SOFDCAR CONSORTIUM

WHITEPAPER

„ML algorithms-based signal boosting based on synthetic automotive CANbus data"

Published by: **P3 digital services GmbH**

Authors: **L. Shianios, L. Limley, F. Laye, P. Ambrosch, M. Juschkin**

Issue date: **November, 24th 2023**

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

1.1 About SofDCar project

The Software-Defined Car project (SofDCar) is a partially publicly funded research program. It is sponsored by the German Ministry of Business Affairs and Climate Control (BMWK) and focuses on the challenges of future electric/electronics and software architecture in vehicles.

1.2 Motivation for HiL in the Cloud (HiC)

The strong upward trend in offered SW-enabled functionalities in cars increases the complexity of testing and validation significantly. The idea of the “HiL in the Cloud” (HiC) concept (also referred to as HiC platform concept) is to be able to connect electronic control units (ECU) location-independently from each other to a performant communication network. This would help to reduce shipping times for ECU to testbenches and therefore increase the possible maximum number of iteration cycles per development phase. Beside that cost savings with regards to shipping cost and developer capacities are a future potential of the concept.

The target of the research project is to assess to which extent the HiC concept can deliver reliable testing results. Real-time criticality is one focus point of the research. The ideal case would be that the complete network setup behaves just like the wired connection of a vehicle bus (as in regular “hardware in the loop” setups). As additional factors regarding signal latencies become relevant in the communication network, various approaches, and methods to increase real-time “capability” are investigated using different machine learning algorithms. As a basic demonstration case a test build-up with one transmitting and one receiving ECU was chosen.

The message broker of the web system through which all signals pass detects increased latency to the transmitting ECU. Pretending the delayed signal would be only static and transmitted cyclically, it is a relatively simple task to transmit it to the receiving ECU and check in hindsight if the transmitting ECU would have sent the message correctly and in time. Now most signals inside a car are not just static and transmitted cyclically. Therefore, the system must listen to the whole CANbus for a certain amount of time and cluster the signals into a variation of formulae using machine learning. At this stage our research does not cover all signal cases (e.g. sudden events like a crash) by the current approach – yet the researched concept has potential to ease testing and validation for a variety of basic functionalities significantly.

1.3 System architecture

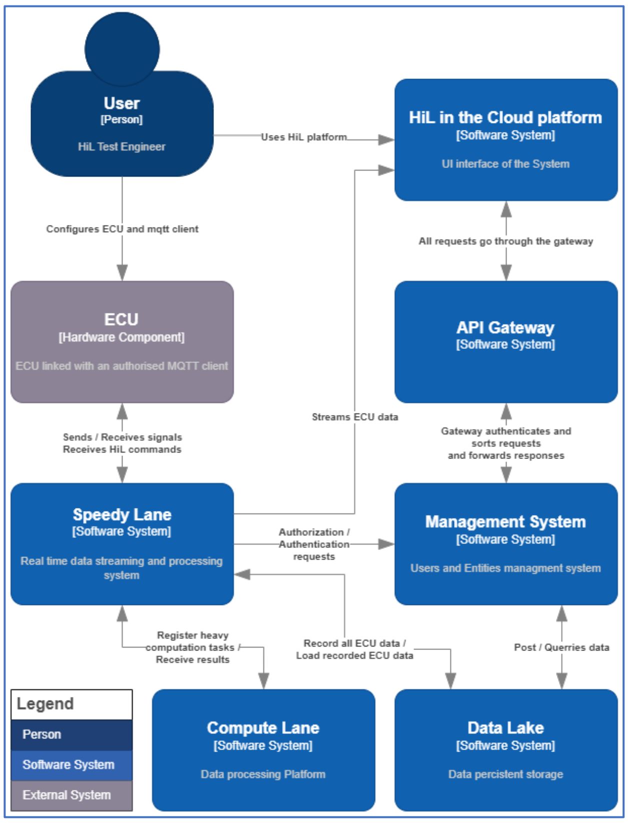
In Figure 1: High level diagram of HiL in the cloud. The deployment diagram of the HiC platform is presented – the entire system is composed from several microservices composed to larger systems. The users interact with the system through the HiL in the Cloud platform. All the ECU generated data is going through the ‘Speedy Lane’ of the system and all computations on the data are executed in the Computer Lane. The management platform handles all the requests coming from the HiL in the cloud platform and interacts with the Data Lake where all data is stored.

Ray is used as the main compute engine for the Computer Lane. Ray is an open-source unified framework used to scale AI and Python applications. It can be used to [1]

- parallelize ML model training,
- hyperparameter optimization and tuning of ML models,
- scaling the deployment of models in production,
- can be used as a distributed systems engine.

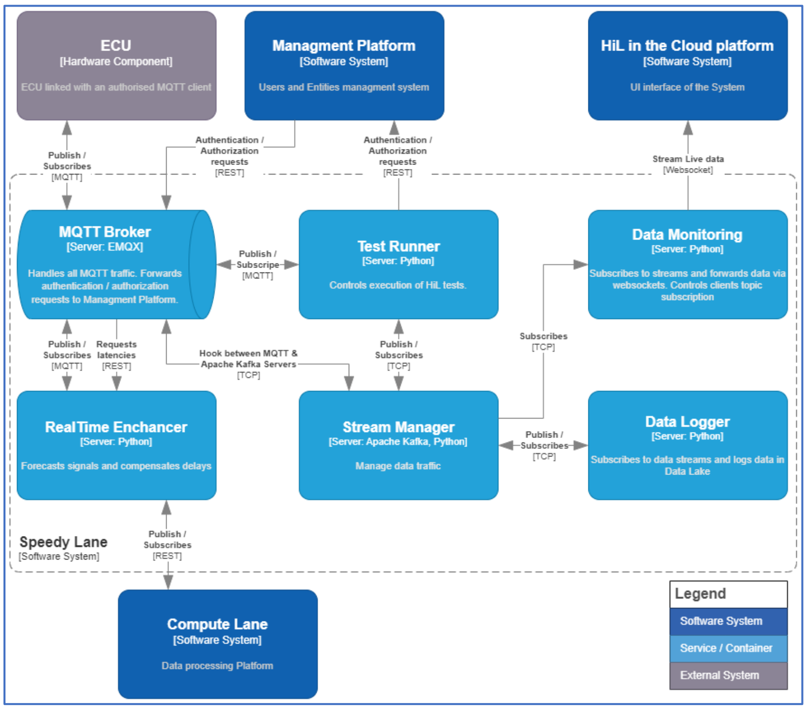
The real time enhancer is deployed as a separated service with three main components.

- An mqtt client that subscribes to the MQTT broker, to fetch messages exchanged between ECUs and the Test Runner.
- The latencies handler, which it sends on regular intervals requests to the EMQX's "slow subscription statistics" [2] model to retrieve the latencies of the ecus.
- The signal booster, that is the actual ML model, is being fed the messages from the client and the latencies from the handler. The output of the model is published to the EMQX broker via the mqtt client.



User
[Person]
HiL Test Engineer
Uses HiL platform
HiL in the Cloud platform
[Software System]
UI interface of the System
Configures ECU and mqtt client
All requests go through the gateway
ECU
[Hardware Component]
ECU linked with an authorised MQTT client
API Gateway
[Software System]
Streams ECU data
Sends / Receives signals
Receives HiL commands
Gateway authenticates and sorts requests and forwards responses
Speedy Lane
[Software System]
Real time data streaming and processing system
Authorization / Authentication requests
Management System
[Software System]
Users and Entities managment system
Register heavy computation tasks / Receive results
Record all ECU data / Load recorded ECU data
Post / Querries data
Legend
Person
Software System
External System
Compute Lane
[Software System]
Data processing Platform
Data Lake
[Software System]
Data persistent storage

Figure 1: High level diagram of HiL in the cloud.



ECU
[Hardware Component]
ECU linked with an authorised MQTT client
Managment Platform
[Software System]
Users and Entities managment system
HiL in the Cloud platform
[Software System]
UI interface of the System
Publish / Subscribes [MQTT]
Authentication / Authorization requests [REST]
Authentication / Authorization requests [REST]
Stream Live data [Websocket]
MQTT Broker
[Server: EMQX]
Handles all MQTT traffic. Forwards authentication / authorization requests to Managment Platform.
Publish / Subscripe [MQTT]
Test Runner
[Server: Python]
Controls execution of HiL tests.
Data Monitoring
[Server: Python]
Subscribes to streams and forwards data via websockets. Controls clients topic subscription
Publish / Subscribes [MQTT]
Requests latencies [REST]
Hook between MQTT & Apache Kafka Servers [TCP]
Publish / Subscribes [TCP]
Subscribes [TCP]
RealTime Enchancer
[Server: Python]
Forecasts signals and compensates delays
Stream Manager
[Server: Apache Kafka, Python]
Manage data traffic
Publish / Subscribes [TCP]
Data Logger
[Server: Python]
Subscribes to data streams and logs data in Data Lake
Speedy Lane
[Software System]
Publish / Subscribes [REST]
Compute Lane
[Software System]
Data processing Platform
Legend
Software System
Service / Container
External System

Figure 2: Deployment diagram of Speedy Lane of HiL in the cloud platform

A basic interaction between the components of the speedy lane is presented in Figure 2.

Kubflow is part of the compute lane of the system, and it is responsible to handle the deployment of the machine learning (ML) models on the Kubernetes cluster and managing the resources they need both during training and inference time.

Some of Kubflows main features are [3]:

- built-in integration with Jupyter notebooks, one of the main IDEs data scientists and machine learning engineers use to explore data and experiment with ML models,
- supports multi-user isolation,
- workflow orchestration,
- identity-based authentication and authorization via Istio integration.

Usually, Kubflow, as with Kubernetes, is used via a distributor like Azure, AWS, or Google Cloud, but it can also be installed in self-hosted servers.

MLFlow can also be part of the Compute Lane, and its main responsibilities are [4]:

- tracking ML experiments,
- packaging ML code for reusability,
- providing a centralized model store.

1.4 Summary

- The aim of this R&D project is to test whether HiL testing can be conducted over the cloud.
- Strict latency requirements make this a challenging problem.
- Machine learning and time series analysis may help battle latencies and enhance the real-time abilities of the system.
- Kubflow is used as the resource management system for ML components.
- Ray is used as the distributed and parallel compute engine for the system.
- MLFlow can be used to keep track of experiments and a centralized model repository.

2. Definition of the test set

2.1 Selected data

At its core, a car is an extraordinarily complex construct, both mechanically and digitally. In modern vehicles, numerous sensors and microprocessors work in a sophisticated interplay to perform the most diverse tasks – from monitoring and controlling mechanical functions to analyzing and processing large volumes of data. Every switch, sensor and actuator continuously generate data that is collected, transmitted, and processed within the vehicle system. At the same time, the vehicle communicates with external sources, exchanging information with other vehicles, traffic infrastructure and possibly even global databases / cloud applications. This increasing data volume generated and processed by a modern vehicle makes it possible to analyze and understand complex patterns and correlations. For example, driving behavior can be optimized, safety can be increased, or fuel consumption is minimized.

In this overly complex data landscape managed within a car, advanced algorithms and artificial intelligence are used to analyze vast amounts of information and gain useful insights. Impressively, all this is done within a compact and mobile unit such as a car. The level of complexity involved in the design and manufacturing of a modern car is a testament to the state of the art in engineering, computer science and data analysis. To manage the complexity of a modern vehicle, we need to streamline our focus for this paper.

Specifically, we will concentrate research on sensor data and actuators. These components are vital as they generate data and perform physical actions in the vehicle. Understanding the interplay between these two types of components is essential as it sets the foundation for our analysis. Predictive analytics uses the data provided by sensors to anticipate future vehicle performance or issues. Our narrowed focus allows us to model and analyze how sensor inputs trigger actuator responses and anticipate the responses. This streamlined approach can help with early fault detection, preventive maintenance, and improving overall vehicle efficiency. By reducing the complexity to focus on sensor data and actuators, we can explore these potentials in a clear and concise manner.

Sensor	Signal type	Function type	Behavior
Speed Sensor	Continuous	$v(t) = V_{max} \times \sin(2\pi f t)$	Periodic
Temperature Sensor	Continuous	$T(t) = T_{start} + k \times \log(t+1)$	Logarithmic, Linear (over certain temperature range)
Torque Sensor	Continuous	$\tau(t) = \tau_{start} \times e^{kt}$	Exponential
Acceleration Sensor	Continuous	$a(t) = A_{max} \times \sin(2\pi f t)$	Periodic, Linear (at constant acceleration)

Light Sensor	Discrete	$L(t) = round(\|\sin(\pi f t)\|)$	Periodic, Steady State (at constant light)
Ultrasonic Sensor	Continuous	$U(t) = U_{max} \times \|\sin(2\pi f t)\|$	Periodic, Steady State (at constant distance)
Tire Pressure Sensor	Continuous	$P(t) = P_{start} e^{kt}$	Exponential, Steady State (at constant pressure)
Oxygen Sensor	Continuous	$O(t) = O_{start} + k \times \log(t+1)$	Logarithmic, Steady State (at constant O_2 level)

Table 1: Sensors and their signals

To pursue this focused analysis effectively, we have defined a synthetic dataset that considers specific sensors and actuators. This dataset allows us to isolate and analyze the interactions between the sensor inputs and actuator outputs, while eliminating the influence from other data that is not relevant for our current research purpose. The synthetic dataset is a representation of complex real-world data in a simplified form, providing a practical tool for examining the relationships between sensor data and actuator responses. This is a crucial step towards building an effective model, and it will be the basis of our further investigations in this paper.

In Table 1, you can see eight chosen sensors with their mathematical signal descriptions. Each of these sensors plays a pivotal role in the modern vehicle's operation and safety. Let's delve deeper into their significance.

- **Speed Sensor**: This sensor measures the vehicle's speed, which is crucial for various systems such as the anti-lock braking system (ABS), cruise control, and traction control. Its periodic behavior reflects the variations in speed as a car accelerates and decelerates.
- **Temperature Sensor**: Monitoring the temperature is essential for engine management. It ensures that the engine operates within optimal temperature ranges, preventing overheating and potential damage.
- **Torque Sensor**: Used primarily in electric power steering systems, this sensor helps in adjusting the steering effort based on the torque applied by the driver, ensuring smoother and more responsive steering.
- **Acceleration Sensor**: This sensor is vital for stability control systems. It detects rapid changes in vehicle direction, allowing the car's computer to make necessary adjustments to prevent skidding or rollovers.
- **Light Sensor**: Integrated into the vehicle's automatic headlight system, it detects ambient light levels and automatically turns the headlights on or off, enhancing safety during dusk, dawn, or in tunnels.
- **Ultrasonic Sensor**: Commonly used in parking assistance systems, these sensors detect obstacles around the vehicle, providing feedback to the driver or even automating parking in some advanced systems.

- **Tire Pressure Sensor**: Monitoring tire pressure is crucial for both safety and fuel efficiency. A sudden drop in pressure can be indicative of a puncture, while consistent low pressure can lead to increased tire wear and reduced fuel economy.
- **Oxygen Sensor**: This sensor monitors the oxygen levels in the exhaust gases, allowing the engine management system to adjust the air-fuel mixture for optimal combustion. This not only ensures efficient performance but also reduces harmful emissions.

The mathematical models provided for each sensor in the table are simplifications of the real-world signals these sensors produce. However, they capture the essential characteristics and behaviors of the actual data. By focusing on these sensors and their interactions, we can gain a comprehensive understanding of the vehicle's operation. These sensors were chosen not only for their individual importance but also because, collectively, they provide a holistic view of the vehicle's functioning. In essence, the selected sensors and their mathematical representations offer a balanced and representative snapshot of the myriad of sensors found in a modern car. By studying these, we can approximate the behavior and interactions of the broader sensor network in real-world scenarios, setting a solid foundation for our analysis in the subsequent sections of this paper.

Table 2 is a selection of actuators presented, each with their respective mathematical signal descriptions. Actuators are devices that convert energy (typically electrical energy) into motion or physical action. In the context of a vehicle, they play a crucial role in executing the commands generated by the vehicle's control systems based on the data received from sensors. The significance of each actuator listed is detailed in the following:

Actuator	Signal type	Function type	Behavior
Electric Motor (Window)	Continuous	$P(t) = P_{max} \times \lvert\sin(\pi f t)\rvert$	Periodic, Steady State (at constant window position)
Servo Motor (Steering)	Continuous	$A(t) = A_{max} \times \sin(2\pi f t)$	Periodic, Linear (at constant steering motion)
Relay (Lighting Control)	Discrete	$R(t) = round(\lvert\sin(\pi f t)\rvert)$	Periodic, Steady State (at constant light state)

Table 2: Actuators and their signals

- **Electric Motor (Window):** This actuator controls the movement of the car windows. The periodic behavior of its signal reflects the up and down motion of the window, with the steady state indicating a constant window position, either fully open, fully closed, or any position in between.
- **Servo Motor (Steering):** A critical component in advanced steering systems, the servo motor assists in steering the vehicle. Its periodic behavior can be attributed to the left and right motions of the steering, with a linear behavior observed when the steering motion is constant.

- **Relay (Lighting Control):** Relays in lighting control systems act as switches, turning the vehicle's lights on or off based on inputs from sensors (like the light sensor) or manual controls. Its periodic behavior indicates the switching on and off the lights, while the steady state suggests that the lights remain either constantly on or off.

The mathematical models provided for each actuator in the table are designed to capture the fundamental characteristics of how these devices operate in real-world scenarios. While they are simplifications, they effectively represent the essential behaviors and interactions of the actual actuators. By examining these actuators and their mathematical representations, we gain insights into the physical actions and responses of a vehicle based on sensor inputs. These selected actuators, much like the sensors in Table 1, offer a representative view of the broader actuator network in modern vehicles. Understanding their operation and interactions is pivotal for our forecast analysis, as they directly influence the vehicle's performance and safety based on the data processed from the sensors.

It is essential to note that the mathematical models provided for both sensors and actuators in Tables 1 and 2 are approximations. While they capture the fundamental characteristics and behaviors of these components, real-world scenarios can introduce complexities and nuances that might deviate from these models. Factors such as wear and tear, external environmental conditions, and unforeseen interactions between components can lead to variations in actual behavior compared to our mathematical representations. As with any model, while they serve as valuable tools, they should be used with an awareness of their inherent limitations and the potential for real-world deviations.

2.2 Testing setup

Building upon the defined framework of sensors and actuators, we have identified a list of functions that are considered within our synthetic dataset. These functions are integral to our model as they represent the core actions and processes within the vehicle's operation that we aim to analyze. The selection of these specific functions is based on their direct relevance to the sensors and actuators we are focusing on, and on their impact on the overall performance and safety of the vehicle. This refined function list ensures that our dataset and subsequent analyses are tightly focused, enabling us to derive meaningful and actionable insights from our model.

Following the identification of these key functions, we have implemented function generators using the Python programming language, specifically utilizing the SymPy framework. These function generators serve as the mechanism for producing variations of each identified function, enriching our synthetic dataset with a wide range of potential scenarios. To ensure a comprehensive but manageable dataset, for each combination of functions, we generate a set of 250 distinct function instances. This size was chosen to strike a balance between data diversity and computational efficiency. Using SymPy allows us to model these functions effectively, given its ability to simulate the asynchronous behavior that is typical in real-world vehicle operation. By generating variations of each function, our dataset can cover a broad spectrum of potential vehicle states and responses, making our model more robust and versatile. Each of these function instances represents a possible scenario that the vehicle could encounter, enabling us to anticipate and prepare for a variety of situations in our modeling.

Signal type	Behavior
Continuous	Periodic
Continuous	Logarithmic
Continuous	Linear (over certain range)
Continuous	Exponential
Discrete	Periodic

Table 3: Relevant data types for HiC concept

2.3 Testing pipeline

In Figure 3, the process of data generation and preparation for subsequent validation is depicted. Data types are categorized as shown in Table 3. For each data type, there's a specialized function generator tailored for linear, periodic, logarithmic, or exponential functions. Notably, each function generator can produce a random assortment of functions meeting certain criteria. Every generator produces 250 distinct functions. Associated with each function type, there are time series generators. These generators convert the SymPy functions into time series data, producing 1.000 data points for each

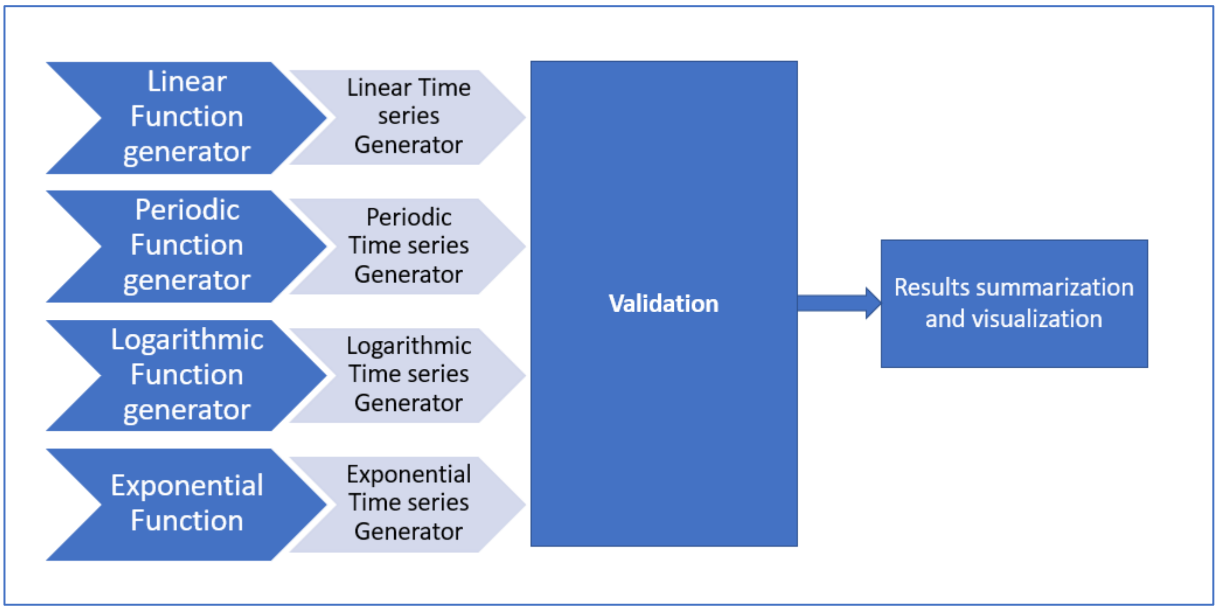

Linear Function generator
Linear Time series Generator
Periodic Function generator
Periodic Time series Generator
Logarithmic Function generator
Logarithmic Time series Generator
Exponential Function
Exponential Time series Generator
Validation
Results summarization and visualization

Figure 3: Data generation and evaluation pipeline

function. In total, the dataset comprises one million data points, which will be utilized for validating the machine learning models.

2.4 Handling signal noise

In real-world scenarios, raw data from sensors and actuators is rarely perfect. Signal noise, which refers to random variations or fluctuations that can distort a signal, is a common challenge when processing and analyzing data from physical systems. This noise can arise from various sources, such as electrical interference, thermal noise, or even external environmental factors. To ensure that our model is robust and can handle real-world imperfections, it is essential to account for and understand the impact of signal noise on our synthetic data. Building on the synthetic datasets described in sections 2.2 and 2.3, we have embarked on creating three new datasets, each infused with different levels of noise. This approach allows us to simulate the potential challenges and variations that might be encountered in actual vehicle operations. Specifically, we have introduced noise with standard deviations of 0.5, 1, and 2. These values were chosen to represent mild, moderate, and high levels of noise, respectively:

- **Mild Noise Dataset (Standard Deviation: 0.5):** This dataset represents scenarios where the vehicle's systems are operating in relatively ideal conditions, with minimal external interferences or disturbances. Such conditions might be typical in controlled environments or laboratory settings.
- **Moderate Noise Dataset (Standard Deviation: 1)**: Here, we simulate a more realistic operating environment, where typical external factors introduce a moderate level of noise. This dataset might resemble data from a vehicle operating in standard urban or suburban settings.
- **High Noise Dataset (Standard Deviation: 2)**: This dataset represents challenging conditions, where significant external interferences or system imperfections introduce a high level of noise. Such scenarios could arise in extreme weather conditions, areas with high electromagnetic interference, or when the vehicle's sensors or systems are nearing the end of their operational life.

To generate these noisy datasets, we utilized noise generation techniques, adding Gaussian noise with the specified standard deviations to our original synthetic data. This process ensures that our model, when trained on these datasets, can be tested against a variety of noise conditions, enhancing its reliability and versatility. In subsequent analyses, these noise-infused datasets will be instrumental in assessing the robustness of various machine learning models under different noise conditions. By examining how noise impacts the accuracy of these models, we aim to identify which models are best equipped to withstand the challenges of signal noise. This evaluation will help in selecting a model that consistently delivers reliable results amidst the uncertainties presented by diverse real-world scenarios.

2.5 Summary of test set

- A synthetic dataset was created since no sufficient real data set was available.
- The generated data simulates eight sensors and three actuators.
- Each sensor and actuator are represented by a class of mathematical equations.
- The signal equations capture the essential characteristics of the sensors and actuators but are simplifications of the real signal.
- The sensors and actuators were chosen so that a holistic view of a vehicles is generated and have a balanced and representative snapshot of the signals exchanged in a vehicle.
- The eleven classes were classified into 5 categories.
- For each category 250 expressions were generated, with their parameters chosen randomly.
- A function generator first generates an expression and then based on the expression a time series with 1.000 data points is generated.
- The dataset is enriched by adding noise to the generated time series, making the data more realistic.
- Three different levels of noise were introduced, simulating mild, moderate and high levels of noise.

3. Machine Learning methods

3.1 Selected methods

The realm of machine learning offers a vast array of models tailored for time series analysis. Given the intricacies and challenges of time series analysis, it is imperative to choose models that can capture patterns, trends, and seasonality effectively. In this paper, we have narrowed our focus to a selected few models, each with its unique strengths and challenges. Below is a brief overview and comparison of these models in terms of runtime and complexity.

Within the domain of machine learning for time series analysis, we have selected a subset of models for evaluation (see Figure 6), focusing on runtime efficiency and model complexity.

We have employed a linear regression model utilizing the, adhering to a minimalist configuration. Our findings indicate that while linear regression demonstrates expedient computational performance due to its elementary nature, it is constrained by its linear paradigm when modeling the multifaceted patterns often present in time series data.

Prophet, a model introduced by Facebook, has been implemented in its standard form, without additional customizations for seasonality or holidays. The model exhibits moderate computational demand and offers an intuitive framework for handling seasonal variations in time series data, maintaining an equilibrium between simplicity and the ability to address time series complexities.

Our approach with Long Short-Term Memory (LSTM) networks involved developing two distinct architectures tailored for continuous and discrete data types. LSTMs are characterized by a moderate runtime and elevated complexity, capable of discerning intricate data patterns and proving effective even with limited data quantities.

Gated Recurrent Units (GRUs) were selected for their computational efficiency and substantial modeling capability. The GRU model is designed with a ReLU activation function, optimizing performance through the Mean Squared Error loss function and the Adam optimizer. GRUs display a quicker runtime relative to LSTMs and possess a moderately complex architecture.

Our instantiation of the Autoregressive Integrated Moving Average (ARIMA) model concentrates on its three principal constituents: the autoregressive, integrated, and moving average components. The default parameterization of the model is intended to reflect a single differencing pass and a moving average of the previous observation, with the provision for parameter adjustments based on the time series dataset in question.

Lastly, the Seasonal Autoregressive Integrated Moving Averages (SARIMA) model is implemented with both non-seasonal and seasonal order parameters. SARIMA's runtime varies from moderate to high, contingent on the dataset's structure and the employed sampling methods.

In summary, our comparative analysis, conducted under predominantly default settings, reveals that deep learning models such as LSTMs and GRUs are adept at capturing complex patterns but are hampered by longer runtimes. In contrast, simpler models such as linear regression offer rapid computation but may not effectively handle complex datasets. Intermediate models like Prophet and ARIMA present a harmonized solution, with Prophet being notably accessible and ARIMA requiring intricate parameter tuning due to its statistical framework.

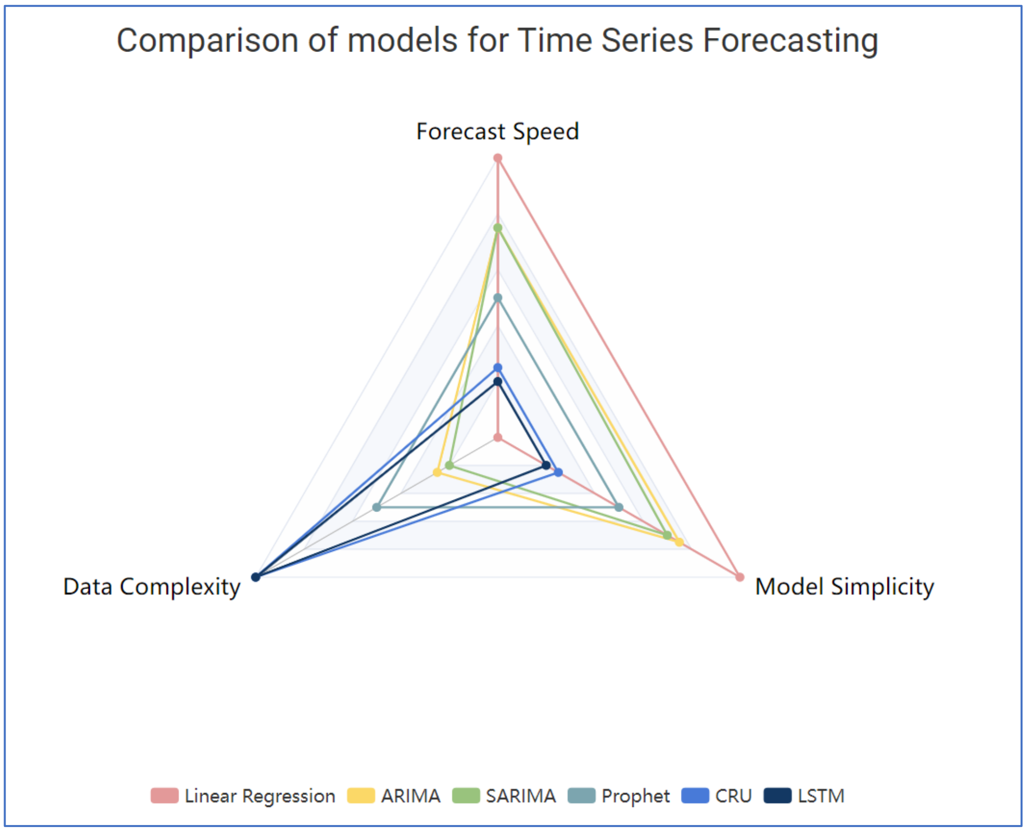


Figure 4: Comparison of models for time series forecasting

In Figure 4 models used and evaluated with the five categories of functions are compared. While all models were validated across every data type, SARIMA was an exception. SARIMA is specifically tailored for periodic and logarithmic data, making it challenging to apply to other data types.

3.2 Summary methods

- Six different models were considered.
- Linear regression is the simplest model, with the lowest run time, but its simplicity makes it incapable of handling even slightly complex datasets.
- Prophet offers a well balance between speed and complexity, it can handle quit complex datasets and generate results fast.
- LSTM is the slowest of all models and the most complex one, it can handle very complex data at the sacrifice of speed.
- GRU is very similar to LSTM but with slightly less parameters, it can handle the same data as LSTM but is a bit faster
- ARIMA models are faster the neural networks like LSTMs and GRUs, but slower than linear regression. It can handle quit complex data but not as complex as LSTMs or GRUs
- SARIMA it is very similar to ARIMA with an extract term to handle seasonality in the dataset, but this term makes it very difficult to use with datasets that do not show any periodicity.

4. Findings & conclusions

4.1 Validation metrics

Three different metrics were used to validate models' signal boosting ability.

- **R2 Score:** It measures how well the output matches the actual data. Higher scores indicate better fit but can suggest overfitting with many features.

- **Dynamic Time Warping:** Measures dissimilarity between two varying-speed temporal sequences, that is time series with data points on different points in time. Lower distances indicate greater similarity between the series. Needs a distance metric, like the Euclidean distance, to compute.

- **Compression-Based Distance:** Measures similarity between two time series using data compression techniques. Smaller distances suggesting higher similarity. Particularly useful when there is no defined distance metric between the two series. It is less computationally demanding than DTW

4.2 Optimization of the hyperparameters

Employing grid search, we meticulously determined the ideal hyperparameters, ensuring our models' peak performance and establishing a robust basis for accurate results.

ARIMA

Tuning of the ARIMA model revealed optimal hyperparameters with differencing, autoregressive, and moving average orders all set to 0, aligning with our data's traits. This suggests we need to curry further analysis.

LSTM

Hyperparameter optimization for the LSTM model concluded with a batch size of 16, 1 dense unit, 10 epochs, mean squared error for loss function, and 10 LSTM units using the Adam optimizer.

GRU

The GRU model was fine-tuned through rigorous optimization, leading to a batch size of 16, 1 dense unit, 10 epochs, mean squared error loss, and 10 GRU units with the Adam optimizer for optimal results.

4.3 Validation by data type

Linear data

With linear data GRU model demonstrated superior performance, achieving a near-perfect R2 score and optimal DTW values; ARIMA also fares well in CBD results, while other models display a range of high and low values (see Figure 5 and Figure 6).

Periodic data

With periodic data, which mirrors common real-world patterns, GRU and LSTM models stand out, showing commendable R2 scores. CBD metrics are generally low, indicating similarity, whereas DTW metrics are high, suggesting differences that merit further research (see Figure 7 and Figure 8).

Logarithmic data

Logarithmic data, often seen in real-world scenarios, presents challenges; however, ARIMA and SAMIRA models show strong performance with continuous data. The GRU model is particularly adept at handling discrete data.

Exponential data

Characterized by consistent relative growth, exponential data poses challenges to work with. Still, ARIMA and GRU models demonstrate a better grasp of this data type, as reflected by more accurate R2 scores, although DTW metrics indicate a high variation across all models (see Figure 11 Figure 12).

4.4 Validation by signal noise

As highlighted in Chapter 2.4, the foundational dataset underwent modifications with varying intensities of noise. The magnitude of this noise can be quantified by the standard deviation; a larger value indicates increased noise. The subsequent illustrations provide insights into how these models manage such data disturbances, offering a perspective on their robustness in handling data with several levels of noise.

Figure 5 and Figure 6 depict the decline in performance. The Y-axis percentage indicates the extent of performance reduction relative to the preceding less-noised dataset. It is evident that the GRU model

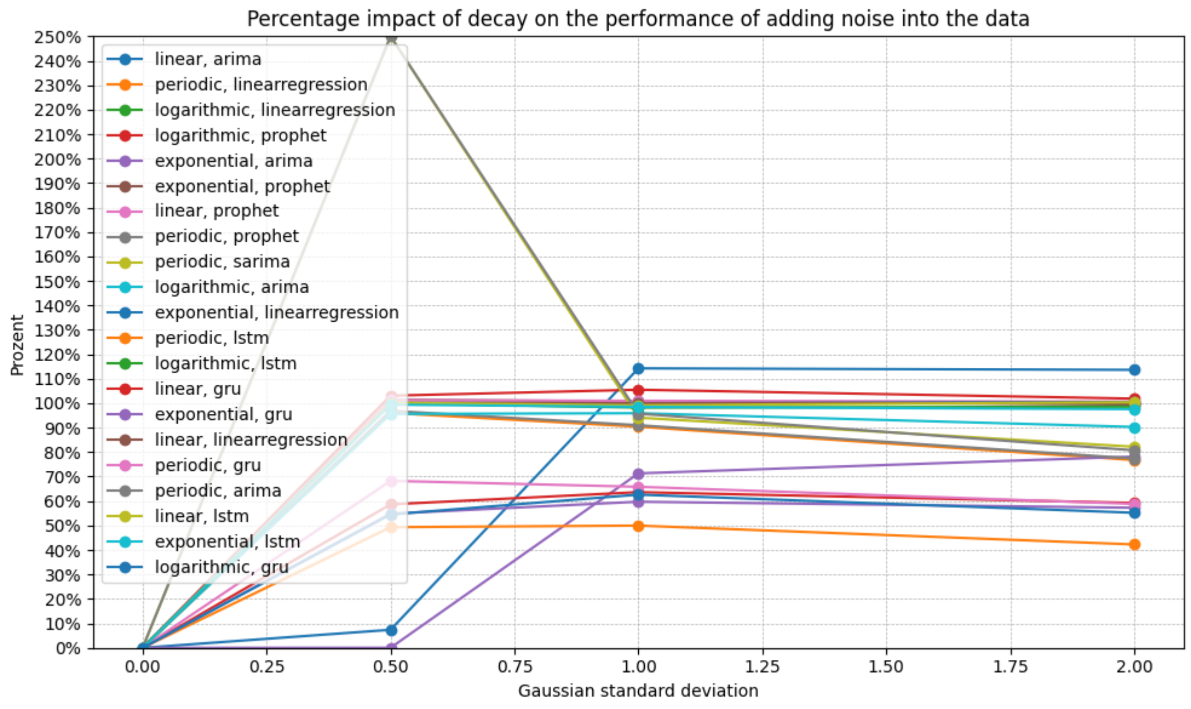


Figure 5: Effect of noise in discrete data on model performance

experiences a smaller dip in performance compared to other models. However, the results remain varied, with no single model excelling consistently across all data types.

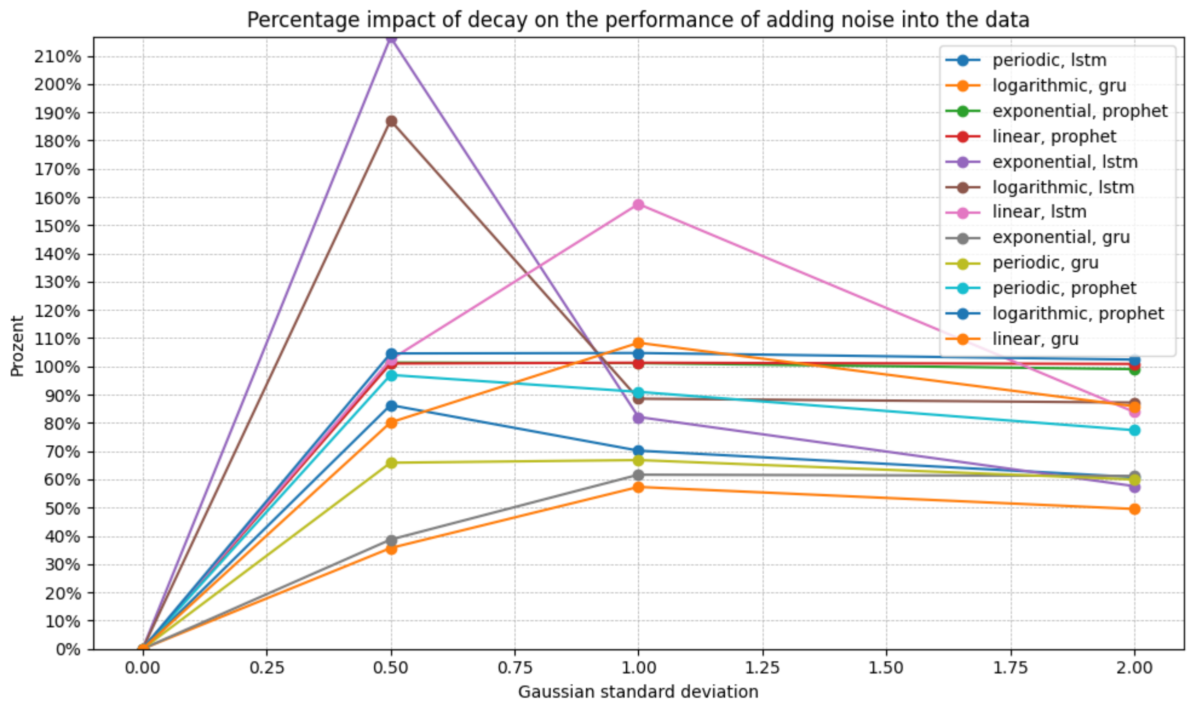


Figure 6: Effect of noise in continuous data on model performance

4.5 Ability of real time boosting

Assessing real-time boosting capabilities, we rigorously tested the LSTM, ARIMA, and GRU models, focusing on the immediacy and accuracy of their responses to new data. This aspect is critical for their potential adoption in fast-paced scenarios demanding timely decisions.

Our study measured the minimum time intervals required by each model to generate 100 data points with accuracy. The LSTM model required a minimum of 2.80 milliseconds (ms), while the GRU model needed at least 2.60 ms for the same task, reflecting its slightly more efficient structure in some cases.

Remarkably, the ARIMA model's performance was superior in speed, demanding only 0.04 ms to accurately generate 100 steps ahead, demonstrating the model's agility and potential suitability for rapid-response situations.

The evaluation highlights ARIMA's speed but also prompts consideration of the trade-offs between quick output generation, precision, and complexity. Each model's real-time boosting potential was gauged to provide insights for scenarios where speed is paramount.

Our findings offer a comprehensive view of how each model performs without specific fine-tuning,

representing their 'out-of-the-box' capabilities across various conditions. While the GRU model shows promise in noise resilience and accuracy, the ARIMA model's quickness is unmatched, potentially beneficial in situations where time is critical.

Model	Accuracy	Robustness to noise	Speed of Output generation	Real Time ability
Linear Regression	Only fair results with linear data	Poor performance with noise	Fastest model	Not tested
Prophet	Mixture of results, consistently low R2 score, but good DTW and CBD in some cases	Moderate performance with noise	Moderate speed	Not tested
ARIMA	Good with continuous data	Worst performance with noise	Fast output	0.04 ms for 100 outputs
SARIMA	Only good results with continues periodic data, but low R2 score	Poor performance with noise	Fast outputs	Not tested
GRU	Consistently good results with all data types. Only R2 score sometimes was low	Best performance with noisy data	Good speed	2.6 ms for 100 outputs
LSTM	Good results with most data types and metrics	Very good performance with noise data	Good speed	2.8 ms for 100 outputs

Table 4: Summary of results

In Table 4 the summary of results is presented. The GRU neural networks are a very good candidate for the real time enhancer, since they have shown high accuracy in a variety of data types and robustness to noise, with reasonable speed. In

4.6 Chapter summary

- Three metrics were used to evaluate the models, R2, Dynamic Time Warping (DTW) and Compression-Based Distance (CBD), on a variety of data with and without noise.
- In the validation of the models with noise free data the GRU shows consistently satisfactory results in all three metrics, while having a moderately good runtime.
- The ARIMA model is the fastest model to generate results.
- When noise is added to the data GRU is affected the least of all models

5. Real-time boosting capability of HiL in the Cloud web system

5.1 Way forward

We are expecting to implement the “real-time” functionality within the upcoming months and present towards the end of the SofDCar consortium a working minimum viable product (MVP).

5.2 Challenges of real-time achievement

Time series analysis is a complex field on its own, serving time series models in production and having to handle a variety of signals simultaneously is even more complex. Adding to the mix, the requirement of using the output of a model to enhance the real-time abilities of a system, makes it an incredibly challenging problem.

First, we need to understand how the model and its output values are going to be used and integrated into the rest of the system. In general, the model needs to generate the values of a given signal before the generator of the signal outputs the value. One of the challenges here is with the initial output that needs to be generated ahead of the real values when there is no data available to feed to the model.

Second, we need to clearly define the requirements for the generating model. The model needs to generate its result well ahead of the original signal, to battle the latencies of the overall system (network, web). To achieve this target, we need to measure the latencies between various components of the system and have the required response time of each participating component.

As an example, let us consider the case where we have Alice and Bob communicating with each other via a message broker. Alice is generating response signals based on Bobs request signal and vice versa. Alice has the requirement of receiving a response signal from Bob after sending a request signal within $r_a\ ms$, and Bob needs to receive its response within $r_b\ ms$. Alice communication line with the broker is a bit faster than Bob’ s line and has an average up stream delay of $u_a\ ms$, whereas Bob’ s average up stream delay is $u_b\ ms$. Alice downstream delay is $d_a ms$ and Bob’s downstream delay is $d_b ms$. The broker needs on average $p\ ms$ to receive, process and transmit messages.

When Alice sends a request signal to Bob the total latency for Alice will be $l_a = u_a + p + d_b + u_b + p + d_a\ ms$ which is the same for Bob as well. This is demonstrated in Figure 7 One thing to note here is that if $l_a < r_a$, then the actual response signals are received within the required latency time and there is no need to use real-time enhancement. But if not, then there are two things we can do. One, we can try and generate Alice request signal before Alice sends it. Two, we can generate Bob’s response signal before Bob generates it.

If we generate Alice signal, then from the moment the generated signal is created, it will take $p + d_b + u_b + p + d_a\ ms$ for Alice to receive the response signal from Bob. If the ML model generates its output at time, t_g then the response signal will arrive at Alice at $t_{a^{rs}} = t_g + p + d_b + u_b + p + d_a$. If Alice generates the request signal at time, $t_{a^{rq}}$ then Alice will expect to receive the response by $t_{a^{rq}} + r_a$. For the response signal to arrive in time, the inequality $(t_{a^{rq}} + r_a) > t_{a^{rs}}$ must hold. We can substitute $t_{a^{rs}}$ with the variables affecting its value and get $(t_{a^{rq}} + r_a) > (t_g + p + d_b + u_b + p + d_a)$.

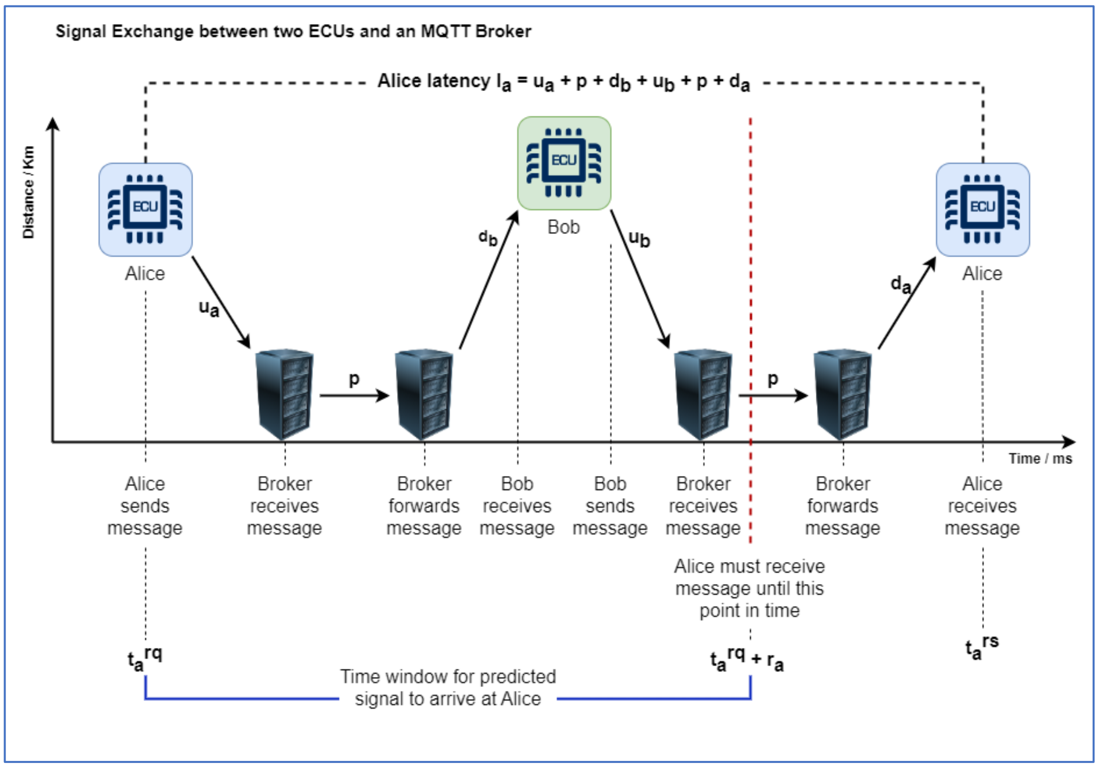


Figure 7: Signal Exchange between two ECUs and an MQTT Broker

In Figure 7 signal exchange between two ECUs and an MQTT Broker is demonstrated. The horizontal axis represents the time dimension, and the vertical axis represents distance relative to the Broker.

If we rearrange the inequality, we can get $(t_{a^{rq}} - t_g) > (p + d_b + u_b + p + d_a - r_a)$, where $H_a = t_{a^{rq}} - t_g$ is the forecast horizon for the signal booster model. This is demonstrated in Figure 8.

Let us consider the following example, if $p = 10ms, d_b = 100ms, u_b = 150ms, d_a = 50ms, r_a = 100ms$, then $H_a > 220ms$, which means that the signal booster needs to generate Alice signal at least $220ms$ ahead of Alice. If we also take into consideration the time the model needs to generate its result, call it t_m, then the model horizon will need to be $H_a + t_m\ ms$. If Alice is generating signals at regular intervals of $i\ ms$, then the number of steps ahead that the signal booster should generate would be $S = (H_a + t_m)/i$.

There are two challenges at this point. One is if H_a is high or t_m is high, or both and i is small, then the number of steps the model needs to generate ahead is very large, making it a long-horizon problem. In this case most probably a very different model will be needed from the one that can be used for short horizons. In general, the error in the generated values grows as the horizon increases, so having a model that is faster will reduce the required forecast horizon. Even if we minimize the time the model needs to generate its outputs, the fact that the network delays between Alice, Bob and the broker cannot be controlled by the system and can randomly change i.e., H_a can vary uncontrollably, raises the requirement to have at least two different models in production. One that can handle short horizon outputs and one that can handle long horizon outputs, and depending on the overall latency of the system the appropriate model is used.

The second challenge is that we need to generate the boosted signal at the right moment in time so that the response arrives to Alice, when Alice will be expecting it, and not sooner in time when Alice

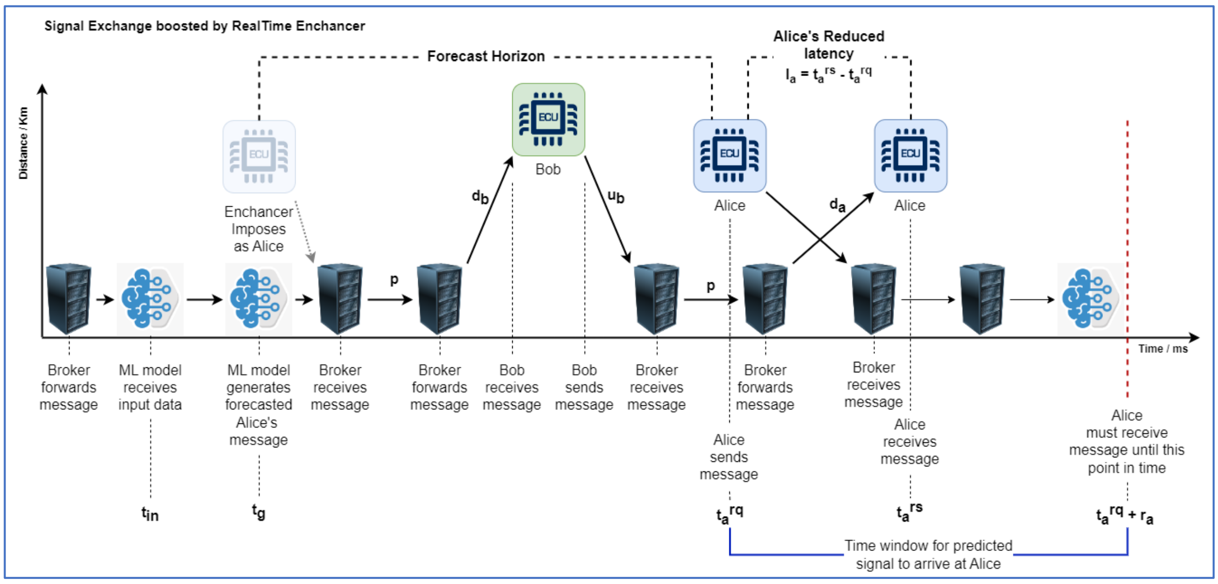


Figure 8: Signal exchange boosted by real time enhancer.

will be expecting the response of another signal. This means we need to closely monitor all the latencies of the system and make the model able to generate the boosted signals at the right instance in time, and not in predefine intervals. Another possible solution to this is to make the model generate the timestamp along with the signal value, but training such model will be more difficult. We can even try to train a Temporal Fusion Transformer which have been shown to be able to handle these kind of problems [5].

Another challenge we might face is the problem of Alice's signals arriving at the broker at irregular intervals. Even if Alice is generating her signals at regular intervals, the network delays may vary, and the perceived periodicity of Alice signals might be irregular. If the delays variation is insignificant, we can ignore this, but if they are not, then it is something we need to take into consideration when designing the final model and the mechanism that will feed it with data in production.

A final challenge is the number of models that might need to be deployed. If we try to have one model for each signal that goes through the system, then we will need to train new models for every new signal introduced in the system. Thus, we will need to have a large repository of models and keep track which model is responsible for which signal. Eventually a continues delivery pipeline of trained models will be needed, where a new model is trained as soon a new signal is introduced to the system. Finally, as soon as we detect a signal joining the system, we need to deploy its corresponding model automatically. All this will need a considerable amount of orchestration and careful resource allocation. Another option to this problem is to have fewer models, able to boost a range of different signals. Although this might reduce the complexity of serving and maintaining a large pool of models in production, it is more difficult to train such generic models. Most probably a mixture of both strategies will yield the best results, but this must be investigated.

As a final comment, we mention that we could generate Bob's response signal before Bob generates the response and send it to Alice. This is not as viable solution as generating Alice request signal. First, this is the same behavior as the one we have from the virtual ecu (v-ecu) module, where a model receives input signals, and based on these signals it generates a response signal. It does not generate future value of the inputted signal. Second, generating responses based on past values, without having the latest input signal value, it is extremely difficult. The underlying mechanism of the receiving module must have some correlations with past input values, and often this is not the case, especially if we have state machines where the state is affected by each incoming signal.

In conclusion using time series analysis to enhance the real-time capabilities of a system, and implement it in production environments, needs extensive analysis and thorough planning. The final solution needs to be a careful balance between model accuracy and model performance. Furthermore, the model needs to be flexible enough to handle a variety of situations arising from the uncontrollable variations in the latencies of the overall system.

5.3 Chapter Summary

- There is the problem of initial results, where the deployed model will need to generate outputs when there are no prior data to feed it with
- The model needs to generate its result well ahead of the actual signal, to eliminate latencies, but not too far ahead so that it does not mix up the order of signals exchanged between ecus.
- The output horizon is affected by the delay between all components, the latency requirements of the ecus, as well as the speed at which the model generates its results.
- The latencies between the system and the ECUs can vary, so the system must closely monitor and feed the latencies values to the model, to control when the output is generated.
- Deploying for every signal a separate model is probably prohibited, thus signals must be categorized.
- For each category of signals different boosting models will be needed, to handle short, moderate, and long latencies, separately.
- Alternatively, a model must be versatile and robust enough to handle varying latency times.
- Trying to generate response signals after a trigger signal, will not help much with reducing latency times.

References

[1] T. R. Team, „Overview,“ 2023. [Online]. Available: https://docs.ray.io/en/latest/ray-overview/index.html. [Zugriff am 11 August 2023].

[2] EMQX, „Slow subscribers statistics,“ emqx.io, 2023. [Online]. Available: https://www.emqx.io/docs/en/v5.1/observability/slow-subscribers-statistics.html#how-it-works. [Zugriff am 11 August 2023].

[3] R. Liul und W. Chiang, „Building a Machine Learning Platform with Kubeflow and Ray on Google Kubernetes Engine,“ 24 September 2022. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/build-a-ml-platform-with-kubeflow-and-ray-on-gke. [Zugriff am 11 August 2023].

[4] M. Project, „MLflow Documentation,“ MLflow.org, 2023. [Online]. Available: https://mlflow.org/docs/latest/index.html. [Zugriff am 11 August 2023].

[5] S. O. A. N. L. T. P. Bryan Lim, „Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting,“ 19 December 2019. [Online]. Available: https://arxiv.org/abs/1912.09363.

[6] L. K. J. &. P. R. Cox, „Balancing Quality and Confidentiality for Multivariate Tabular Data. 87-98.,“ 2004.

[7] S. Ellis, „Instability of least squares, least absolute deviation and least median of squares linear regression, with a comment by Stephen Portnoy and Ivan Mizera and a rejoinder by the author. Statistical Science 13, 337-350,“ Statistical Science, 1998.

[8] L. M. A. P. M. G. A. B. A. M. C. &. M. M. Menculini, „ Comparing Prophet and Deep Learning to ARIMA in Forecasting Wholesale Food Prices. ArXiv, abs/2107.12770.,“ 2021.

[9] A. N. D. &. L. S. Shewalkar, „Performance Evaluation of Deep Neural Networks Applied to Speech Recognition: RNN, LSTM and GRU. Journal of Artificial Intelligence and Soft Computing Research, 9, 235 - 245.,“ 2019.

[10] T. G. M. &. B. H. Boulmaiz, „Impact of training data size on the LSTM performances for rainfall–runoff modeling. Modeling Earth Systems and Environment, 1-12,“ 2020.

[11] A. J. V. M. C. &. B. M. Benterki, „Long-Term Prediction of Vehicle Trajectory Using Recurrent Neural Networks. IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, 1, 3817-3822.,“ 2019.

[12] S. H. Y. Z. S. H. J. W. G. Z. M. &. L. Q. Gao, „Short-term runoff prediction with GRU and LSTM networks without requiring time step optimization during sample generation. Journal of Hydrology, 589, 125188.,“ 2020.

II. List of figures

III. List of Tables