Modelling processing latencies with machine-learning techniques and comparing to classical methods

Christoph Funda\textsuperscript{1}, Fabian Imkenberg\textsuperscript{2}, Tobias Konheiser\textsuperscript{2}, Thomas Herpel\textsuperscript{2}, Gordon Pipa\textsuperscript{3}, Reinhard German\textsuperscript{4}, and Kai-Steffen Hielscher\textsuperscript{4}

\textsuperscript{1}Test System Development ZF Mobility Solutions GmbH
\textsuperscript{2}Test System Development ZF Mobility Solutions GmbH Ingolstadt
\textsuperscript{3}Department of Neural Science
\textsuperscript{4}Department Computer Science University Erlangen-Nuremberg

January 29, 2024
Abstract—This paper describes an application of a machine learning based method and compares it to classical methods for input modelling of software processing latencies for a discrete-event simulation model of a hardware-in-the-loop test bench. We apply and compare the following four different methods for input modelling: stochastic distribution functions, phase-type distributions with Markovian state changes, phase-type distributions modelled by an advanced Markov-chain, and a machine learning linear regression approach based on recurrent neural networks. We compare the different model results with the measured data using different metrics. We use two common metrics used in machine learning: the mean absolute error and the root mean square error. And we introduce two new metrics inspired by the service curve concept of network calculus. We compare the cumulative sum of the latencies describing a service curve visually and with two metrics. We discuss the advantages and disadvantages of the models for our use case. The resulting models can be used to evaluate the performance and improve the design of a hardware-in-the-loop streaming system. We focus on the playback buffer backlog and the end-to-end latency, under the challenging conditions that the underlying data does not fulfil the independent and identically distribution assumption.

I. INTRODUCTION

Hardware-in-the-loop (HIL) simulation is a widely used approach within the industrial development process of mechatronic systems including electronic devices. The HIL is part of the development process and is used to validate and verify the functionality and robustness of mechatronic systems, including software and hardware.

The development of HIL test systems requires performance evaluation and validation. HIL systems require a high quality of service (QoS) in the form of a high data integrity and no data loss. The key performance metrics are the end-to-end latency and the buffer backlog.

However, HIL test systems are complex systems and there is little or no discussion of performance evaluation methods and how they compare. In [1] and [2] we introduced a discrete event simulation (DES) model of the HIL system and an analytical model based on network calculus (NC) for streaming systems. These methods are highly dependent on the processing times measured on the HIL system. Even a statistical experiment with simple stochastic distributions using the DES model requires unimodal stochastic distributed data. However, we have also found multimodal distributed data in our measurements. These data can be described by phase-type distributions in combination with Markov-chains as discussed in [3] and [4]. However, Markov-chains are memoryless and are usually applied under the independent and identical distribution (i.i.d.) assumption of the data.

In this work we answer the following research questions:

• What methods, other than direct measurements, can be used for the input modelling of processing latencies in a DES when the data do not fulfill the i.i.d. assumption?
• How well do classical methods based on stochastic distribution functions and Markov-chains perform compared
to machine learning (ML) based models, applied to non i.i.d. data?
- What metric can be used to compare and evaluate the processing latency prediction methods that best describe the HIL use case?

Our confirmed hypothesis is: A ML model can be built and works as well as conventional models, even for non i.i.d. data.

For this purpose, we have developed and trained a machine learning based model and compared it with models based on classical methods. Processing latency measurements from the application layer are used to train the model. The processing latencies themselves depend on the input workload (cycle-time and payload in bytes), but also on software (SW) and hardware (HW) dependent mechanisms such as central processing unit (CPU) processing or memory allocation. For this purpose, we developed the machine learning model and compared it with classical methods as a proof of concept for future extensions, e.g. CPU utilisation estimation.

As a classical model, we use stochastic distributions with their extension of clustering data into phases and use Markov-chains to switch between these clustered states. We furthermore extend the classical Markov-chain approach with an extended Markov-chain introducing deterministic and stochastic state changes to reproduce the pattern in the data and to overcome the memoryless property of the Markov model. The aim is to incorporate stochasticity into the DES model using different modelling techniques.

II. Methods

A. Terminology

There is no official IEEE definition of HIL as mentioned in [6], nor in the Automotive SPICE [7]. However, the author of [6] proposes the following definition for HIL systems: “Hardware-in-the-loop system is a non-intrusive test approach, containing physical controller connected in open- or closed-loop with virtual or semi-virtual subsystems, providing faithful physical replicas of the real world and evaluating the System under test in either black/grey/white box manner.” This definition provides a comprehensive overview of HIL systems.

We also use the term cycle-time, which is commonly used when dealing with sensors. It is the same as the interarrival time of messages or packets in the network community and its reciprocal is known as the rate.

A playback buffer is used to meet real-time (RT) and QoS requirements. It is placed at the end of the HIL streaming chain before the packets are sent to the device under test (DUT) to ensure correct playback of the recorded data.

The pre-buffer time or pre-buffer delay is the time that is used to fill the playback buffer in the HIL PC before starting to stream in real-time to the DUT.

ROS stands for Robot Operating System, which is a middleware SW for communication between robot operating system (ROS) nodes written in C++ and used in this system under study on the HOST PC. LabVIEW is a graphical programming language from National Instruments and is used in our system under study on the HIL PC.

Service curves are a concept from NC, which is the mathematical framework for deterministic delay and backlog bound calculation based on the min-plus algebra invented by Cruz in 1991 [8]. Rate-latency service curves are a basic linear approximation of the service described by a worst-case latency and a constant rate function.

Furthermore, we will use the following abbreviations for the two phase-type distribution Markov-chain models defined later:
- 2-state markov-chain model (2MM)
- 15-state markov-chain model (15MM)

B. Metrics to compare the different methods

To describe the accuracy of the model in predicting processing latency we propose to use the following quantitative metrics. The following classical machine learning metrics are used to compare the model result with the measured target result, for each data point:

- Root-mean-square-error (RMSE)

\[ RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\text{Predicted}[i] - \text{Actual}[i])^2} \]  

- Mean-average-error (MAE)

\[ MAE = \frac{1}{N} \sum_{i=1}^{N} |\text{Predicted}[i] - \text{Actual}[i]| \]  

The following two metrics are designed for our special use-case. They are inspired by the rate-latency service curve concept of NC.

- Cumulated-latency-error (CLE)

\[ CS_{Actual}[i] = \sum_{k=1}^{i} \text{Actual}[k] \]  

\[ CS_{Predicted}[i] = \sum_{k=1}^{i} \text{Predicted}[k] \]

\[ CLE = \max_{i \leq N}((CS_{Predicted}[i] - CS_{Actual}[i]), 0) - \min_{i \leq N}((CS_{Predicted}[i] - CS_{Actual}[i]), 0) \]

The CLE calculates the maximum deviation between the predicted cumulative sums (CSs) and the measured CSs. The latency values are accumulated to describe a flow over time, inspired by the NC concept of service curves. The CLE represents the deviation of the service curve latency between model and measurement. Compared to the classical metrics, it is the long-term behaviour of the flow that is represented by this metric.

- Mean-rate-error (MRE)

\[ MRE = 1 - \frac{\frac{1}{N} \sum_{i=1}^{N} \text{mean}(\text{Predicted})}{\text{mean}(\text{Actual})} \]
With the MRE we calculate the relative error of the model’s predicted mean-rate values to the actual mean-rate values from the measurements. This metric is also inspired by rate-latency service-curves and represents the deviation of the rate between model prediction and measurement as a target.

C. Input models for processing latency generation

1) Phase type distributions with Markov-Chains for switching the states: One method of modelling the input data for the simulation model is to use phase-type distributions. This method uses multiple distributions and a Markov-chain to represent and generate data. The basic method of phase-based distributions is used and applied by Hielscher [4] and described theoretically by Buchholz et al. [3].

In our use-case, processing times or latencies are used as an input. This data needs to be clustered to get distinct groups. This can be done using algorithms such as k-means or clustering by upper and lower limits. The number of groups can vary according to the general distribution of the input data. For each group of data different types of distributions are fitted. The best fitting one will be used to represent that group. The goodness of fit can be assessed using, for example, the Kolmogorov-Smirnov-Test (KS-Test).

As the input data are described by multiple distributions, a Markov-chain is needed to switch between them. Each distribution is represented by a state, and the probabilistic state transitions are calculated empirically from the clusters in the data. In addition, an initial state must be specified.

To obtain a new value from the model, a random distribution value is generated from the distribution correlating with the current state of the Markov-chain. A new state is then randomly chosen according to the transition probabilities, which represents the memoryless property of the Markov-model. If there is non i.i.d. data and the new system state depends on the previous system state, the Markov-chain can be extended to include deterministic state-changes.

2) Machine-learning model: Predicting processing latencies in a HIL test system using a machine learning approach is a rather specific task. Therefore, there is not much directly related work available. Nevertheless, there is work addressing, for example, on latency prediction in other fields that have similarities or could also be valid for the prediction of processing latencies in a HIL system.

Data-driven modelling approaches for dynamical systems using neural network architectures have become very popular in recent years as they have successfully solved a variety of tasks [9]. An example of a machine learning application for latency prediction is described in [10] [9]. They use classical feature extraction with a subsequent classifier to assign the latencies occurring in 4G mobile networks to a specific range.

Furthermore, there are several approaches that aim to simulate queuing systems with machine learning networks. Gorbunova et al. [11] propose a new method for predicting the average end-to-end delay of multi-phase queuing systems using a neural network in the form of a multi-layer perceptron. Another way of modelling queuing systems is introduced by Garbi et al. [12], who developed a RNN that directly learns the properties of the queuing system.

In this paper, a RNN, more precisely its modification LSTM, is used to predict processing latencies using machine learning. The main idea of RNNs is to introduce a sequential memory for each neuron [13]. Basically, a neuron receives an input and processes it into an output. This output should now not only depend on the current input, but also on previous inputs. Therefore, a loop connection from the neuron back to the same neuron is introduced. In this way, the neuron not only produces an output, but also passes information to itself for the next time step.

Although the RNN architecture was developed to handle sequential data, it has problems retaining information from previous time steps when processing long sequences [14]. This phenomenon arises from the nature of backpropagation, more precisely from the problem of vanishing gradients. The solution to the short-term memory of the classical RNN is the LSTM architecture [15] [16].

A LSTM has the same chain-like structure as the classical RNN, but instead of transmitting only one state over time, another state, the cell state, is added. The cell state is the key element of the LSTM architecture, as it can be considered as the implementation of a memory that counteracts the problem of vanishing gradients. This is achieved by applying small changes over time, such as adding or removing values to the cell state. The various alterations of the cell state and the computation of the hidden state are performed by so-called gates. The gates are responsible for selecting which information is read from, erased from, or written to the cell state and the hidden state. This allows information to be stored for long periods of time.

This allows the model to learn patterns from measured processing latency sequences that are difficult to find or analyse, to predict processing latencies, while taking previous data into account.

III. APPROACH

A. HI as the system under study

Our HIL system is structured as outlined in diagram 1.

The HIL system consists of a HOST PC, a HIL computer, and a device or system under test (DUT/SUT). The HOST PC is the master of the HIL system. It controls and configures the HIL system and is the interface to the user, usually a test engineer or a HIL system developer. The HOST PC is connected to the HIL computer via Ethernet. The HIL computer is a real-time computer connected to the DUT via a real-time communication interface.
The DUT is described in the definition of [6] as a physical controller. These are electronic devices or mechatronic systems that are connected to the HIL testbench and are part of the HIL system itself. In our use-case, the system under test (SUT) are high-performance computers for autonomous driving functions that test the robustness of the perception software in an open-loop approach at the HIL.

However, since we assume the communication interfaces to be RT capable, the DUT is out of scope in our performance evaluation.

Furthermore, there are two basic possibilities of operational use-cases or setups of the HIL, also described in [6]. One possibility is the open-loop approach, where measured or simulated data is streamed in RT to the DUT, and the output of the DUT is measured and compared to the ground truth. So, there is no feedback loop to the HIL computer, which means that the generation of input data to the DUT is independent of its output data. The other setup is the closed-loop approach, where such a feedback loop is implemented. In this paper we focus on the open-loop approach.

B. Measurements performed at the system under study

As a measurement system, a pure SW logger is used. As previously shown in figure 1, the structure and logging points are now shown in detail in figure 2.

![Fig. 2. Detailed conceptual model and software instrumentation](image)

The system under study consists of a HOST PC where the SW is ROS based and a HIL PC where the SW is based on LabVIEW. The SW is instrumented in the user space on both PCs, to measure the SW processing latencies in microseconds. As data arrives, a system timestamp is generated at each processing step. The timestamp is generated directly and placed in a separate worker queue. The meta-information is processed in a separate thread in parallel, to reduce the influence on the normal system behaviour. The measured SW processing latency consists of the processing time on the CPU and any memory access latency. We disabled the turbo boost features of the CPU and used core binding and core isolation to avoid any additional effects on the SW processes under study, such as interrupts or context switches.

C. Discrete event simulation model of a HIL system

The model is created using OMNET++ and the INET framework. According to the structure of the HIL system, servers, delayers, queues and a TCP connection are modelled. The inputs for the model are time values that are mainly used for processing times of the server modules. The inputs for the simulation model are set in OMNET++ in a parameter file (ini file). It is possible to set single values, read multiple values (from external files) or set distributions, where values are generated during run time of the simulation.

D. Input modeling with distributions

The simplest approach including stochastical behaviour for input modelling into the DES model are stochastic distribution functions. The processing times calculated from the underlying measurements are used to fit a distribution, which is used in the simulation model to generate random distribution values as processing times.

In our implementation we select the following distributions available in OMNET++:
- Exponential
- Normal
- Lognormal
- Gamma
- Beta
- Weibull_Min
- Uniform

We fit all the distributions to the data. To find the distribution that best describes the data, we perform the goodness-of-fit analysis described by Nurmi et al. in [17]. This consists of the following two steps: First we create a sample from the measurements, then we perform the KS-Test on the distribution and the sample data. This test gives a p-value that describes the goodness of fit. We loop through this test 1000 times and calculate the mean of the p-values for each distribution. For the input of the simulation model, we select the distribution with the highest mean p-value. To ensure valid processing times, the distribution values are truncated to non-negative values and values below the maximum data range. The input for the OMNET++ simulation model consists of the selected distribution with the corresponding parameters and the range limits. The processing time values are generated during the runtime of the simulation. This can also be done in a pre-processing step, where processing time traces are generated.

For data with multiple distributions or even patterns, the simple distributional model is no longer sufficient. Therefore, the following model with clustering in phases and individual distributions is introduced.

E. Input modeling with distributions switching phases with 2-state Markov chain model

The simple distributional approach uses a single distribution to cover the whole data range. This leads to the problem, that outliers or multimodal data are not well represented. With the advanced distributional approach, we improve the previously described method.

To cover outliers and anomalies as well as multimodal data, we change the input for a module from a single distribution to multiple distributions, which are alternated with a Markov chain. The number of states depends on the input data. We found two dominant groups in the input data, resulting in a 2MM with phase-type distributions, meaning individual distributions for each state. After plotting the processing times over time, a visual inspection revealed clearly separated clusters,
which we modelled on a more abstract level with two states and later in more detail.

In order to get the corresponding measurement data to fit a distribution of each phase, the original data is clustered into two phases. For each cluster of the data, a distribution is fitted as described above.

![Fig. 3. 2 state Markov chain](image)

The switching probability between the two states is derived empirically from the clustered data. The best-fitting distribution and the Markov-chain are shown in figures 3 and 4. Figure 3 shows the structure of the Markov chain with the corresponding state changes and transition probabilities that are calculated from the measured data. Figure 4 shows exemplary the number of occurrences and cumulative density function for the measured processing times plotted together with the fitted distribution. The lower two plots show the first cluster in detail and are a subset of the upper graphs.

![Fig. 4. distribution fit for 2MM](image)

F. Input modeling with phase-type distributions and pattern with 15-state Markov chain model

To switch between the states a Markov chain is used, which is constructed using the inherent pattern in the data, which is described in much more detail by the 15MM than by the 2MM. The state-change probabilities are derived empirically from the measurements according to the clustered data. We switch between the two processing time clusters in the data with the 15 state Markov chain according to the pattern we find in the data. Visual inspection and analysis of the plotted processing latencies over time reveals that the state changes within the first cluster are deterministic, whereas the state changes from the second cluster are stochastic. The processing times of the first state occur in groups of three to five. After these groups, one processing time from the second cluster is observed. The length of the deterministic processing time chain does not follow a clear pattern and therefore the transition probabilities from the second to the first cluster are calculated from the measured data. The complete 15MM is shown in figure 5.

![Fig. 5. 15 states Markov chain which describes the pattern in the latency data based on two clusters with deterministic and stochastic](image)

The algorithm was derived by empirical evaluation of data clustering and state-changes. It results in a simple algorithm that calculates the number of values between processing times belonging to state two. This results in the state change probabilities.

We use the same distribution functions for each system state of the model as in the previously described 2-state Markov chain. The results of the model compared to the measurements are shown in the following 6.

All plots show a high similarity between the model output and the measurement output. The overall behaviour represented by the three plots shows a high similarity between the model and the measurement data. The autocorrelation plot shows the non i.i.d. property of the data. The data point plot as well as the histograms show only minor differences. The detailed results are discussed later.

G. Detailed ML model approach

The design of the ML model using the LSTM architecture as its central component is depicted in figure 7. It serves as a linear regression model predicting the processing latency in form of a continuous value. The LSTM architecture is a commonly used approach to model time series data without the vanishing gradient problem.

![Fig. 7. Architecture of the linear regression model.](image)
The model is a multilayer RNN consisting of four stacked LSTM layers, each with an increasing number of units, corresponding to the layer width, starting with 16 and going up to 128. The input sequence of previous latencies has the dimension 15 x 1 due to the sequence length of 15, while the output dimension is simply 1 since it represents the continuous value of the processing latency. As activation function of the output layer tanh is used to provide the latency in the normalised target value range of [-1, 1].

Each LSTM layer takes a sequence of 15 consecutive previous latencies as input and outputs a sequence of the same length, but with an increased width, i.e., number of units per sequence step. The width of the sequence steps increases from layer to layer as the number of units does. In addition, the LSTM layers are bidirectional, meaning that each incoming sequence is not only fed into the layers in the forward direction, but also in the reverse direction. This is possible because the entire input sequence is available at once.

To enable the models to generalise the training progress for the application to unseen test data, measures against overfitting to the training data are established. Therefore, the LSTM layers are followed by a dropout layer with a dropout rate of 0.5 as proposed in [18].

Although the ML model contains some trainable parameters, the training and evaluation of the model was carried out without the need for special equipment. As with the other methods, the concept generation clearly outweighs the evaluation effort.

The test box plots for the predictions and their corresponding targets are shown in 8. In both box plots, the median is quite small and close to the boundary of the first quartile. Furthermore, the whiskers are so short that they are hardly recognizable, while the upper outliers range from 30ms to 50ms. As the box plots show, the linear regression model is generally able to learn the wide-ranging latency distribution with its large outliers. However, some latencies are incorrectly predicted to be in the range between 2ms and 30ms, where there are no target latencies. In addition, the highest outlier latencies of the predictions do not reach the maximum target of around 48ms.

IV. RESULTS

The models are compared with the previously defined metrics and presented in table I:
The RNN model performs particularly well when the classical metrics MAE and RMSE are compared with the other models. These metrics directly compare the prediction of each sequence step with the corresponding target. This is exactly what the RNN model is designed and optimised for. It therefore performs significantly better here than the other models, at least a power of ten better than the next best model. As some of the other methods do not aim to make the right prediction at the right time, their results for MAE and RMSE are worse. However, for the other metrics, CLE and MRE, which are tailored to our specific use case, the RNN model does not perform better than the other methods.

The classical phase-type distribution models (2MM and 15MM) and the RNN model derive comparable results with the service curve based metrics CLE and MRE. Comparing the MRE, the 2MM and the RNN models have a negative trend comparing the mean-rate, while the 15MM has a positive trend. These drifts can be observed in the following figure 9 on the right side, which describes the error between the cumulative sum of the model prediction and measured data. The 2MM has the highest deviation of 1s, around the mean-rate. However, comparing the following figure, the other two models, the 15MM and the RNN model, have a tendency to drift away from the mean value.

## V. Discussion

We have chosen challenging data with dependencies and multimodal distributions. The simple distribution model is not able to represent this data set, which is represented by the worse values of the metrics and what can be seen in 9. The left side of the figure shows the accumulated number of send messages over time. On the right side the cumulative sum of the prediction error can be seen. It can be clearly seen that the stochastic distribution model produces by far the largest error. The cumulated dataflow for measurement and model clearly drift apart. The other three models create predictions close to the target and result in an error in the same order of magnitude.

Therefore, an advanced model is needed to represent this dataset. The 2MM model performs well and is comparable to the more complex 15MM and RNN models. Furthermore, the 2MM model is easy to build up, compared to the other two models. The 15MM model is slightly better in most metrics than the 2MM model, but it requires much more effort to develop it and to adapt it to other data. The RNN performs best for the RMSE and MAE metrics. Nevertheless, the CLE and MRE metrics are more important for our particular use-case to represent the long-term behaviour of the flow. Furthermore, the RNN model has to be retrained to new data sets, what is more time consuming than recalculating the 2MM model. It is clear that non i.i.d. data can not be represented by a simple distribution well enough for our use-case.

The generation of the distribution values and the training of the RNN model include a random component, which needs to be considered. To be able to use a model for our DES, a close representation of the real measurements is needed. Some stochastic behaviour can be beneficial in some test cases. Short-term fluctuations, as seen in the 2MM, 15MM and RNN models are less important, because they can be compensated by the playback buffer. Long-term trends, on the other hand, can not be compensated for. However, long-term trends can not be excluded for all three models.

## VI. Conclusion

For the use-case of input modelling to discrete event simulation, the long-term behaviour of the flow is most important, which is represented by our metrics CLE and MRE. The 2MM model performs similarly to the more complex 15MM and RNN models. We would decide for the 2MM model in this use-case. However, the RNN model can be extended by adding further relevant inputs, such as byte-size and cycle-time of the input workload, which are difficult to capture in the classical Markov models. A possibility for the extension of the Markov-models is to estimate the parameters of the distribution function with a linear-regression, derived from measurements with different input workloads. Moreover, the RNN model can be extended with additional performance metrics of interest, like CPU utilisation, and thus could replace the DES model completely. With the phase-type distributional Markov-chain model, this can only be achieved in combination with a DES model including a scheduler and a CPU model. The shortcomings of the RNN model are its time-consuming development and optimisation, the training data generation, and its black-box model property. However, as the DES model is also time-consuming to develop and a ML based model could replace it, it is worthwhile to develop such a model to make performance predictions on CPU utilisation and processing latencies based on different input workloads. This is left for future work.

## ACKNOWLEDGMENT

This research was supported by ZF AG and ZF Mobility Solutions GmbH (a company of ZF group).
Fig. 9. comparing the four models with the measurement data based on the service curve approach

REFERENCES


APPENDIX

The HIL cluster consists of a HOST PC and a HIL RT-PXI. The HOST PC is equipped with a 2012 Intel Xeon E3-1230 V2 3.3 GHz processor (four physical CPU cores) and 16 GB system memory. All worker nodes are connected via 40 Gbit Ethernet in a single-switch star topology. Each node runs Gentoo Linux (kernel version 3.6.11) and Java 1.7.0.13. The HIL RT-PXIe-8880 is equipped with an Intel(R) Xeon(R) CPU E5-2618L v3 @ 2.30GHz (8 physical CPU cores) and 24 GB system memory. All worker nodes are connected via 40 GBit Ethernet in a single-switch star topology. Each node runs NI Linux Real-Time x64 4.14.146- rt67 and other NI LabVIEW Runtime and NI Drivers.