SOFDCAR CONSORTIUM

WHITEPAPER

# „Machine Learning (ML) algorithm-based signal boosting for latency reduction."

| | |
|---|---|
| Published by: | **P3 digital services GmbH** |
| Authors: | **L. Shianios, L. Limley, S. Salsano, T. Müller** |
| Issue date: | **December, 20th 2024** |

Funded by the European Union NextGenerationEU

Supported by: Federal Ministry for Economic Affairs and Climate Action on the basis of a decision by the German Bundestag

## Contents

## 1. Introduction

The shift towards software-defined vehicles (SDVs) has significantly amplified the demand for rigorous testing methodologies, particularly Hardware-in-the-Loop (HiL) testing of electronic control units (ECUs). As vehicles increasingly rely on complex software systems, traditional HiL testing approaches are facing scalability challenges. The exponential growth in testing requirements, driven by sophisticated features and safety-critical systems, necessitates innovative methods that leverage modern computational paradigms.

Cloud-based technologies present a promising avenue to augment the capacity and flexibility of HiL testing frameworks. By offloading testing processes to distributed cloud infrastructures, the automotive industry can potentially achieve unprecedented scalability and cost-efficiency. However, this paradigm introduces unique challenges, primarily stemming from the inherently distributed nature of cloud environments. One of the most critical limitations is latency—the delay introduced during the signal exchange between remotely connected ECUs and cloud-based testing systems. For time-sensitive automotive systems, such delays can compromise the fidelity and feasibility of HiL testing, particularly in scenarios requiring real-time or near-real-time interactions.

In this paper, we build upon the methodologies and findings presented in "ML Algorithms-Based Signal Boosting Using Synthetic Automotive CANbus Data". The previous work explored the efficiency of machine learning (ML) algorithms in signal optimization within synthetic automotive communication networks. Extending this foundation, the following study investigates ML algorithm-based signal boosting techniques specifically tailored for latency reduction in cloud-based HiL testing environments.

By addressing latency challenges, this research aims to enable seamless integration of cloud technologies in testing workflows, ensuring reliability and efficiency in next-generation SDV development.

## 2. Problem Definition

To effectively illustrate the latency problem in cloud-based HiL testing, let us consider a practical example. Imagine two ECUs that must exchange Controller Area Network (CAN) messages through a broker over the internet. The system latency is defined as the total time required for a signal to travel from the sender ECU (Alice), reach the receiver ECU (Bob), and for Bob's response signal to return to Alice. This process is depicted in Figure 1, where $u_a, d_a$ represent the upload and download delays for Alice, and $u_b, d_b$ denote the upload and download delays for Bob. Additionally, the total latency $l_a$ and Alice's required response time $r_a$, are noted, emphasizing the stringent timing requirements inherent in automotive systems.

The latency problem becomes particularly significant in distributed cloud environments, where these delays can render HiL testing ineffective, especially for real-time interactions. To address this challenge, we propose the use of Machine Learning (ML) models specialized in multi-horizon time series forecasting. As illustrated in **Fehler! Verweisquelle konnte nicht gefunden werden.**, the proposed ML model, termed the **Real-Time Enhancer (RTE)**, operates as follows. The ML model, Real Time Enchancer receives all messages exchanged in the network, and generates its results so that the generated messages will be send and received by the receiving ECU (Bob) before the original message. Bob will generate the response and the response will arrive to Alice after Alice has send the message but before the actual response message, thus reducing the total latency in the system.

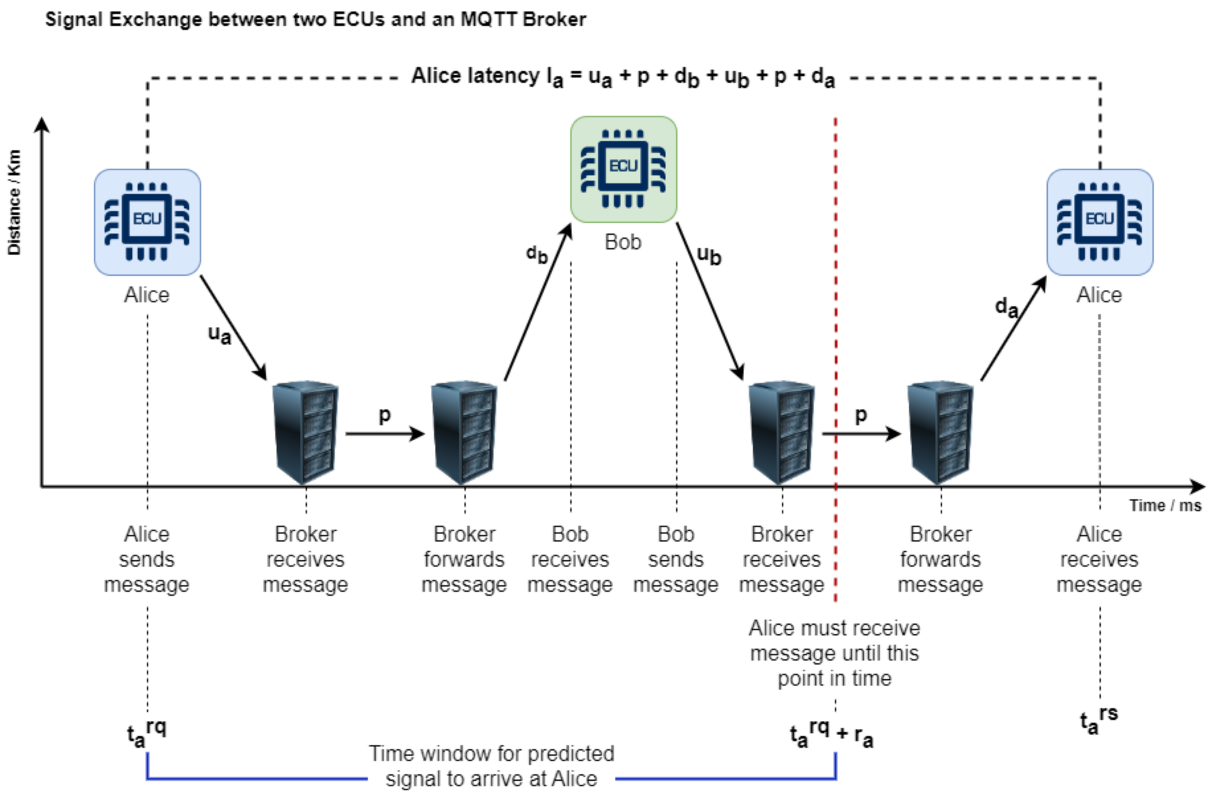

*Figure 1: The communication between two ECUs, Alice and Bob, with the delays.*

The operations of the Real-Time Enhancer (RTE) can be summarized as follows:

1. It receives all messages exchanged within the network.
2. The RTE generates predicted signals, ensuring that these predictions are sent to and received by Bob before the original messages reach him.
3. Consequently, Bob can generate a response, which arrives at Alice after she has sent her message but before the original response would have been received.

This proactive approach effectively reduces the overall system latency, enabling real-time signal exchanges even in distributed testing environments.

While promising, implementing this solution presents several challenges. First the forecasting horizon for the model is unknown during the training phase, since the horizon is being affected by the real time delays of the network, which also makes it a dynamic value that changes constantly during inference time, i.e while the model is in production the horizon is not a fixed parameter. There are a huge range of signals that can be exchanged in a CAN network and a separated model for each signal can be trained as this is going to be a very expensive solution, both in terms of training costs and maintainability costs. Having to monitor and maintain a large number of ML models in production is expensive. Thus, we need to use a versatile model that can handle multiple types of signals, varying levels of latencies, and good at multi-horizon tie series forecasting.

Given these considerations, we have selected the **Temporal Fusion Transformer (TFT)** as our ML model of choice. TFT has demonstrated exceptional performance in multi-horizon time series forecasting and is adept at managing multiple data series simultaneously. Its ability to dynamically adapt to varying

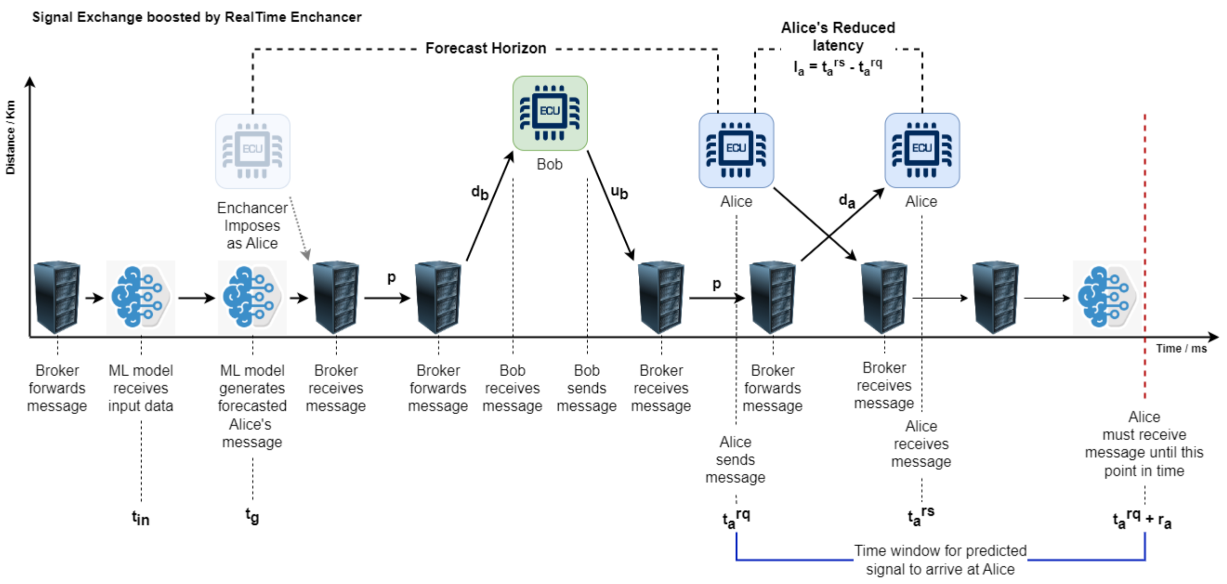

*Figure 2: The communication between two ECUs, boosted by the Real Time enchancer.*

conditions and efficiently handle diverse signals makes it an ideal candidate for addressing the latency challenges in cloud-based HiL testing.

## 3. Theoretical Framework

Temporal Fusion transformers were first introduced for interpretable multi-horizon time series forecasting[1]. Multi-horizon forecasting is the prediction of time series variables at multiple future steps. Such forecasting applications have data from a variety of sources that include known future inputs, observed i.e. historical time series, and static metadata.

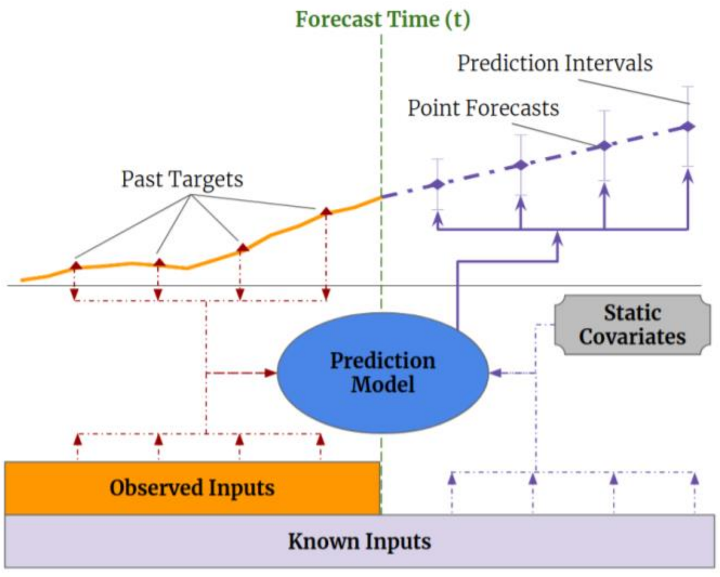

*Figure 3: Illustration of multi-horizon forecasting with static covariates, past-observed and a priori-known future time-dependent inputs, source [1].*

Temporal Fusion Transformer (TFT) is an attention-base deep neural network. Attention mechanism help capture the importance of input values in relation to each other. When used in time series data, it helps to learn the relevance of each time step with respect to the rest of the series. Attention mechanisms were first used in time series with the purpose of interpreting the results of forecasting models.

TFT incorporates some novel ideals to achieve SOTA benchmarks. It includes static covariate encoders which encode data to condition temporal dynamics and the encoded data to be used in other parts of the network. Gating mechanisms and sample-dependent variable selection to exclude irrelevant inputs affecting the result by skipping over unused parts of the model. This allows to have adaptive depth and network complexity based on the dataset and the scenario. Sequence-to-sequence encoder to process known future values, and past observations. A novel interpretable temporal multi-head self-attention to capture any long-term and short-term dependencies in the data, both across the observed and known time-varying inputs. For multi-horizon forecasting the model adopts quantile regression. That is for each time step the model predicts the 10, 50 and 90% quantiles. A quantile of probability $\tau$, represents a threshold where the probability of observing a value lower than the threshold is exactly t. For example, the 10% quantile with threshold 1, means that the probability of observing a variable

[1] Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting, Lim et.al, Sep 2020, link

having a value lower than one is exactly 10%. In the case of TFT the model generates those thresholds for the predefine probabilities of 10, 20 and 90%.

Adapting the Temporal Fusion Transformer (TFT) model to address our specific problem requires minimal modifications. The process primarily involves identifying and categorizing input variables into three key types: observed inputs, known future inputs, and static covariates.

In our case:

- **Observed Inputs:** These include the actual signal values and the corresponding latency values, which are dynamically observed during testing.
- **Static Covariates:** The name of the signal is treated as a static covariate, as it remains constant throughout the process.
- **Known Future Inputs:** Although the maximum latency of a signal is inherently static, it is passed as a known future input to better align with the model's requirements.

This structured categorization ensures that the TFT model can effectively process the data, leveraging its strengths in handling diverse inputs to optimize latency prediction and reduction.

## 4. Methodology

The core approach leverages the Temporal Fusion Transformer (TFT) to perform multi-step-ahead forecasting, explicitly accounting for system latencies. Rather than training separate models for each signal type, the methodology involves training a model for each distinct latency distribution identified between ECUs in the system.

The process starts with measuring latency distributions for various ECU configurations. These measurements create synthetic datasets used to train TFT models. This allows the models to effectively manage the variability of real-world latency. In this R&D project, the focus was on one ECU configuration, but the method can be expanded to include multiple setups or distributions. This streamlined approach not only simplifies the modeling process but also ensures scalability, enabling the integration of the TFT model into diverse cloud based HiL testing scenarios with varying latency characteristics.

## 5. Latency Measurements

To measure the actual latencies in the system we developed a virtual ECU where it sends periodic CAN messages that include the signals *StartTime, MessageNumber, Latency, Period, SendFrameId* and *ReceiveFrameId*. The measurements were done by connecting two ECUs on the same virtual connector box, running on the same laptop in Greece. The connector box allows the ECUs to communicate with the MQTT broker and the rest of the platform. The ECUs were sending messages to the mqtt broker deployed in West Germany, (on Azure data center), and the broker was bouncing the messages back to the connector box, which in turn delivered to the ECUs. When an ECU was generating a message, it registered the local time as the StartTime signal. The message number was a counter of the messages generated by the ECU, counting from zero and increasing with each consecutive message by one; it was registered as the MessageNumber. The time between consecutive messages was registered as the Period. The Latency and ReceiveFrameId were initiated with the default values of zero. The message was packed by the connector box and send to the broker.

Once a message was received by the connector box, it was sent unpacked and delivered to the ECUs. On receiving a message, the ECUs were set to register the local time, and subtract the StartTime from the message to calculate the Latency. They updated the Latency signal in the message and the ReceiveFrameId signal with their id and send the message back to the connectorbox. The connector box sent the updated message to the broker, which again bounced the message back. This time when the ECUs received the message, they found that the Latency and ReceiveFrameId signals were not equal to the default values, thus they dropped the message. During testing the Period of the messages was changed every five minutes; the values used were 1000ms, 500ms, 50ms, 25ms, 10ms, 5ms.

All messages that arrived at the broker were also stored in the object store of the system. Those stored messages were used to analyses the latencies of the system. It was found that with the minimal resources allocated to the broker, 0.5 CPU cores at 3.4 GHz and 500 GB RAM, at around 200 incoming messages per second and 100 outgoing messages per second, the broker could not handle the load and introduced sever delays in the system. This can be seen in Figure 4. All messages generated after the time the load increase to those levels, were discarded from the analysis. The reason is because the major factor of the latency measured was caused by the resource allocation to the system and not due to the network. Any latencies introduced with the system should be addressed primarily by better configuration of the system and not by the Real Time Enhancer.

For the surviving messages the latency was found to follow a heavy tail distribution, something that was expected for public internet lines. The distribution is shown in Figure 5. The peak of the distribution is around 50miliseconds, with 75% of the measurements falling under 128 milliseconds. It is this long tail in the distribution that makes the job of the Real Time enhancer particularly challenging.

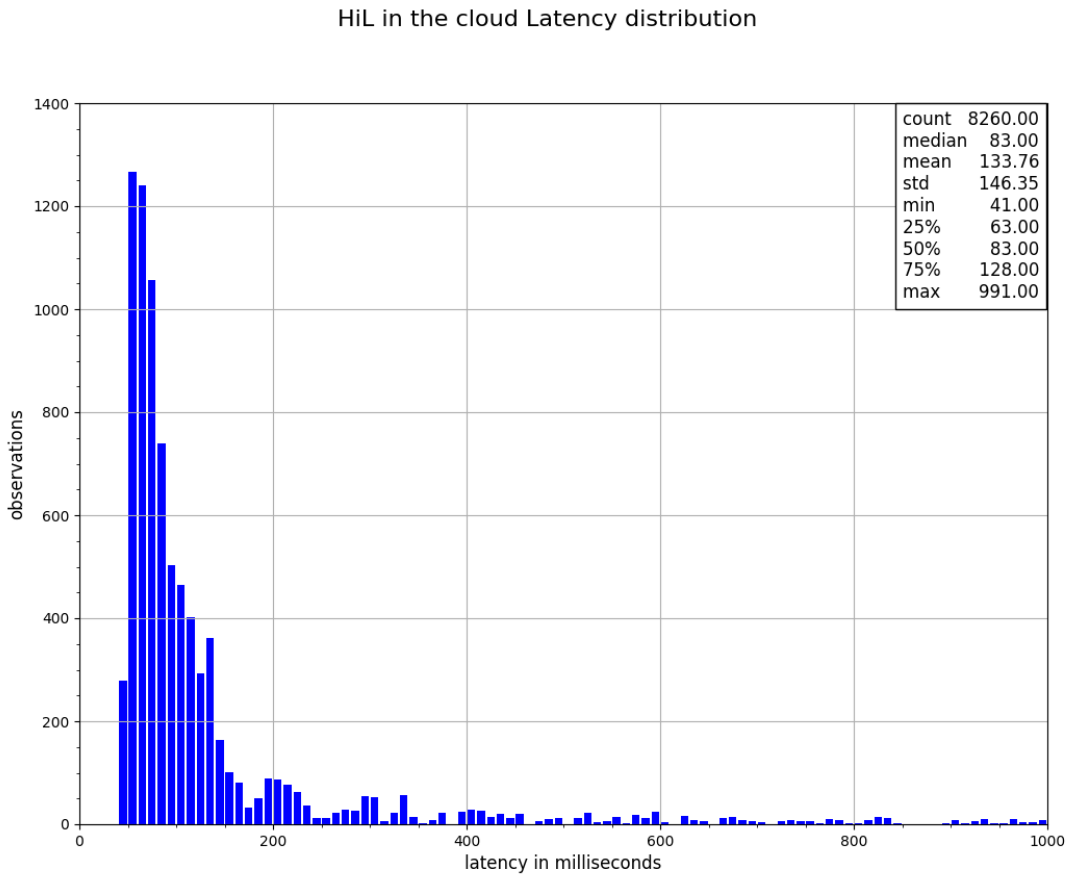

*Figure 5: The latency distribution, as measured between ECUs deployed in Greece and the server deployed in Germany.*

To summarize the section

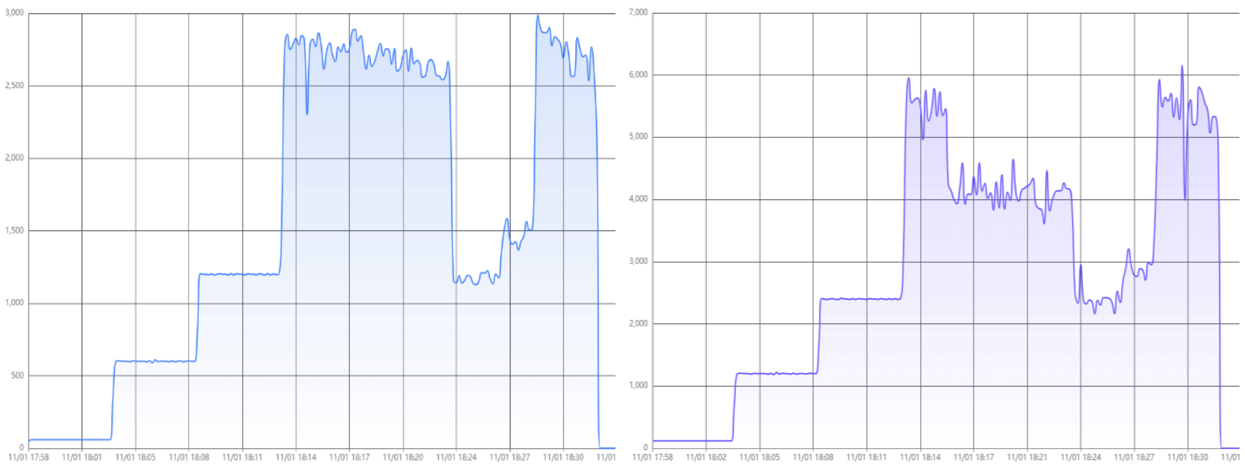

*Figure 4: The message throughput of the broker during one of the latencies measurement tests. To the left is the total number of incoming messages per 30 seconds and to the right is the total number of outgoing messages per 30 seconds.*

- To model the latency in the synthetic data used to train the real time enchancer, the actual latency needs to be measured.

- To measure the actual latency a virtual ECU was developed that was exchanging simple messages with the broker.
- With the minimum resources allocated to the broker the limits of the system were found to be 200 inbound messages per second and 100 outbound messages per second.
- The measured distribution of the latency was found to follow a heavy tail like distribution shown in Figure 5.
- The peak of the distribution was at 50 ms, and the 75% percentile was at 128ms, meaning that 75% of the messages were delayed by at most 128ms, to complete a round trip.

## 6. Data generation

To prepare the data set, first we generated a set of signals, that span a range of parameters. First four classes of signals were considered, continues periodic, discrete periodic and step functions, exponentials and sigmoid, and finally linear. Each function was given a set a variables to be generated, those were, the step which indicated how often the signal should be sampled, the frequency, which in the case of periodic functions sets the frequency of the signal, where as in the case of linear and exponential it was a multiplying factor in the expressions, the amplitude of the signal, the x-offset and the y-offset. The full set of parameters is shown in **Fehler! Verweisquelle konnte nicht gefunden w erden.**.

| Parameter name | Values | Total number of values |
|---|---|---|
| step | [1.0, 0.5, 0.1, 0.05, 0.01] | 5 |
| frequency | [1, 0.5, 0.1, 0.05] | 4 |
| max_latency | [1, 0.5, 0.1, 0.05] | 4 |
| amplitude | [0.1, 1, 10] | 3 |
| x_offset | [0.0, 5.0, -5.0, 10.0, -10.0] | 5 |
| y_offset | [0.0, 1.0, -1.0, 10.0, -10.0] | 5 |

*Table 1: The set of parameters used to generate the synthetic data for the training of the TFT model.*

Each combination of parameters was passed to the generator function and a total of 600 timesteps were generated for each signal. The generator function with the signal values was also generating the timestep in seconds for each value. In some combinations for the periodic signals, due to the values of the frequency and step, the resulting series was not exhibiting the periodicity it should, hence those series were dropped from the data set. There was a total of 6000 parameter combinations and four classes of functions giving us a total number of signals of 24K, and after dropping the problematic periodic functions, a little more than 22K survived in the final dataset. The max latency is to simulate the latency requirement of an ECU.

To include the latency in the generated signals, a random generator was used that generated values that were falling under the measured latency distribution. A random value was then added to each timestep of the generated signals. That number was also included in each record of the dataset. Using the max latency for each signal, the target value for each time step was calculated. If the latency for a timestep $v_t$, was less than the max latency of the signal then the target value for that time step was the value of the next step i.e., $t_t = v_{t+1}$. If the latency value for the timestep $t$, is greater than the max latency of the signal then, the target value for the timestep $t$ will be $t_t = v_{t+s}$, where $s = ceil(latency/step)$. That is, we divide the latency by the step value of the signal and round up to the rearrest integer, to get the number of timesteps ahead the target should be. An example of a generated signal is shown in Figure 6. The black line shows the original signal in noiseless timesteps. The blue dots are the signal values with timesteps that include the latency and the red dots are the target value for each timestep.

The features of the final dataset include the "signal name" which it was generated based on the parameters used and the type of the signal, the signal value, the timestep, the latency, the max latency, and the target value. Also, an id field is included which has the same value as the signal name. This is needed for the implementation of the TFT model.

## 7. Data preprocessing

For the implementation of the TFT model we used the publicly available git repositories which can be found in this links[2][3]. They include the data preprocessing steps for the TFT model. For our case the observed variables are the signal value and the latency, the known input the max latency and the static input the signal name.

The data was split into three sets, the training, validation, and test set. For the test set a few signals were excluded from the original set, based on their names, and only included in the test set. For the

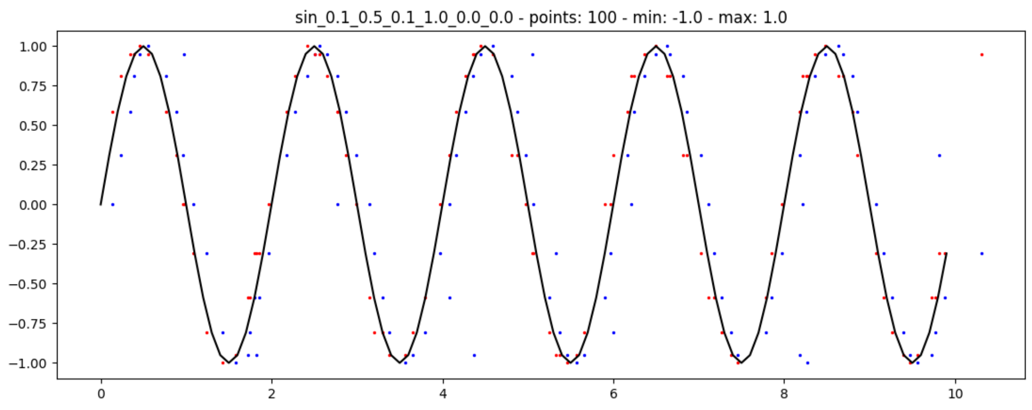

*Figure 6: A generated signal, the name of the signal includes the function type and the parameters used to generate it.*

[2] https://github.com/google-research/google-research/tree/master/tft
[3] https://github.com/greatwhiz/tft_tf2

remaining signals, they entire series was split into three equal sized and overlapping intervals. The first one included the first one third of the series, the second one included the middle third of the series and the final one included the final third of the series. Each split was randomly assigned to one of the three sets. By doing so we ensure we will validate and test our model with unseen data and avoid overfitting on the training data.

The signal values and the target were not scaled. The name of the signal was encoded using character level n-grams of sizes 1 to 4. The resulting encoding was then passed through SVD, singular value decomposition, to reduce the number of dimensions of the input vectors and to avoid the problems of sparce data. It was found that the first 8 values of SVD were sufficient to be used. Finaly the values from the SVD were further scaled using the hyperbolic tan function with a scaling factor. This unsured that the values will fall in the range of -1 and 1. It was found that this helped the performance of the model especially in signals where the amplitude was very low compared to the singular values.

## 8. Model and Hyperparameter tuning

The repositories include Tensorflow's Hyperparameter optimizer. After a few trials, we tuned the space of hyperparameters to include the values found in Table 2 under "training hyperparameters"**Fehler! Verweisquelle konnte nicht gefunden werden.**. To restrict the search space, we trained the model on a small number of samples and using various number of configuration parameters and only those value that showed potential were left in the final training.

For the fixed parameters used in the model training we used the values found in Table 2 under "fixed parameters"**Fehler! Verweisquelle konnte nicht gefunden werden.**. Note that the total time steps denote the number of time steps included in one sample. The number of encoders steps denote the number of timesteps the encoder will first encode before it generates a result. This configuration sets the model to forecast up to 30 timesteps ahead. Finally, 20K samples were used to train a model each epoch and 4K were used to evaluate the model.

| **Training Hyperparameters** | | **Fixed parameters** | |
|---|---|---|---|
| dropout_rate | [0.05, 0.1, 0.2, 0.5, 0.9] | total_time_steps | 50 |
| hidden_layer_size | [10, 20, 40, 80, 160, 240, 320] | num_encoder_steps | 20 |
| minibatch_size | [64, 256] | num_epochs | 50 |
| learning_rate | [1e-4, 1e-3, 1e-2] | early_stopping_patience | 10 |
| max_gradient_norm | [0.01, 1.0, 100.0] | multiprocessing_workers | 8 |
| num_heads | [1, 2, 3, 4] | svd_dims | 8 |
| stack_size | [1] | | |

*Table 2:The set of hyperparameters and fixed parameters used in the tuning of the TFT model.*

## 9. Results and Observations

Major findings:

- The best validation loss was 0.769. with p50 score equal to 0.251 and the p90 score was 0.108.
- The scores were calculated for all the timesteps the model was set to forecasting.
- The model has shown the potential to reduce latencies up to 1.5 seconds, well beyond the latencies that were measured.
- From the post analysis of the training data, it is clear that there is plenty of room for improvement.

The initial findings are promising. The best validation loss was 0.769. The p50 score for the best model was 0.251 and the p90 score was 0.108. The scores were calculated for all the 30 timesteps the model was set to forecasting. In a scenario where each timestep is approximately 50 milliseconds, the model can help reduce latencies up to 1.5 seconds, well beyond the latencies measured between Germany and Greece, a distance of roughly 1500Km.

In Figure 8 and Figure 7, the sharp spikes indicate a new training section starting with a new set of hyperparameters being used for the model configuration. We can see that for some configurations the loss was following the expected curve but not reaching a plateau at the end, indicating that the model could be trained even further. The sadden spikes which are close by indicate that the model hit an unstable configuration, and the early stopping mechanism kick in to stop further training, thus avoiding losing time on a bad set of parameters. Another think we can see in the graphs is that the training and validation losses follow similar trends and have values remarkably close to each other, indicating that the model was not overfitted to the training data and that it can handle new unseen data that was not part of the training set. Furthermore, in some cases the later values of the signal were used to train the model and the former values to validate it. This indicates that the model has been well adapted to the set of signals and can extrapolate both forward and backwards in time.

We need to note that the model was not extensively trained. It is planned to further train the model, but the initial results are very promising and show clear signs of potential use of the model in reducing the latencies in cloud based HiL testing.

An important observation emerged after training the model. In certain cases, the latency value exceeded the timestep value. When latency is simply added to the timestep, it can result in a slight reordering of the signal values, potentially disrupting the temporal sequence. This issue needs to be addressed in future iterations, as it contributes to a degradation in the model's overall performance.

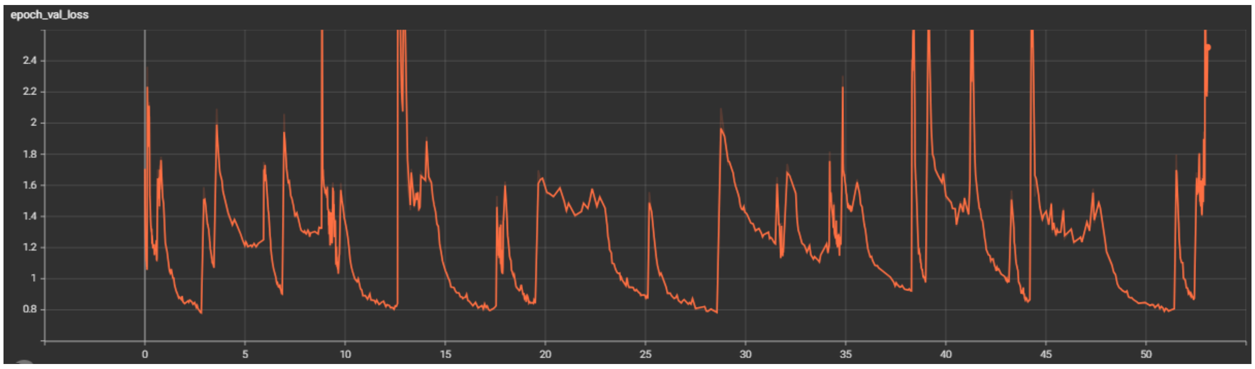

*Figure 7: Validation loss between hyperparameter iterations.*

Correctly aligning signal values with their respective timesteps is essential to ensure the integrity of the forecasting process and improve the model’s accuracy.

## 10. Conclusions

This study demonstrates the potential of Temporal Fusion Transformers (TFTs) for reducing latency in cloud-based HiL testing. Inspired by the success of TFT models in interpretable time series forecasting, particularly in demand forecasting, this research adapts the approach to address latency challenges in distributed automotive testing systems.

While the results are promising, there is significant room for improvement through more extensive training and hyperparameter tuning. A notable limitation of this study is the unaddressed issue of irregular time series data, where timesteps do not follow consistent intervals. Future research needs to explore modifications to the model's architecture, including adjustments to its final layers, loss function and output structure, to better manage such irregularities.

Additionally, the integration of other data sources, such as traffic load in simulated CAN networks, should be investigated to enhance model performance. Addressing these challenges and further refining the proposed approach could unlock transformative possibilities for remote testing, enabling efficient, scalable, and reliable cloud-based HiL systems for the automotive industry.

# I. List of Figures

# II. List of Tables