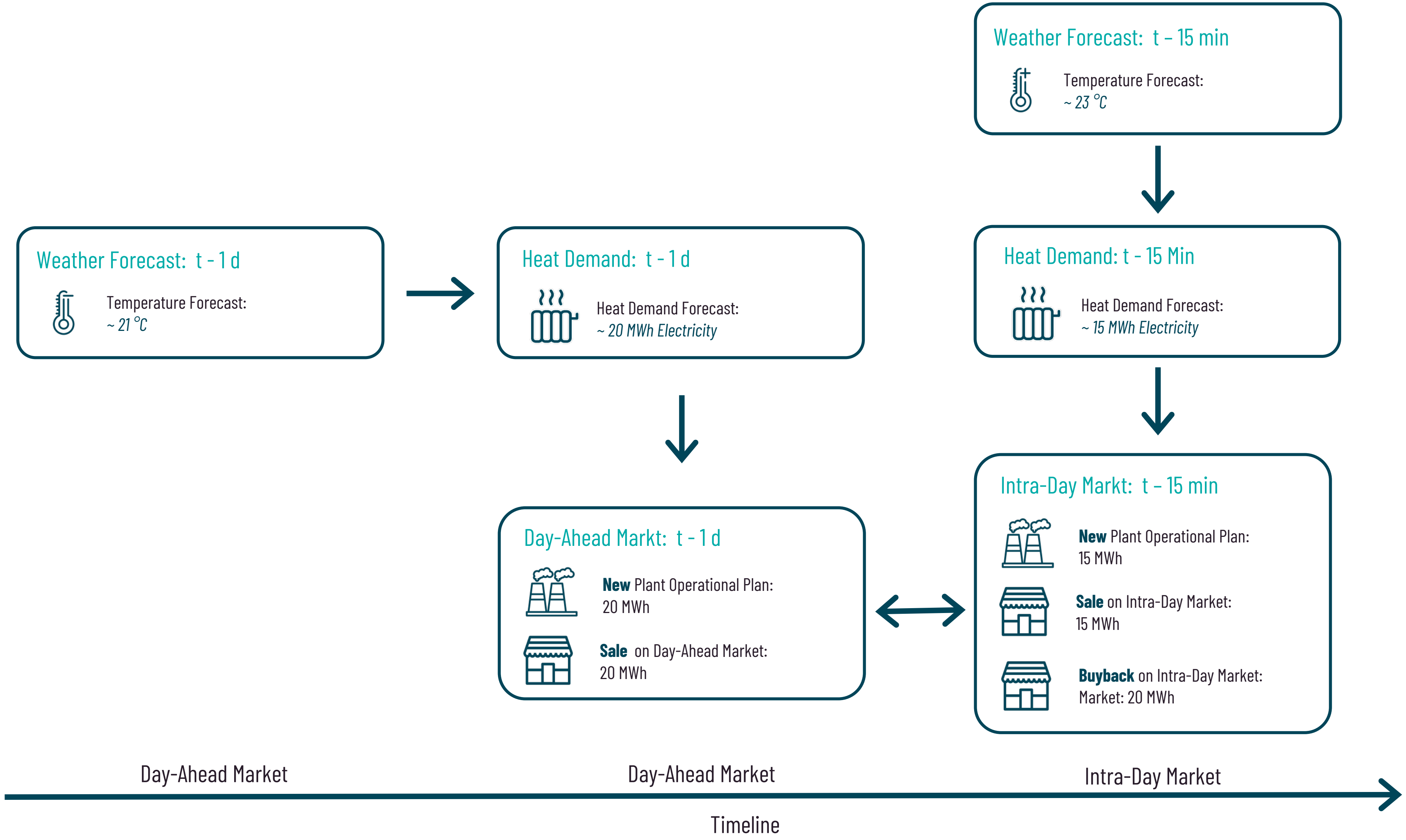
Partner bei mehr als 10 kommunalen Wärme-versorgungen in Deutschland

Vorläufige Kennzahlen 2020	MW_{el}	MW_{th}
Biomasse › seit 2002	55	148
Grubengas › seit 1908	166	113
Geothermie › seit 1994	-	145
Dezentrale Anlagen › seit 1961	77	774
Wind › seit 2010	231	-
Gesamt	**528**	**1.180**

Use Case: District Heating Network Saar (FVS)



Use Case: Trading on the Energy Market



Let´s talk about the initial situation and ARIMA Models

scieneers
DRIVEN BY DATA

Use Case: Initial Situation FVS

Complexity

Due to a network of consumers and producers, the Saar district heating system is to be regarded as an extremely complex product

Existing Forecasting Tool

Due to its complexity, an existing forecasting tool provides only inadequate forecasts for the Saar district heating network.

Data Science

Insights will be gained from data to provide sufficient understanding of the complex relationships that modulate heat demand

In-house Development

On the basis of knowledge gained, own model development and provision will be aimed at in order to improve the prediction accuracy

Auto Regressive Integrated Moving Average

Description
Model for the description and analysis of deterministic and stochastic time series.

How it works
Combination of an autoregressive and moving average model. The dependent variable thus depends on the own lags as well as past forecast errors

Speciality
In an ARIMA model, the time series was differentiated at least once with the goal of realizing a stationary time series from a non-stationary time series

Usage
Short-term predictions of a temporal course based on past or historical time series

scieneers
DRIVEN BY DATA

ARIMA Model for Heat Demand Forecasting

Historical Time-Series

Time series of historical energy demand

- Specific for each District Heating System
- Sampling interval of 15 min

Exogenous Regressors

Calculation of Fourier terms assuming daily periodicity:

- Temperature forecast
- Weekdays and holidays

Fitting of ARIMA-Model

Auto-ARIMA Model for heat demand forecasts over a horizon of 4 days:

- Intervals of 15 minutes each
- No seasonality
- No stationarity
- Optimization according to Powell

Forecasts

Heat demand forecasts for the next 4 days

Potential parameters to enrich data basis

Weather

- Temperature (API)
- Air Pressure (API)
- Humidity (API)
- Wind Speed (API)
- Wind Direction (API)
- Sun Hours (API)

Consumer

- Production Planning (SNE)
- Number of Households (SNE)
- Investment in Solar Systems (API)
- Number of Solar Systems (API)
- Conclusion or termination of contracts

Demographics

- Population (API)
- Average Income (API)
- Average Age (API)

Producer

- Price per kWh (SNE)
- Heat Energy Minimum (SNE)
- Production Planning

Datetime

- Weekday
- Month
- Quarter
- Public Holidays (API)
- School Holidays (API)

Others

- Covid-19 Incidence (API)

Let´s do a little recap on the ARIMA Model

Findings

- Other weather parameters tend to have no further added value
- Production planning has little or no influence on ARIMA Model
- Other tested models benefits enormously from production planning
- Temperature forecast error has significant influence on heat demand forecast
- Severe Artifacts found in data of the historical heat demand

Ideas and opportunities

- Temperature forecast error as an additional feature
- Production planning as a promising feature

Problem of ARIMA Models

- ARIMA model shows weaknesses for complex, multivariate problems leading to insufficient forecast for such cases

Let´s talk about Recurrent Neural Networks

Sequence2Sequence Model

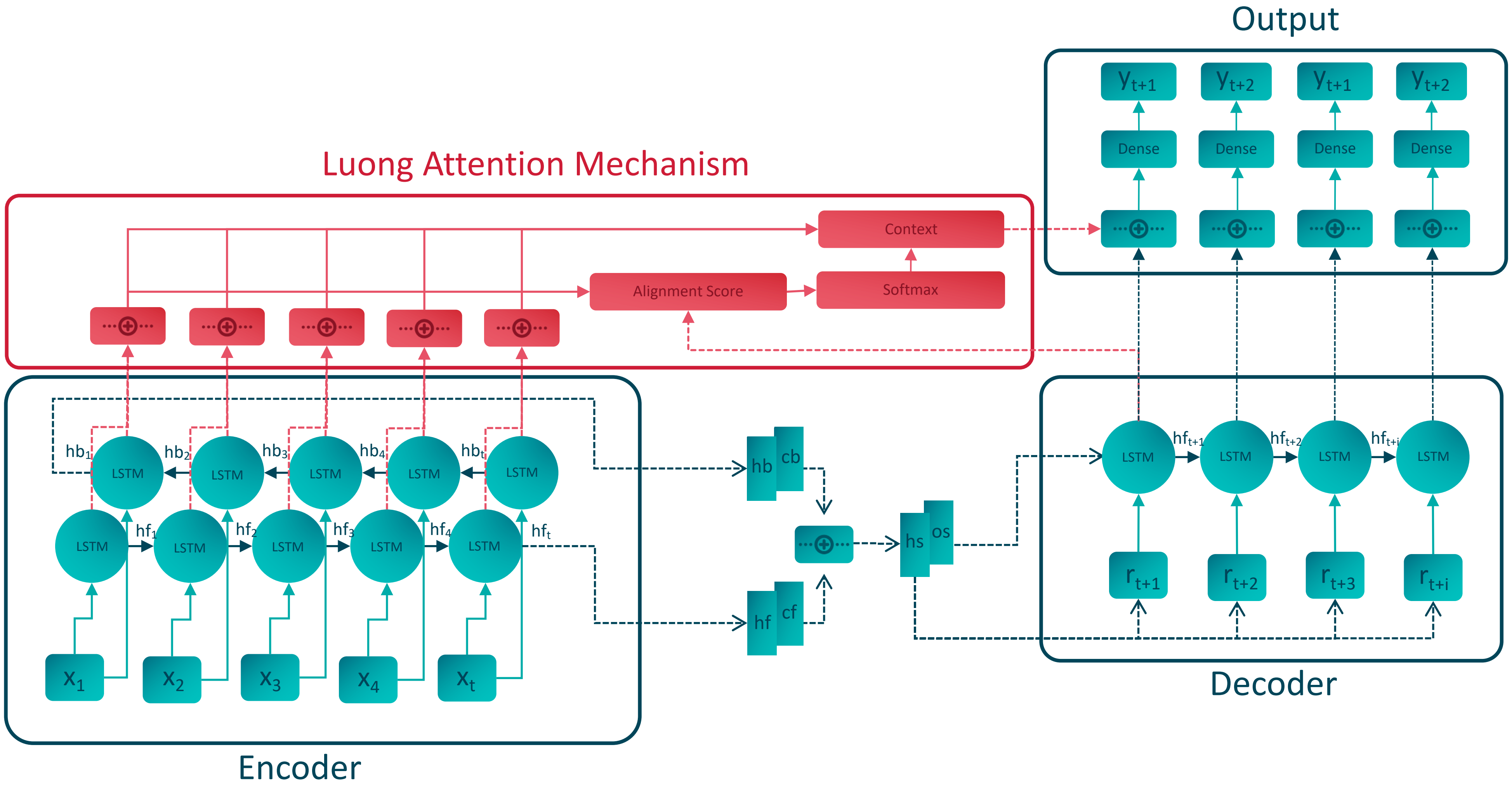
Description
A Family of machine learning models to transform sequences (time-series) into other sequences.

How it works
The model architecture consists of two submodules - an encoder and decoder. Both the architecture of the encoder and the decoder are based on so-called Recurrent Neural Networks. The encoder network transforms the input sequence into a representative encoded representation. The decoder takes this representation, decodes it, and generates the desired output sequence step by step

Speciality
Sequence2Sequence models combine several advantages. Based on neural networks, they are able to capture complex correlations even between several input sequences. Probably the biggest advantage is that the length of the input sequence can differ from the length of the output sequence.

Usage
Applications are mainly found in the field of natural language processing. Sequence2Sequence models are used there primarily for machine translation. Further applications can be found, for example, in the prediction of time series.

Bidirectional Seq2Seq LSTM Model



Legend

X = Input Features

R = Regressors

Y = Forecast

H = Hidden States

C = Output States

f = Forward

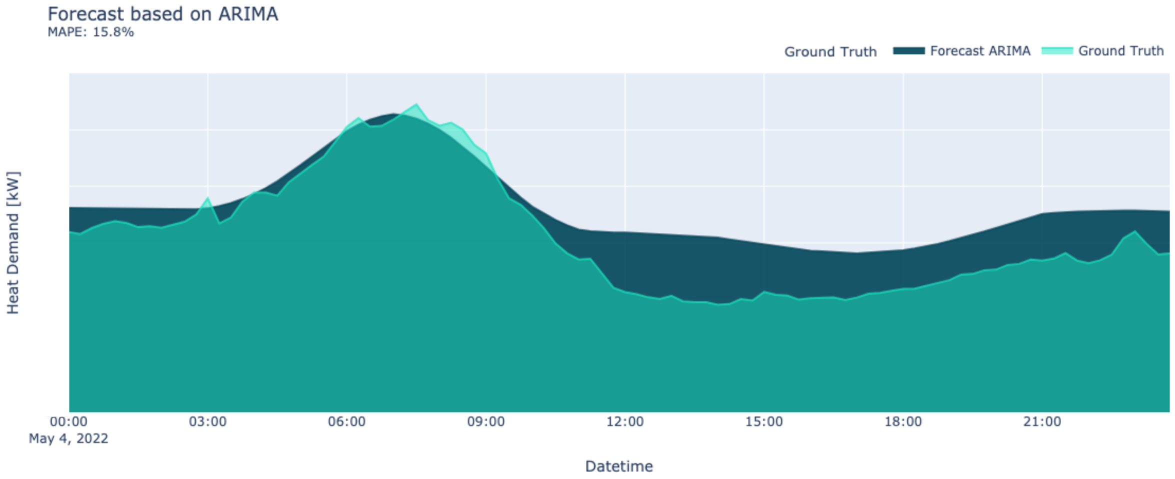
b = Backward

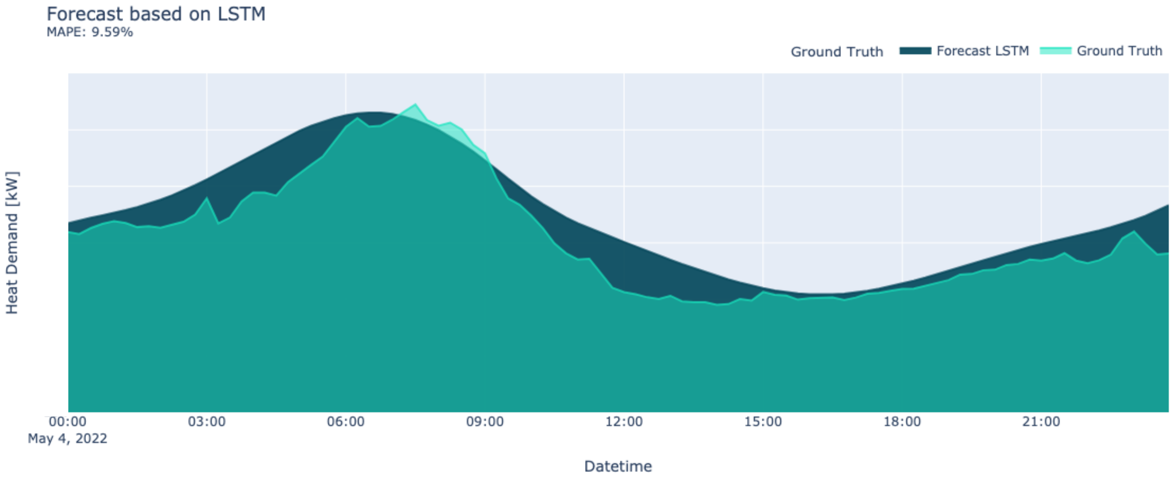
+ = Concatenation

scieneers
DRIVEN BY DATA

Forecast could be improved significantly

Comparison of heat demand forecast ARIMA vs. LSTM (May 2022)

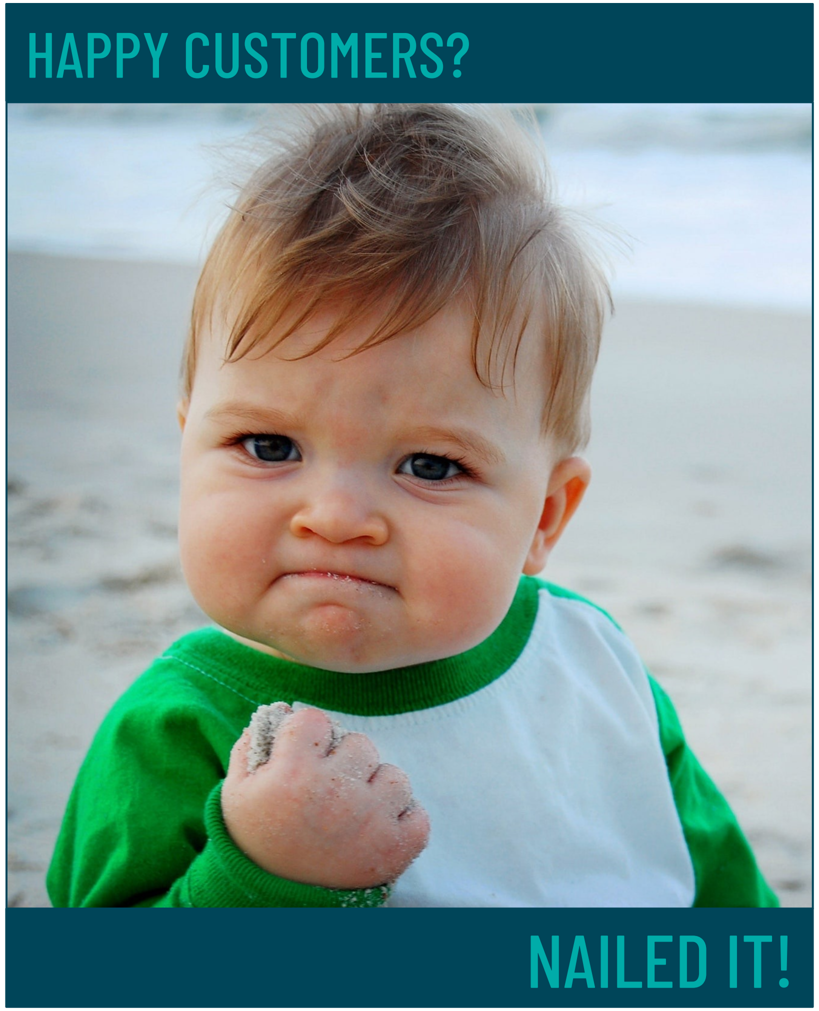




Tested Models

- Random Forests
- FB Prophet
- FB NeuralProphet
- 1D-CNN
- LSTMs/GRUs
- 1D-CNN-LSTM
- Bidirectional Seq2Seq
- Time-distributed Attention based Seq2Seq
- Time-distributed Bidirectional LSTM

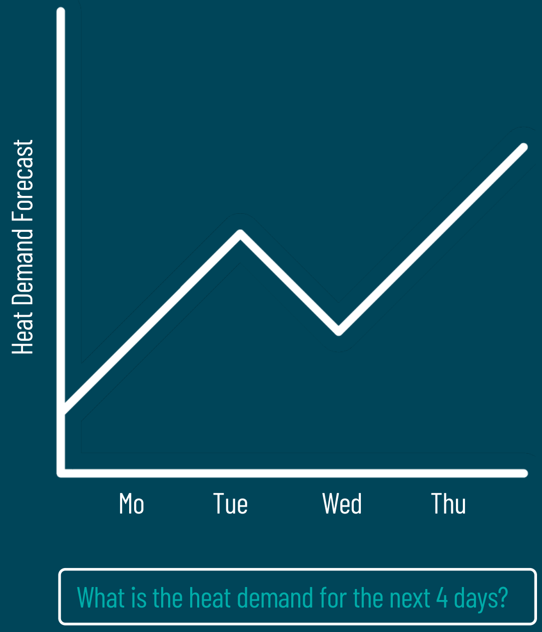
… but is it already the end of our journey?



No since...

A glimbse on our journey

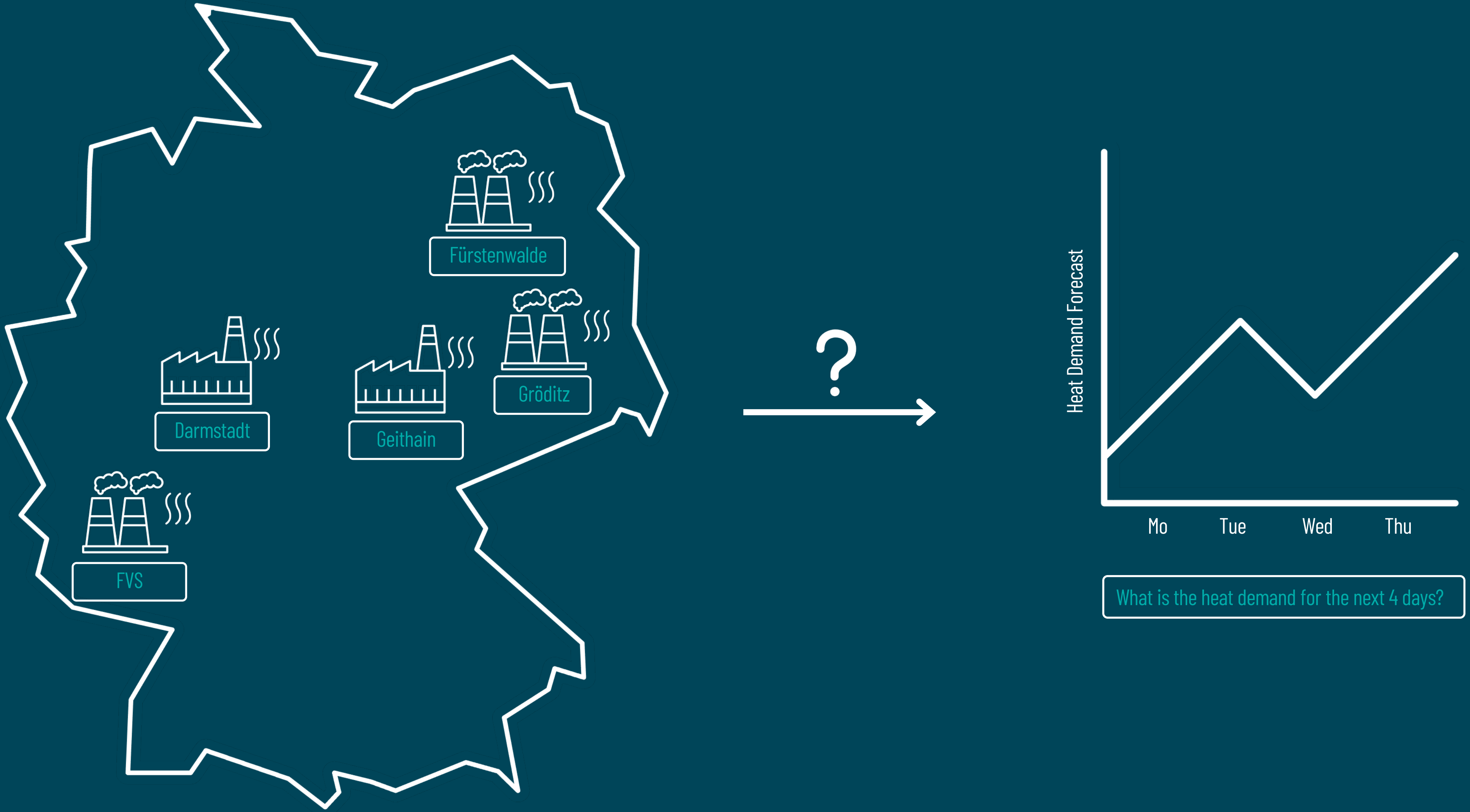
Thats where we are within our Journey right now...



scieneers
DRIVEN BY DATA

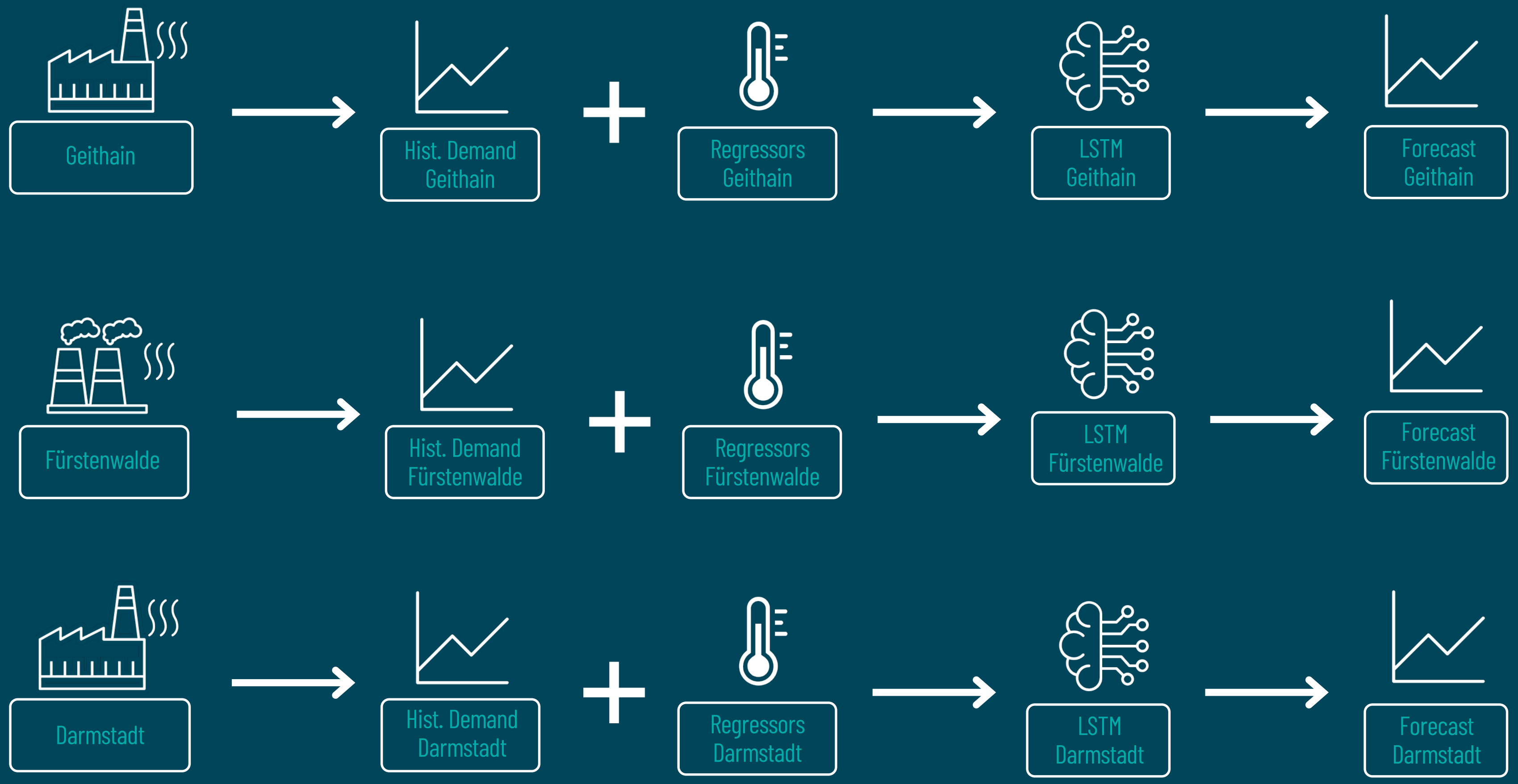
New Business Requirements

Rollout for multiple district heating systems



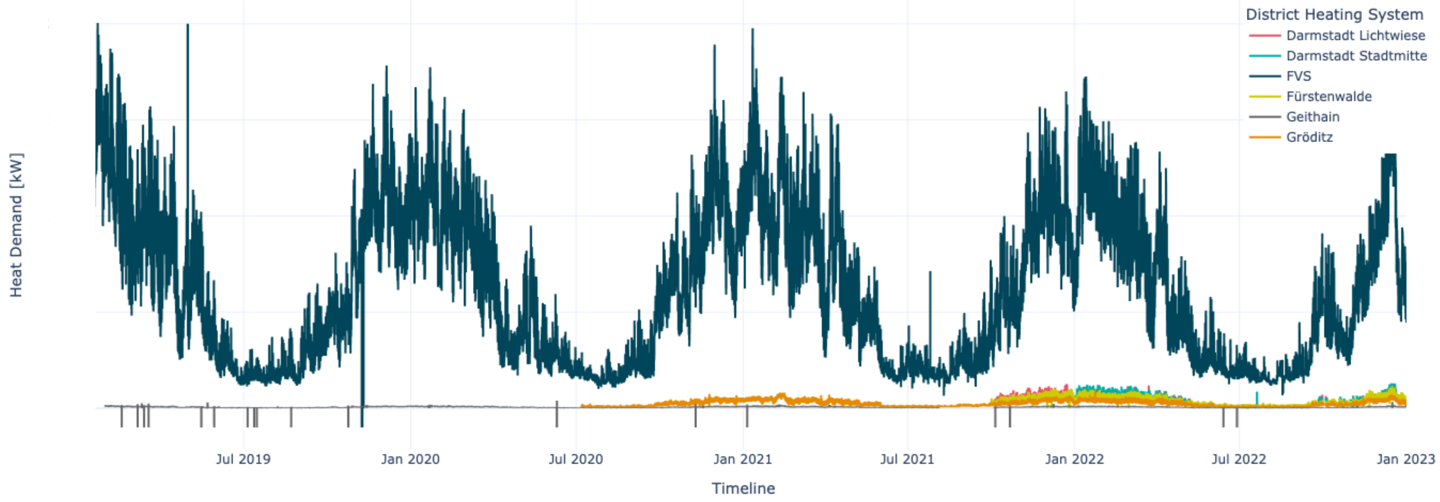
Solution

Actually not a big deal – isn´t it?



Rollout is not as simple as it seems!

Historical Heat Demand of Indiviual District Heating System at Iqony

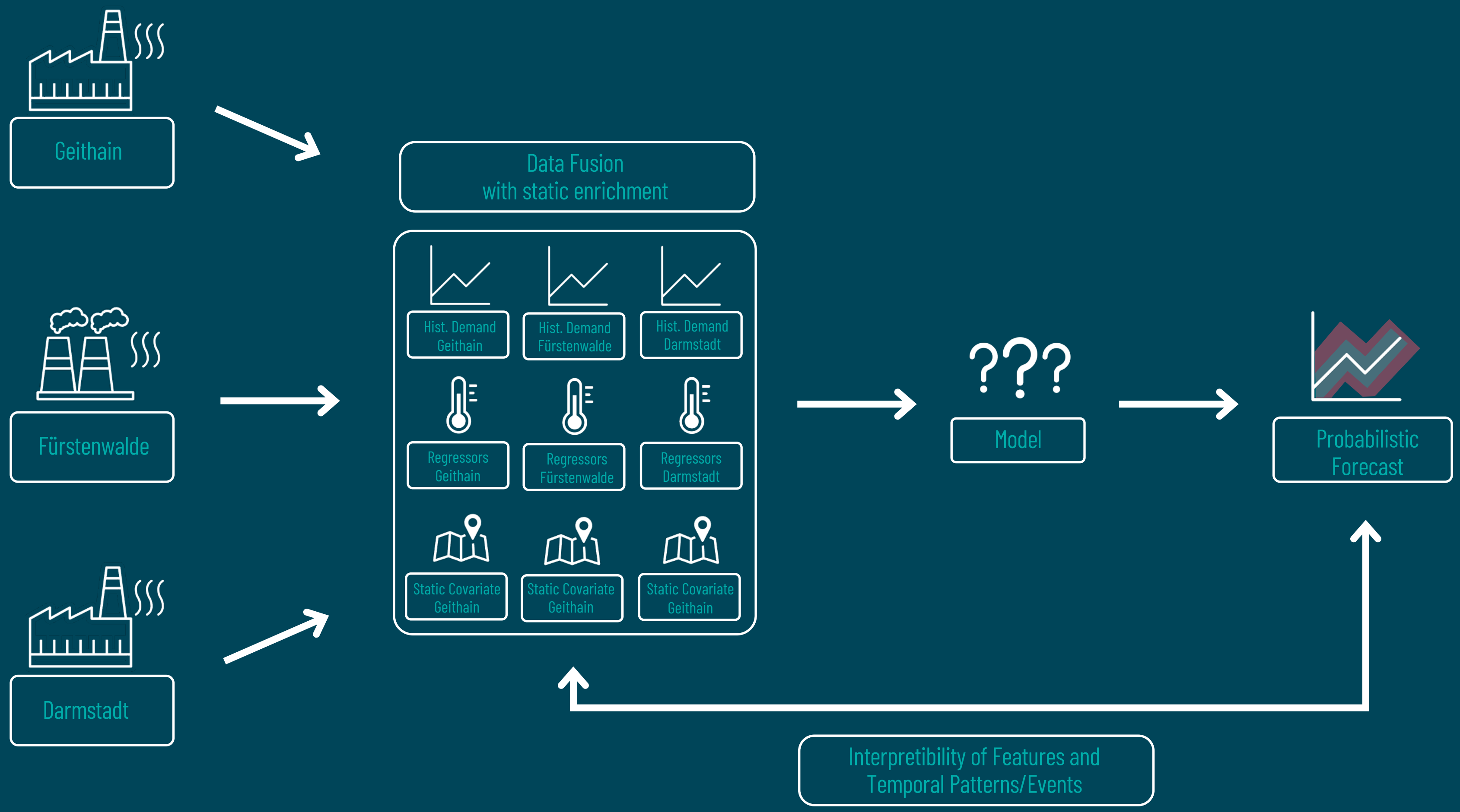


Hurdles to overcome

- Need of maintaining multiple models
- No learning from interrelationships between different modalities
- Deterministic forecasts are not suitable for any kind of risk assessment
- Lack of interpretability
- Deterministic forecasts are not suitable for any kind of risk assessment
- Hyperparameters can not be adopted
- How to handle data bases to small for a sufficient model training

What are we looking for?

A new model is required to fulfill business requirements

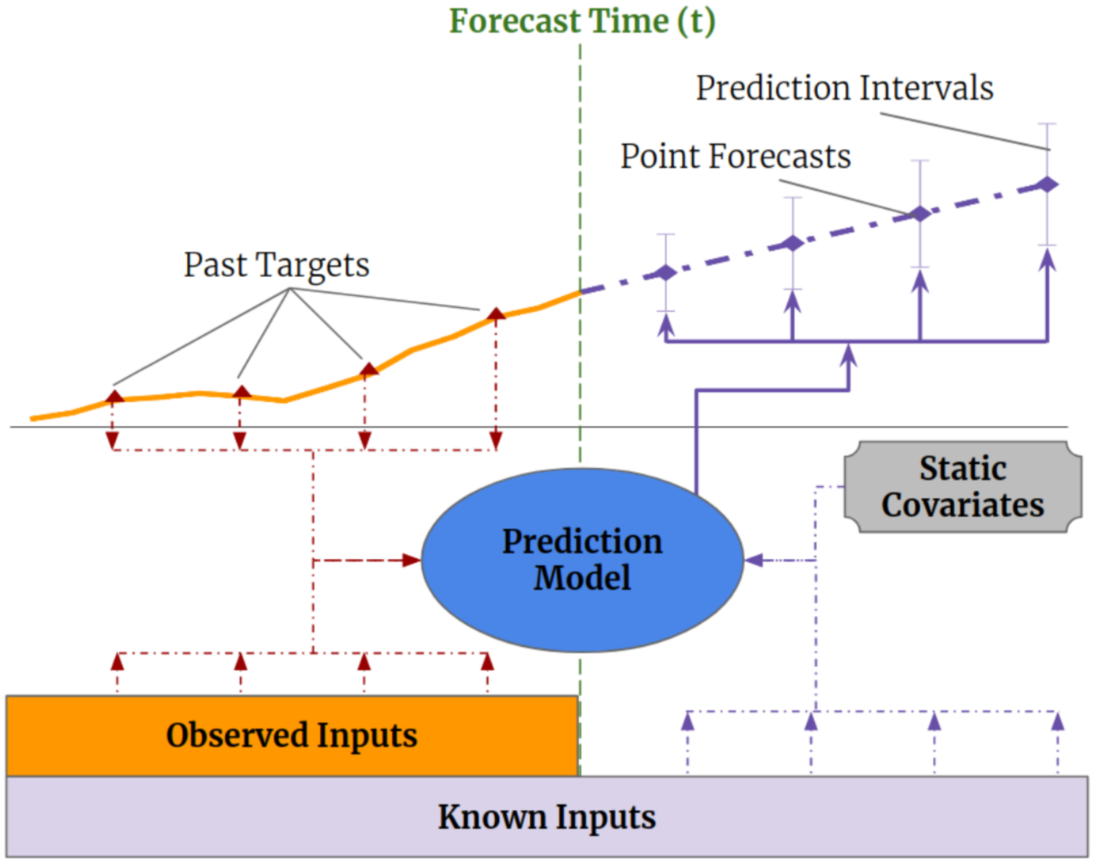


Let´s talk about
Temporal Fusion Transformers

for Interpretable
Multi-horizon Time Series Forecasting

Temporal Fusion Transformer

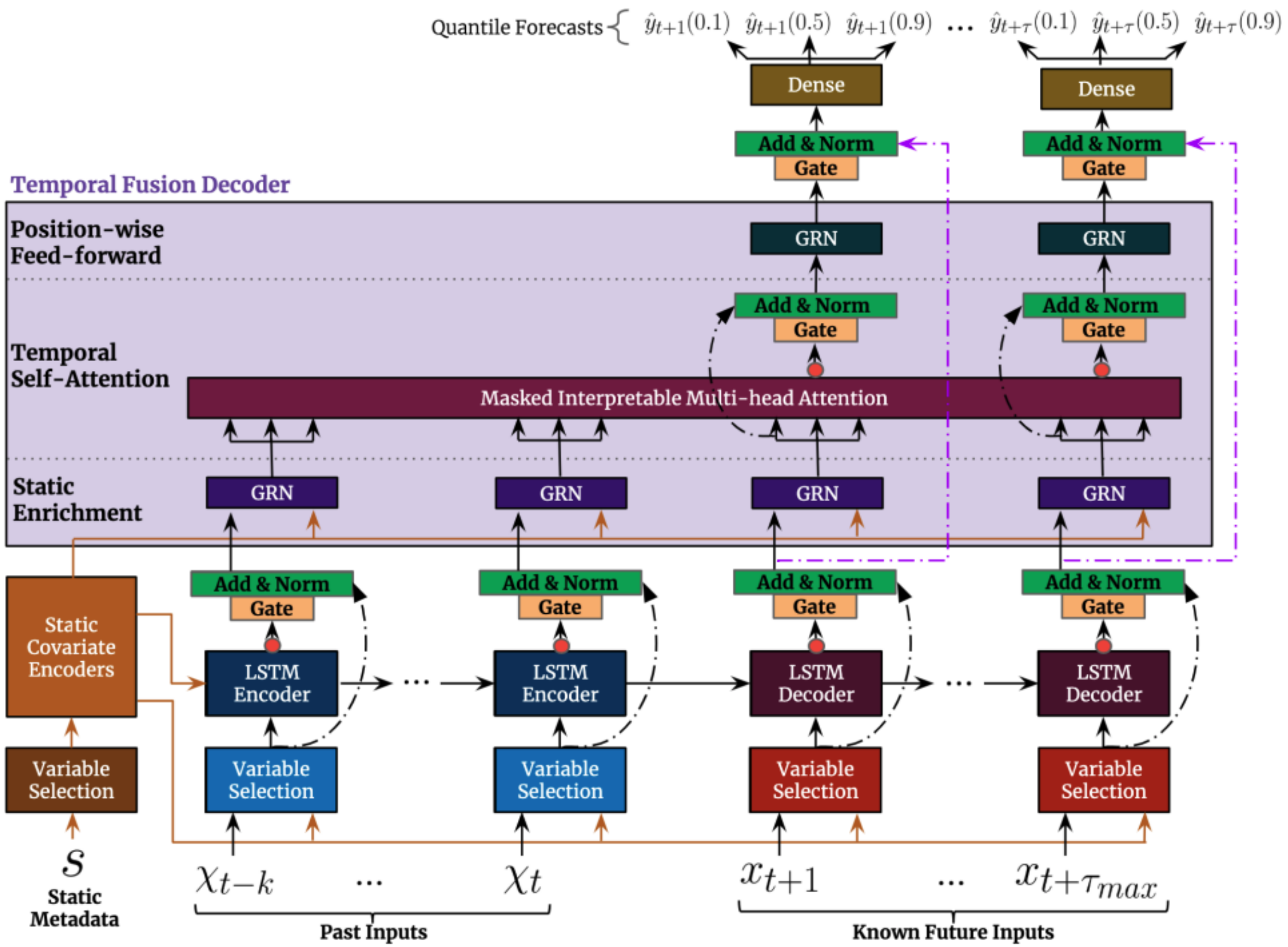
How does it work?



Source: https://arxiv.org/abs/1912.09363

Temporal Fusion Transformer

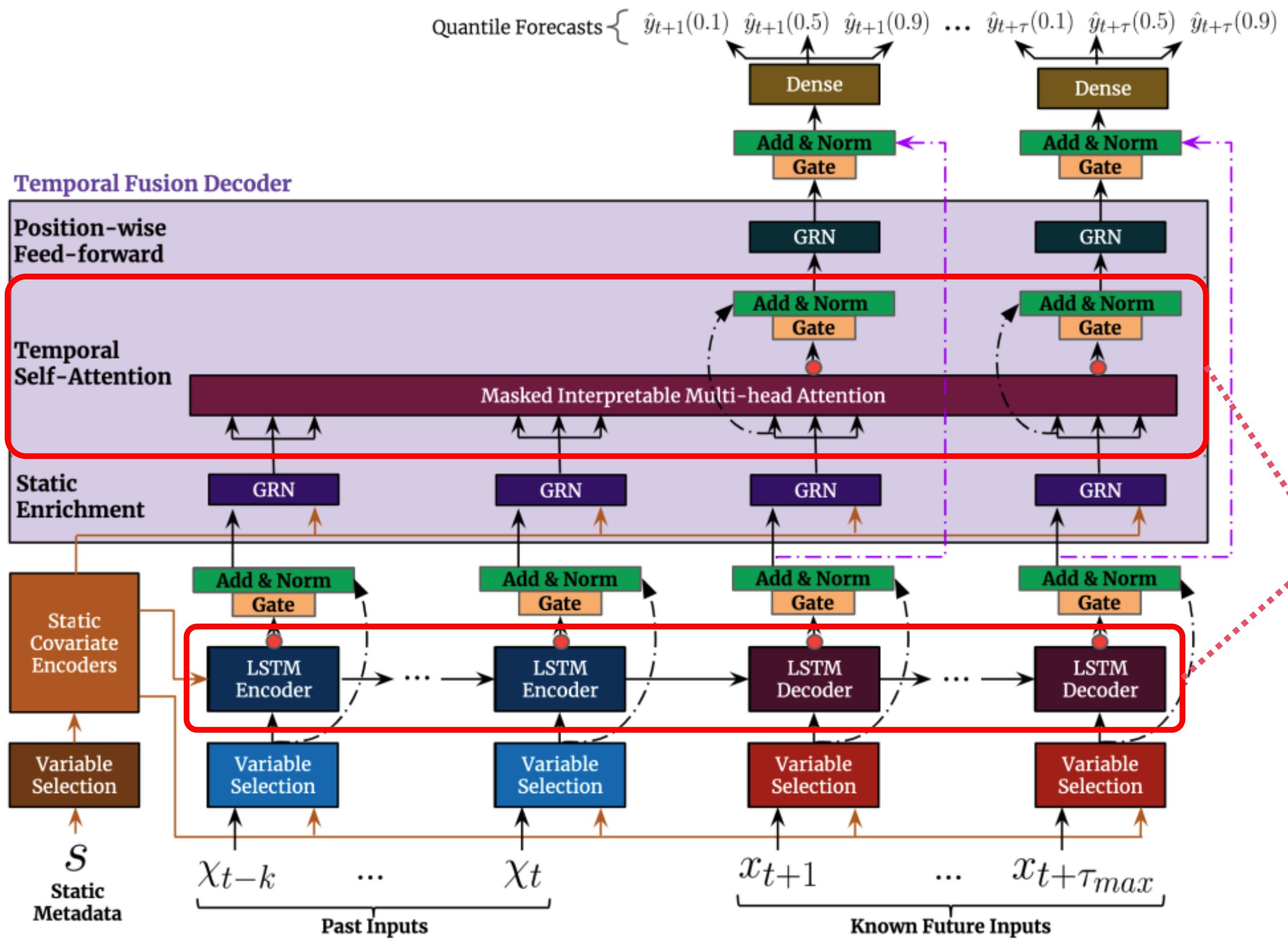
Architecture



Source: https://arxiv.org/abs/1912.09363

Temporal Fusion Transformer

Temporal Processing

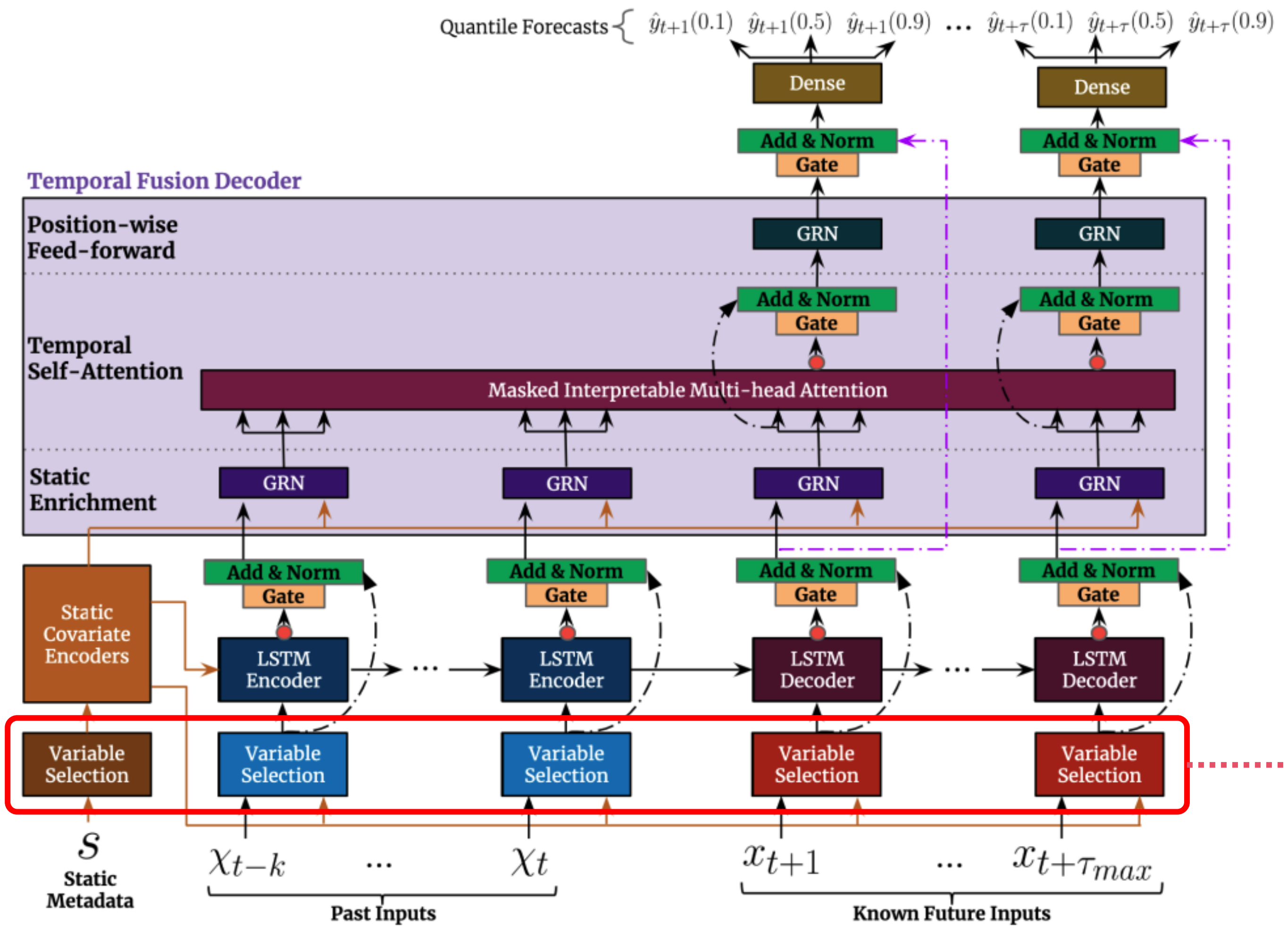


Seq2Seq layer is used for local processing, whereas the attention block captures long-term dependencies.

Source: https://arxiv.org/abs/1912.09363

Temporal Fusion Transformer

Variable selection networks

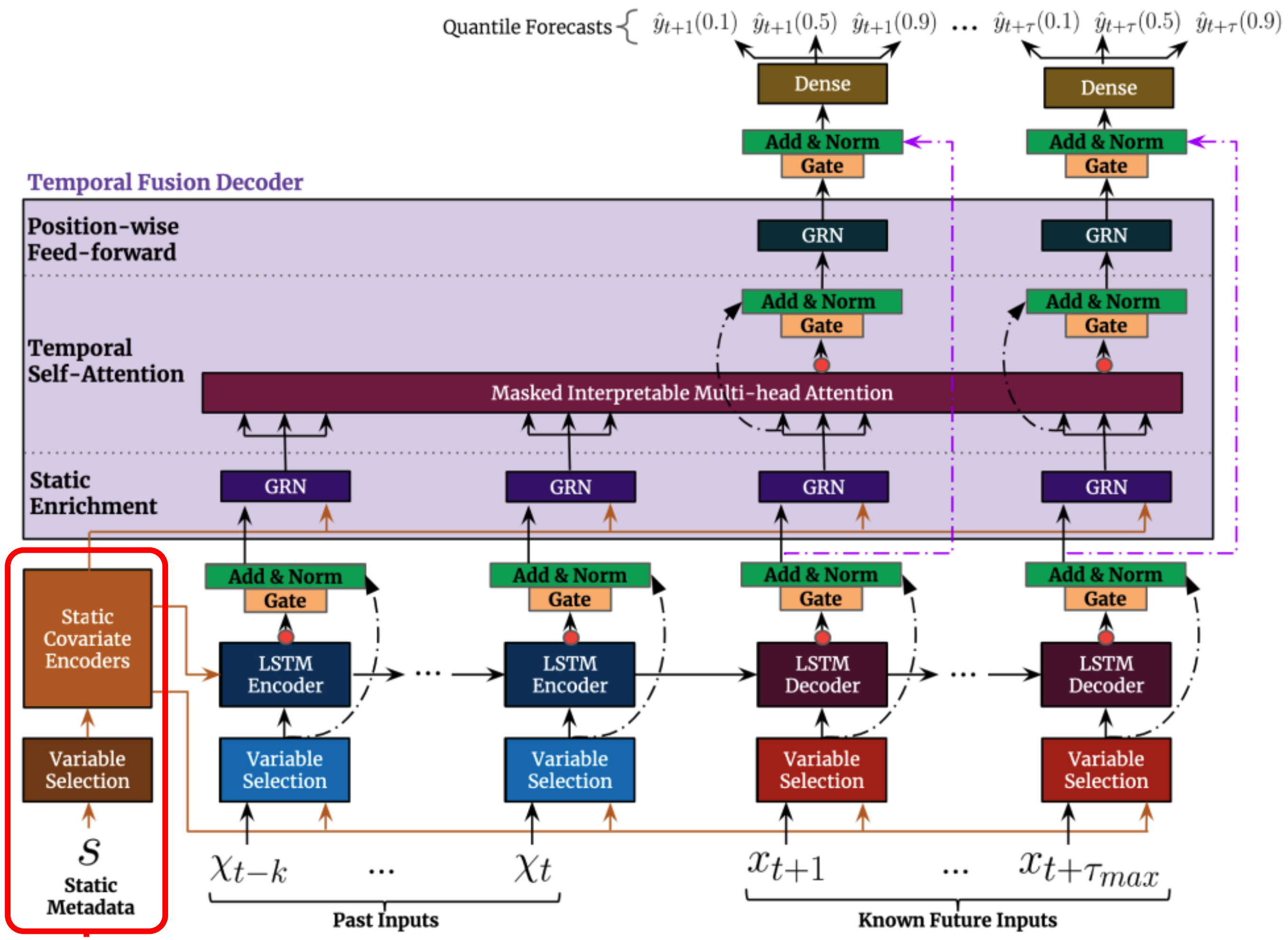


Source: https://arxiv.org/abs/1912.09363

Variable selection networks select relevant input variables at each time step.

Temporal Fusion Transformer

Static covariate encoders

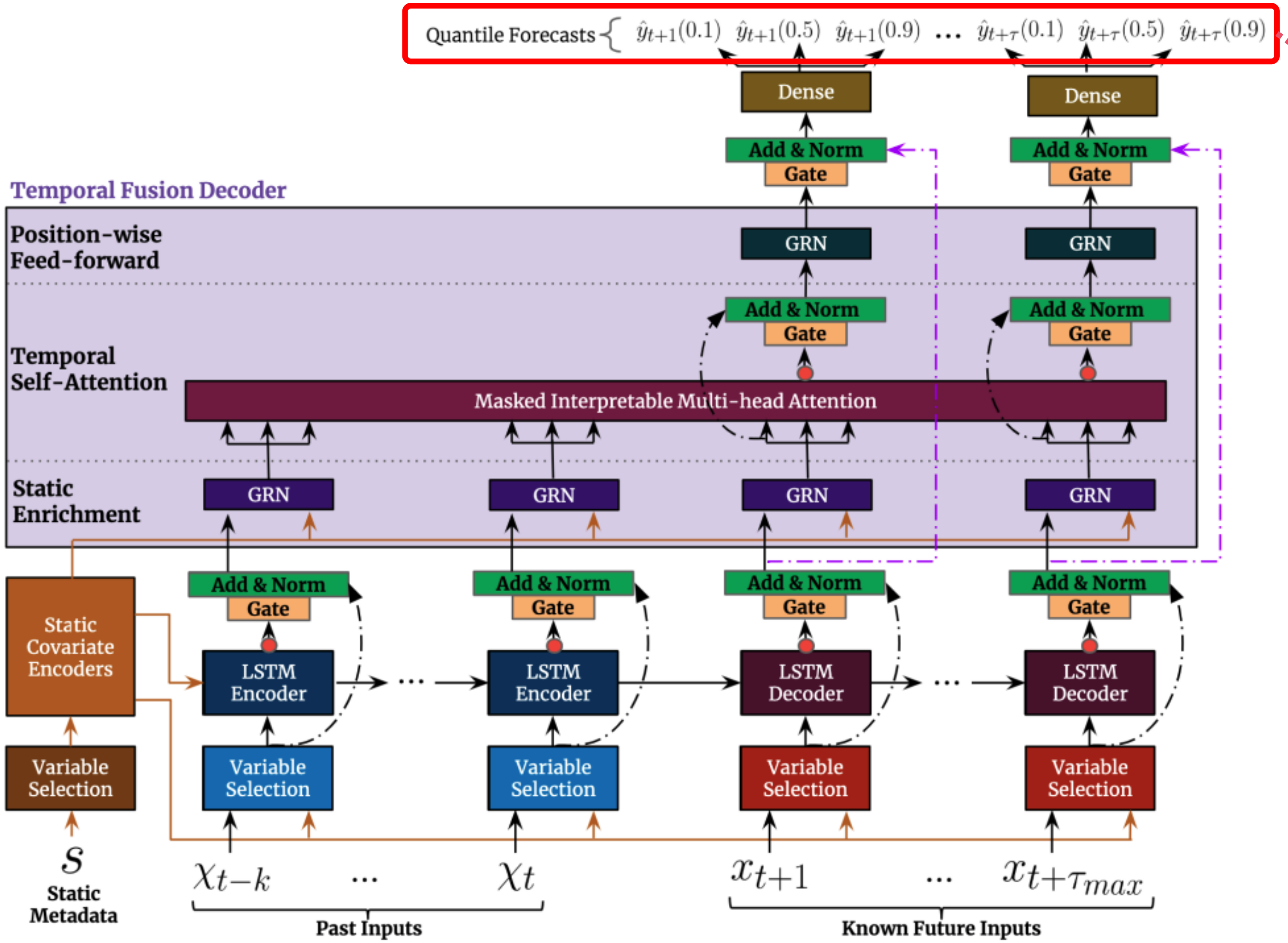


Source: https://arxiv.org/abs/1912.09363

Static features are integrated throughout the whole architecture to control how temporal dynamics are modeled.

Temporal Fusion Transformer

Quantile Forecasts



Source: https://arxiv.org/abs/1912.09363

Quantile Loss enables prediction intervals to determine the range of target values at each prediction horizon.

Temporal Fusion Transformers

Quantile Loss

Let $E_q(\hat{y}_i, y_i)$ be the quantile loss for the qth quantile and where y_i is the real value and $\hat{y}_i$ the forecast for the qth quantile the quantile loss can be defined as:

$$
\begin{aligned}
E_q(\hat{y}_i, y_i) &= q(\hat{y}_i - y_i) && if\ y_i \geq \hat{y}_i \\
&= (q-1)(\hat{y}_i - y_i) && if\ \hat{y}_i > y_i
\end{aligned}
$$

OR

$$E_q(\hat{y}_i, y_i) = max[q(\hat{y}_i - y_i), (q-1)(\hat{y}_i - y_i)]$$

scieneers
DRIVEN BY DATA

Temporal Fusion Transformers

Recommended Libraries for the Implementation

Senior: Great let´s implement the model!

Junior: But I have no idea how.

Senior: Don't worry!
There are great libraries!

PyTorch Forecasting

Darts
by Unit8.

Temporal Fusion Transformers

Implementation with PyTorch Forecasting

```
timeseries = TimeSeriesDataSet(
    dfs_train,
    **ts_kwargs
)
#create sampler for training and validation
train_sampler, validation_sampler = get_sampler()
#create corresponding dataloaders
train_dataloader = timeseries.to_dataloader(train=True, sampler=train_sampler,
↪   **train_dataloader_kwargs)
val_dataloader = timeseries.to_dataloader(train=False, sampler=val_sampler,
↪   **val_dataloader_kwargs)
#create callbacks
callbacks = []
#monitor loss on validation for early stopping
callbacks.append(EarlyStopping(monitor='val_loss', mode='min'))
#save model for best checkpoint
callbacks.append(ModelCheckpoint(monitor='val_loss'))
#create trainer for orchestrating training
trainer = pl.Trainer(callbacks=callbacks, **trainer_kwargs)
#create model which copies some parameters from TimeSeriesDataSet (e.g. encoder
↪   length)
model = TemporalFusionTransformer.from_dataset(timeseries,**model_kwargs)
```

TimeSeriesDataSet acts as interface between data and model:

- Contains information about features (static, past and future covariates)
- Defines dynamic features (e.g. relative time index)
- Sets input and output lengths of samples

TimeSeriesDataSet creates samples, which can be used for training and validation.

There is a great tutorial:
https://pytorch-forecasting.readthedocs.io/en/stable/tutorials/stallion.html

Temporal Fusion Transformers

Example – Samples from *TimeSeriesDataSet*

	Komponente	time_idx_first	time_idx_last	time_idx_first_prediction
0	DILLINGEN	0	767	384
1	DILLINGEN	1	768	385
2	DILLINGEN	2	769	386
...	...	...	...	...
534914	WALLERFANGEN	133724	134494	134111
534915	WALLERFANGEN	133725	134495	134112

Index example for samples with input and output length of 384 and static covariate *Komponente*.

Temporal Fusion Transformers

Implementation with PyTorch Forecasting

```python
timeseries = TimeSeriesDataSet(
    dfs_train,
    **ts_kwargs
)
#create sampler for training and validation
train_sampler, validation_sampler = get_sampler()
#create corresponding dataloaders
train_dataloader = timeseries.to_dataloader(train=True, sampler=train_sampler, **train_dataloader_kwargs)
val_dataloader = timeseries.to_dataloader(train=False, sampler=val_sampler, **val_dataloader_kwargs)
#create callbacks
callbacks = []
#monitor loss on validation for early stopping
callbacks.append(EarlyStopping(monitor='val_loss', mode='min'))
#save model for best checkpoint
callbacks.append(ModelCheckpoint(monitor='val_loss'))
#create trainer for orchestrating training
trainer = pl.Trainer(callbacks=callbacks, **trainer_kwargs)
#create model which copies some parameters from TimeSeriesDataSet (e.g. encoder length)
model = TemporalFusionTransformer.from_dataset(timeseries,**model_kwargs)
```

Sampler defines a strategy for drawing samples:

- Default for training samples: Random sampling without replacement
- Default for validation samples: Sequential sampling without replacement

Dataloader builds batches of samples for each epoch.

There is a great tutorial:
https://pytorch-forecasting.readthedocs.io/en/stable/tutorials/stallion.html

scieneers
DRIVEN BY DATA

Temporal Fusion Transformers

Implementation with PyTorch Forecasting

```
timeseries = TimeSeriesDataSet(
    dfs_train,
    **ts_kwargs
)
#create sampler for training and validation
train_sampler, validation_sampler = get_sampler()
#create corresponding dataloaders
train_dataloader = timeseries.to_dataloader(train=True, sampler=train_sampler,
↪   **train_dataloader_kwargs)
val_dataloader = timeseries.to_dataloader(train=False, sampler=val_sampler,
↪   **val_dataloader_kwargs)
#create callbacks
callbacks = []
#monitor loss on validation for early stopping
callbacks.append(EarlyStopping(monitor='val_loss', mode='min'))
#save model for best checkpoint
callbacks.append(ModelCheckpoint(monitor='val_loss'))
#create trainer for orchestrating training
trainer = pl.Trainer(callbacks=callbacks, **trainer_kwargs)
#create model which copies some parameters from TimeSeriesDataSet (e.g. encoder
↪   length)
model = TemporalFusionTransformer.from_dataset(timeseries,**model_kwargs)
```

Callbacks allow to interact with the training process, e. g.:

- *EarlyStopping* to abort trainings without improvements
- *ModelCheckpoint* to save promising checkpoints

When training on multiple GPUs, syncing of metrics between threads is important!

There is a great tutorial:
https://pytorch-forecasting.readthedocs.io/en/stable/tutorials/stallion.html

Temporal Fusion Transformers

Implementation with PyTorch Forecasting

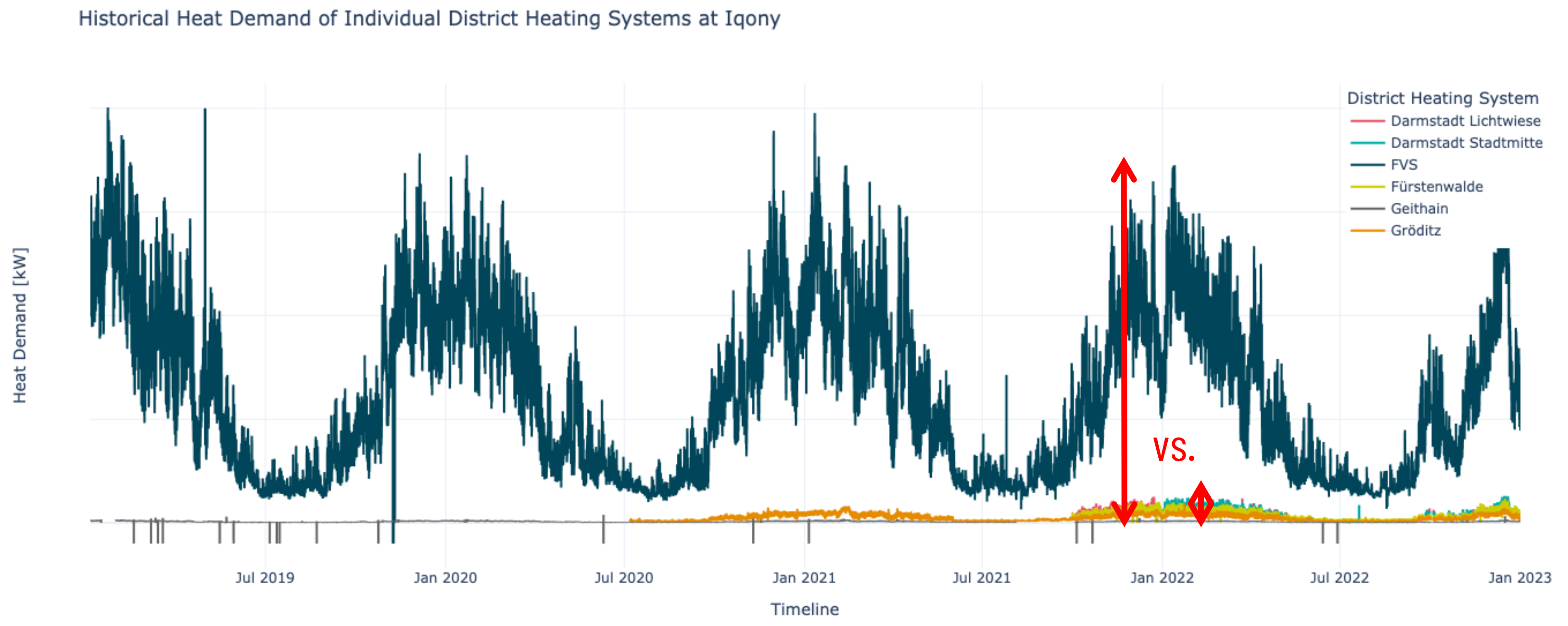
```python
timeseries = TimeSeriesDataSet(
    dfs_train,
    **ts_kwargs
)
#create sampler for training and validation
train_sampler, validation_sampler = get_sampler()
#create corresponding dataloaders
train_dataloader = timeseries.to_dataloader(train=True, sampler=train_sampler, **train_dataloader_kwargs)
val_dataloader = timeseries.to_dataloader(train=False, sampler=val_sampler, **val_dataloader_kwargs)
#create callbacks
callbacks = []
#monitor loss on validation for early stopping
callbacks.append(EarlyStopping(monitor='val_loss', mode='min'))
#save model for best checkpoint
callbacks.append(ModelCheckpoint(monitor='val_loss'))
#create trainer for orchestrating training
trainer = pl.Trainer(callbacks=callbacks, **trainer_kwargs)
#create model which copies some parameters from TimeSeriesDataSet (e.g. encoder length)
model = TemporalFusionTransformer.from_dataset(timeseries,**model_kwargs)
```

Training of model is done by *Pytorch Lightning Trainer.*

There is a great tutorial:
https://pytorch-forecasting.readthedocs.io/en/stable/tutorials/stallion.html

Temporal Fusion Transformers

Pitfalls with Data Fusion - Scaling



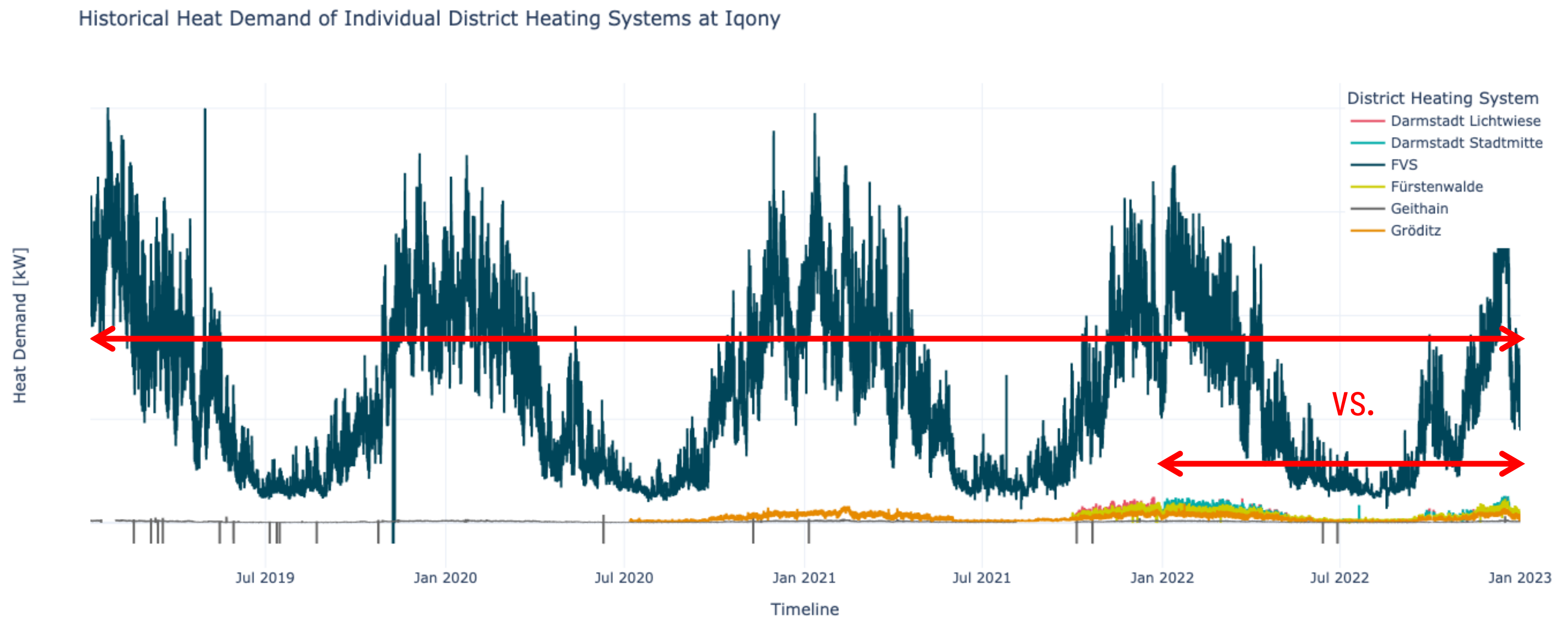
Pitfall Scaling

- As Quantile Loss does not scale, it gives preference to time series with higher amplitudes.
- Training is ignoring time series with lower amplitudes.

Scaling is mandatory!

Temporal Fusion Transformers

Pitfalls with Data Fusion - Sampling



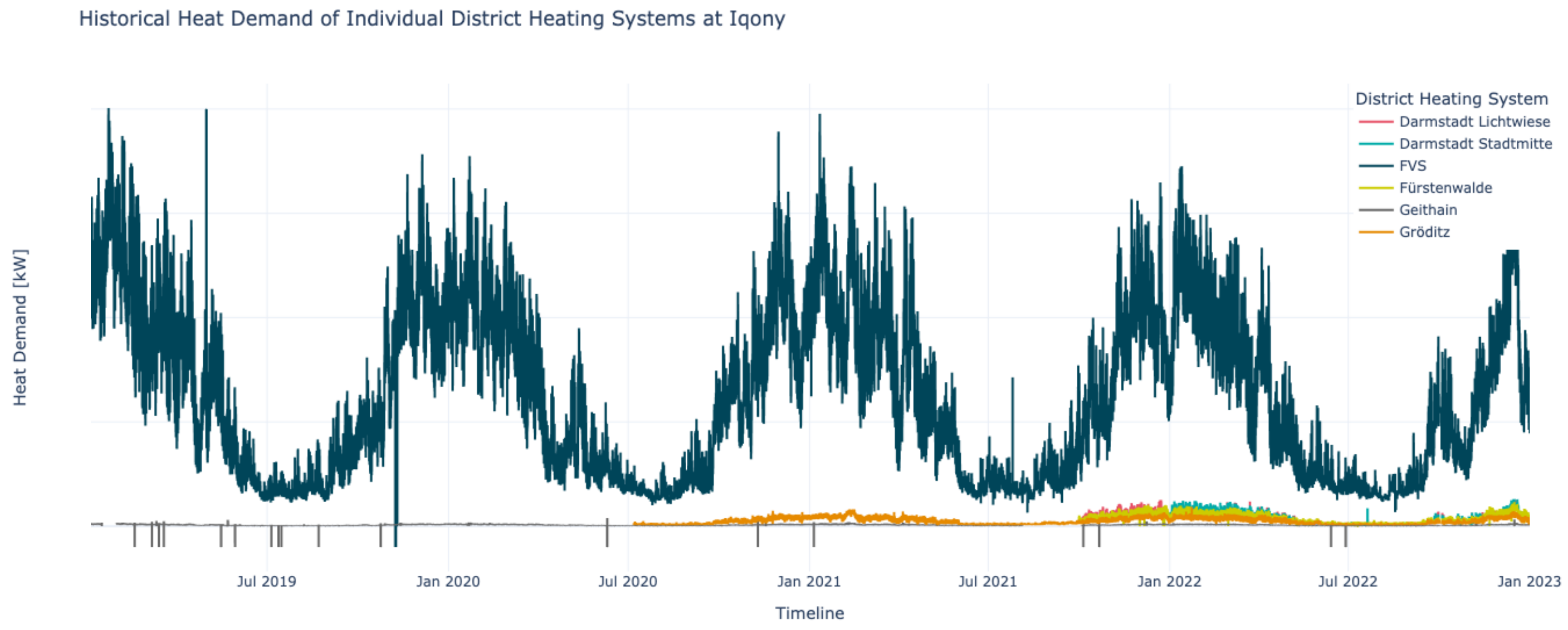
Pitfall Sampling

- As shorter time series provide less samples, random sampling results in imbalanced batches.
- Weighted sampling without replacement fixes imbalance, but samples from shorter time series may be missing in later batches due to exhaustion.

Weighted sampling with replacement fixes group imbalance.

Temporal Fusion Transformers

Pitfalls with Data Fusion - Validation



Pitfall Validation

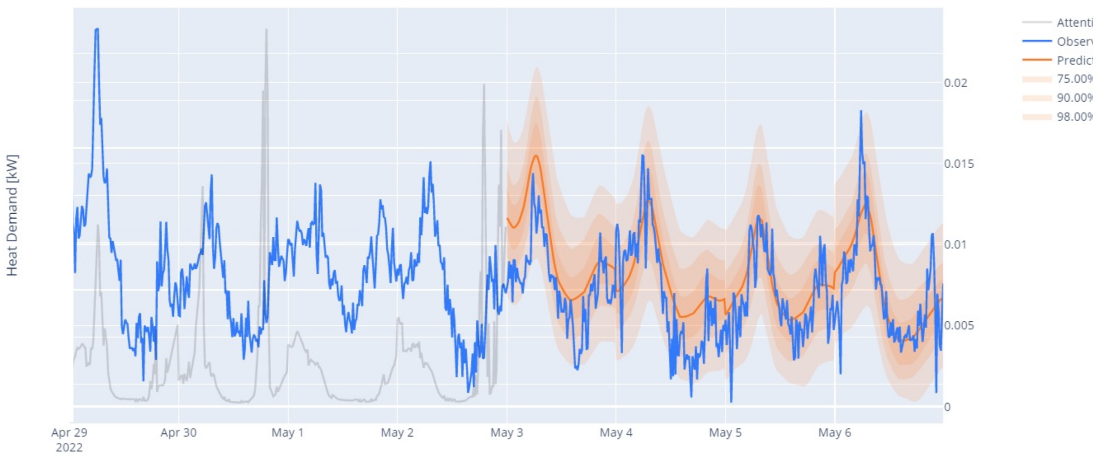
Validation may be challenging:

- How to handle shorter time series, when withholding samples for validation reduces training samples significantly?
- Do we use the same validation periods for all time series?

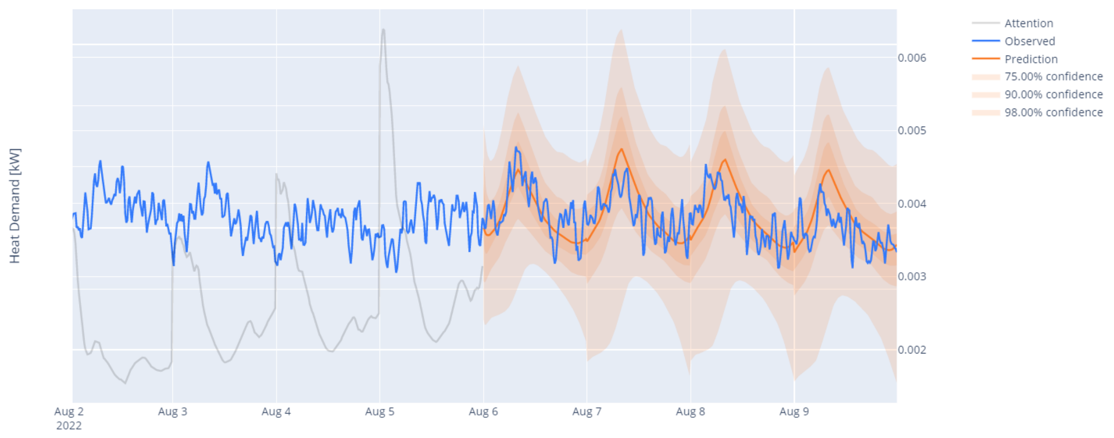
Temporal Fusion Transformer: Results

Probabilistic Forecasts for a Better Decision Making

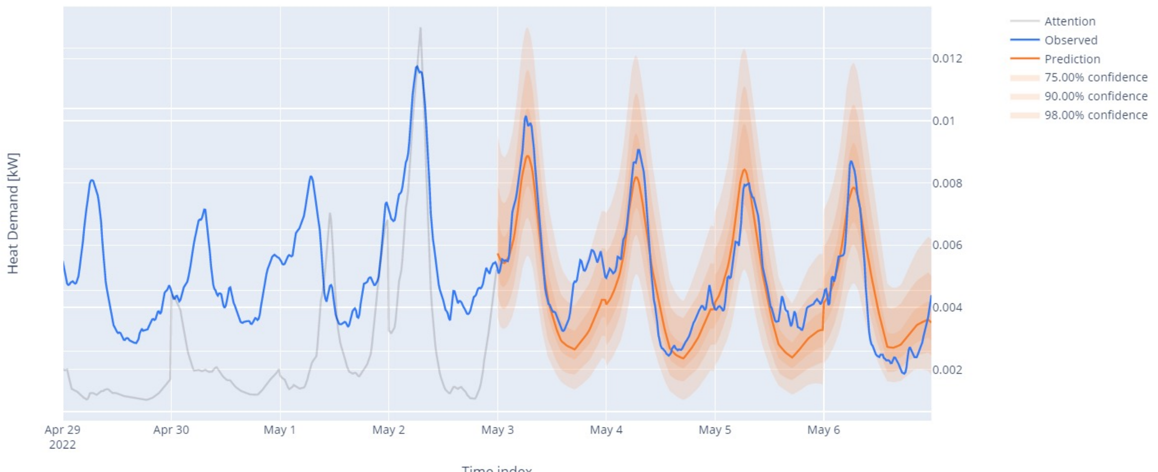
Fuerstenwalde - MAPE 13.148



Darmstadt.Lichtwiese - MAPE 8.897

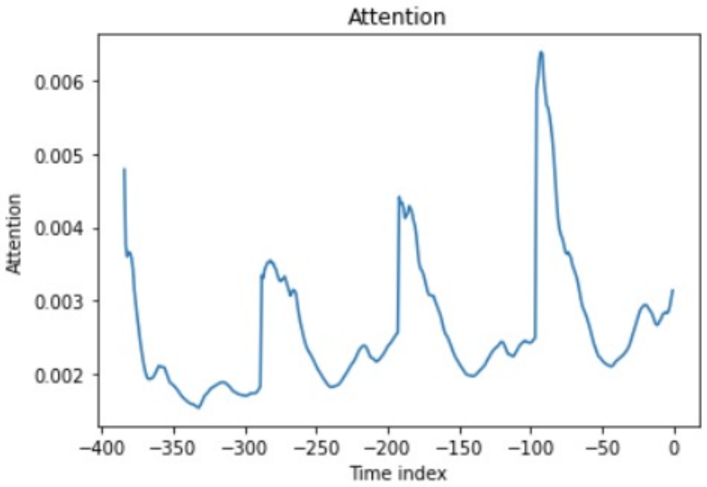


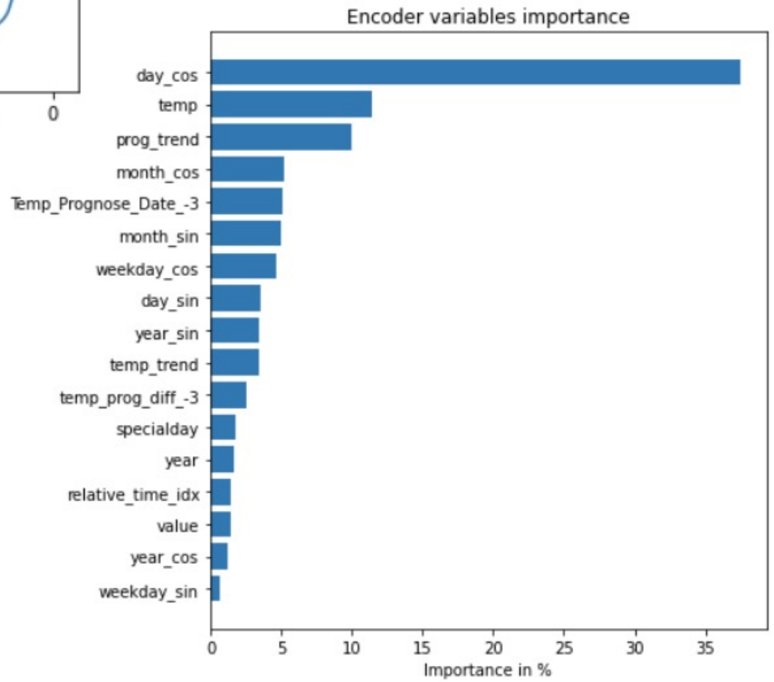
FVS - MAPE 9.952

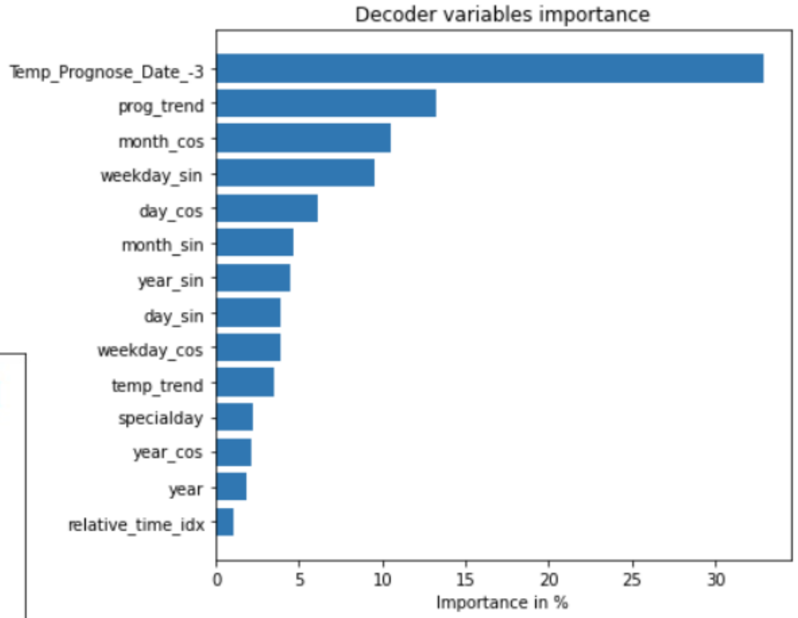


Temporal Fusion Transformer: Results

Great Plots coming along with the library provide a great insights!







Temporal Fusion Transformer: Results

A Final Comparison

Training Insights

- Attention Head Size: 3
- Dropout: 0.1
- Hidden Continous Size: 16
- Hidden Size: 64
- VM: Standard_NC12_Promo
  (12 cores, 112 GB RAM, 680 GB disk)
- GPU: 2x NVIDIA Tesla K80
- Epochs: ~ 80
- Training Time: ~24h

scieneers

DRIVEN BY DATA

Summary & Outlook

Overall Take Aways

- ARIMA model is a low hanging fruit -> fast and easy implementation leads to direct revenue
- LSTM is able to outperform ARIMA models on complex, multivariate problems
- LSTM can be highly adapted to your business requirements steadily increasing the business revenue
- TFT is especially strong when there are several modalities involved in your forecasting problem
- TFT is able of solving the cold-start problem for new modalities by making use of data fusion
- TFT is highly interpretable and provides prediction intervals for a better decision making
- Do not use models like LSTMs or TFTs if underlying problem is not that complex or you do not have much data
- Do not use models like LSTMs or TFTs if your results are already great and inference time doesn´t matter

Outlook

- Generation of additional revenue by migrating the TFT to the production environment leading to improved heat demand forecasts for multiple district heating systems at Iqony
- Using the TFT as a tool to increase the revenue through more efficient trading in the intra-day energy market.

Acknowledgement

Get in Contact!

We are happy about an exchange.

Martin Danner

Data Scientist

martin.danner@scieneers.de

Mobil +49 151 551 52 568

Jan Höllmer

Data Scientist

jan.hoellmer@scieneers.de

Mobil +49 151 551 52 562

Discussion