Works in Progress in Embedded Computing Journal

Special Issue on selected papers of
Works in Progress Session within 28th Euromicro Conference on Digital System Design (DSD) and 51th Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA), Salerno, Italy, Sep. 10th – 12th, 2025.

September 2025 · Volume 11 · Issue 1 · ISSN 2980-7298

Message from the Editor

It is my great pleasure to serve as the Editor of the WiPiEC Journal Special Edition on Selected Papers from the Works in Progress (WiP) Session, held within the 28th Euromicro Conference on Digital System Design (DSD'2025) and the 51st Euromicro Conference on Software Engineering and Advanced Applications (SEAA'2025), taking place in Salerno, Italy, September 10–12, 2025.

This is the 3rd Special Edition dedicated to DSD/SEAA conferences, which represent some of the longest-standing traditions in the fields of electronics, computer science, and the modern technologies that have evolved from them. This year, we are pleased to present 16 selected works covering a broad range of thematic areas.

The success of the WiP program would not have been possible without the valuable contributions of the authors, the enthusiastic support of the DSD'2025 and SEAA'2025 Committees, as well as the dedication of MECOnet, MANT, and the EUROMICRO staff, whose efforts made this edition possible. We are especially grateful to Jovan Djurković for his outstanding work on the technical editing and preparation of this publication. As in previous years, the Works in Progress in Embedded Computing Journal (WiPiEC) proudly publishes these selected papers in open-access format. The quality of the contributions is excellent, with many papers originating from leading universities and research centers around the world.

I am confident that the WiP Session will continue to grow in the future and become increasingly popular, particularly among young researchers, who play a vital role in shaping the future of our conferences.

Welcome to DSD'2025, SEAA'2025, and the Works in Progress (WiP) Session. Congratulations to all the authors whose papers have been selected for publication in WiPiEC Journal, Vol. 11, No. 1.

Prof. Dr Radovan Stojanović
University of Montenegro, Montenegro
General Chair of WiP Session within DSD'2025 and SEAA'2025
General Chair of WiPiEC Journal
https://wipiec.digitalheritage.me

Contents

Reliability-Aware Hyperparameter Optimization for ANN-to-SNN Conversion

Saeed Sharifian[1,*], Mahdi Taheri[2,3], Vahid Rashtchi[1], Ali Azarpeyvand[1,3], Christian Herglotz[2], and Maksim Jenihhin[3]
[1] University of Zanjan, Zanjan, Iran
[2] Brandenburg Technical University, Cottbus, Germany
[3] Tallinn University of technology, Tallinn, Estonia
* sharifian.sa@znu.ac.ir

Abstract—**Spiking Neural Networks (SNNs) have emerged as an energy-efficient alternative to Artificial Neural Networks (ANNs), particularly for edge-computing and safety-critical applications. Unlike conventional ANNs, SNNs leverage sparse event-driven processing to reduce energy consumption while significantly maintaining high computational efficiency. This paper presents a framework designed to optimize the conversion of ANNs into equivalent SNNs while balancing accuracy, reliability, and energy efficiency. The proposed framework systematically explores SNN hyperparameters to identify configurations that achieve superior performance compared to their ANN counterparts. Experimental evaluations on MNIST and Fashion-MNIST datasets with different network topologies demonstrate that the optimized SNNs achieve comparable accuracy while offering in some cases 27.81× and 15.17× lower energy consumption and 1.92× and 1.84× less accuracy drop in the presence of faults, respectively, over the ANN baseline. The results highlight the applicability of SNNs in reliability-critical power-constrained environments.**

Index Terms—deep neural networks, spiking neural networks, reliability, edge applications, safety-critical applications

I. INTRODUCTION

Spiking Neural Networks (SNNs) are gaining traction due to their bio-inspired processing, event-driven computation, and energy efficiency. Unlike Artificial Neural Networks (ANNs), SNNs operate similarly to biological neurons, making them well-suited for low-power edge devices and neuromorphic computing [1], [2]. Their sparse and asynchronous nature enhances computational efficiency and scalability, making them ideal for applications in autonomous systems and safety-critical environments [3], [4]. However, the temporal dynamics of SNNs both at the neuron and network levels, along with the non-differentiability of spike functions, have made it difficult to train efficient SNNs [5]. Different studies with various approaches have attempted to adapt backpropagation-based supervised learning algorithms to SNNs [6]. To overcome these challenges and leverage the effectiveness of ANN training, many methods focus on converting well-trained ANNs into functionally equivalent SNNs. One key challenge in transitioning from ANNs to SNNs is ensuring structural consistency between the two architectures [7]. Many deep learning models are highly optimized for specific tasks, and modifying their topology during conversion can result in accuracy loss, increased training complexity, and inefficiencies in hardware deployment. For instance, in edge AI applications like real-time image recognition for autonomous vehicles, maintaining the original ANN topology ensures that pre-trained weights and feature extraction mechanisms remain effective while benefiting from SNNs' energy efficiency [8].

Beyond training, ensuring the reliability of SNNs is critical, especially in noisy or faulty hardware environments where robustness is essential [9]. Several frameworks have addressed specific aspects, such as memory fault tolerance (e.g., ReSpawn [10], rescueSNN [11]), Fault Injection (FI) and analysis (e.g., SpikingJET [12], SpikeFI [13]), or energy-efficient architecture search (e.g., AutoSNN [14]).

Despite these advancements, there remains a lack of unified approaches that convert ANN to SNN, jointly considering reliability, energy efficiency, and accuracy with hyperparameter tuning.

To fill these gaps, this paper proposes an automated framework for generating optimal, reliable, and low-energy consumption SNNs from ANNs. The proposed framework generates SNNs with topological similarity to the original ANN while searching for optimal SNN configurations within them that have balanced accuracy, reliability, and energy consumption. The proposed algorithm utilizes only the ANN architecture to generate new SNNs, distinguishing it from conventional ANN-to-SNN conversion methods that aim to transfer learned parameters for SNN training. SNN networks are learned with the surrogate gradient method. By using FI scenarios, our method ensures that the generated SNNs maintain or exceed the performance of their ANN counterparts while significantly reducing energy consumption.

The key contributions of this paper are:

- A hyperparameter optimization-based technique to ensure a high-performance, high accuracy reliable SNN network
- An automated framework for optimized ANN-to-SNN conversion based on accuracy, reliability, and energy consumption
- Experimental validation on different datasets and network topologies demonstrating the energy, accuracy, and reliability trade-offs between ANNs and SNNs

Manuscript received May 20, 2025; revised July 31, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025.
Paper category: Regular
DOI: doi.org/10.64552/wipiec.v11i1.85

The proposed approach offers a practical and efficient pathway to leveraging SNNs in safety-critical power-constrained edge applications, making them viable alternatives to conventional ANN-based solutions.

The remainder of this paper is structured as follows: Section II presents the proposed methodology. Section III discusses experimental results and the comparison of the initial input ANN and the selected SNN. Finally, Section IV concludes the paper.

II. Proposed Methodology

The framework is developed using *PyTorch* for ANN implementation and *snnTorch* [15] for SNN implementation, both of which support GPU acceleration for training and inference. The *snnTorch* framework supports multiple spiking neuron models, with one of the most widely used being the Leaky Integrate-and-Fire (LIF) model [15] which was also used in this research. Equation (1) represents the discretized form of the LIF neuron's differential equation, which consists of three main components. The neuron's membrane potential is denoted as U. The input component is the product of the input vector X (a spike train of 0s and 1s) and the synaptic weights W. The decay term, governed by the decay factor β, causes the membrane potential to decrease at a rate of β per time step.

The neuron's threshold voltage is represented by θ, which ensures that when the membrane potential reaches a certain threshold, it resets to a predefined value, producing a spike at the output [15]. The spikes generated at the output are denoted as $S_{out}[t] \in \{0,1\}$. As described in (2), when $S_{out} = 1$, θ is subtracted from the membrane potential; otherwise, no reset occurs. This mechanism, known as the subtraction reset or soft reset mechanism, is widely used in spiking neural networks [15], [16].

$$U[t] = \underbrace{\beta . U[t-1]}_{\text{decay}} + \underbrace{W . X[t]}_{\text{input}} - \underbrace{S_{out}[t-1] . \theta}_{\text{reset}} \quad (1)$$

$$S_{out}[t] = \begin{cases} 1, & if\ U[t] > \theta \\ 0, & otherwise. \end{cases} \quad (2)$$

The *snnTorch* framework also supports multiple spike encoding schemes, such as rate coding and temporal coding. In this study, rate coding is employed, which converts input intensity into a spike count [15].

Since all networks in this study are bias-free, the energy consumption of ANN models is computed using the equation $\sum w.x$ where w and x represent the weight and input data, respectively. The computational operations required in ANN neurons consist of Multiply-Accumulate (*MAC*) operations, which can be theoretically estimated. The total energy consumption is then determined using (3).

$$E_{ANN} = TotalOperations \times E_{MAC} \quad (3)$$

For spiking neurons, computations follow (1). However, in this study, the decay factor β is set to approximately 1, allowing us to disregard its effect for simplification. Additionally, the accumulation term U in (1) is ignored. Consequently, the computational operations in spiking neurons primarily involve the summation of weights, represented as $\sum w$, corresponding to Accumulation (*AC*) operations.

To determine the number of operations in spiking neurons, this study employs a state-of-the-art technique that accurately measures computational complexity by counting the average number of spikes fired across the entire network. This method, which accounts for dataset characteristics, spiking neuron hyperparameters, and encoding schemes, has been widely adopted in recent research [17], [18]. Specifically, after applying the full dataset to the network, the number of spikes generated in each layer is recorded, and the average spike count per layer is reported. The total energy consumption is then estimated by incorporating this spike count into (4) [17].

Table I shows the energy estimation resulting from the implementation of a 32-bit multiplier and adder at 45nm CMOS technology according to reference [19]. Therefore, the E_{MAC} and E_{AC} in (3), (4) can be calculated using this table.

$$E_{SNN} = SpikeCount \times E_{AC} \quad (4)$$

TABLE I. Energy estimation of AC and MAC operation in 45nm CMOS technology

45nm Technology	Energy (pJ)	
	INT	FP
ADD	0.1	0.9
MUL	3.1	3.7
ACC	0.1	0.9
MAC	3.2	4.6

SNNs have multiple hyperparameters affecting their performance, such as the spiking neuron model, time steps, neuron threshold voltage, and neuron reset type [1], [15]. These hyperparameters significantly influence SNN performance, impacting accuracy, spike rate, and energy consumption [1], [15]. A critical factor in SNN efficiency is the selection of an appropriate time step. Higher time steps improve accuracy but increase spike rate, leading to higher latency and energy consumption. Conversely, lower time steps reduce latency but may degrade accuracy. Similarly, adjusting the neuron threshold voltage modifies spiking behavior, influencing both the learning and inference phases. At the same time, these hyperparameters play an essential role in the reliability of the networks. The learning process in this research was conducted using the surrogate gradient method supported by *snnTorch*. Since a key objective of this research is to identify a suitable network for edge applications, adopting an integer number format is crucial compared to floating-point representation. To achieve this, quantization is applied to convert network parameters into an integer format. Notably, the proposed framework supports quantization with arbitrary precision; however, in this study, an 8-bit integer format was used. Quantization improves efficiency in hardware implementations such as FPGAs and ASICs by reducing memory size and computational complexity. To assess reliability, the FI method [20] is used, employing Bit Error Rate

(BER) analysis to simulate faults. This allows systematic evaluation of model robustness without requiring exhaustive FI into all bits, thus reducing computational overhead. To model transient faults, the bit-flip FI method is employed, applying different BER to network parameters (weights) to simulate cumulative faults. The results are presented in terms of accuracy drop as an indicator of network reliability.

Considering the described matters, Fig. 1 provides an overview of the proposed methodology. The workflow consists of three steps designed to find an optimal SNN topologically equivalent to an ANN, maintaining efficient accuracy, reliability, and energy consumption.

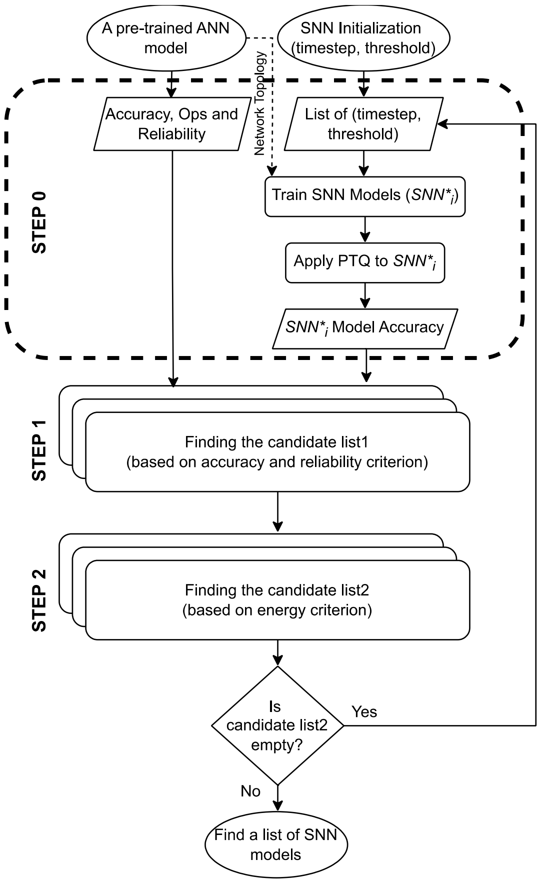

A pre-trained ANN model
SNN Initialization (timestep, threshold)
Network Topology
STEP 0
Accuracy, Ops and Reliability
List of (timestep, threshold)
Train SNN Models (SNN*_i)
Apply PTQ to SNN*_i
SNN*_i Model Accuracy
STEP 1
Finding the candidate list1 (based on accuracy and reliability criterion)
STEP 2
Finding the candidate list2 (based on energy criterion)
Is candidate list2 empty?
Yes
No
Find a list of SNN models

Figure 1. The proposed methodology flowchart

At first, a pre-trained ANN and a set of hyperparameters defining its equivalent SNN are input into the framework.

In **STEP 0**, according to the pseudo-code proposed in Algorithm 1, the framework trains a set of SNN models, denoted as SNN_i^*, using the input hyperparameters. After training, post-training quantization (PTQ) is applied, allowing the user to specify bit-width precision. The accuracy check is performed at the end of this stage on SNN_i^*. On the other hand, the accuracy, reliability, and total operations (Ops) of the ANN are also measured for comparison.

Algorithm 1: ANN-to-SNN Conversion Algorithm

```
Input  : A pre-trained ANN model with a certain topology, Dataset
Output: A set of the optimized SNN models or optionally just an
        optimized SNN model
STEP 0: Test ANN Accuracy, Ops, and Reliability, initialize the SNN
  architecture with a time step and threshold list;
Define $ann$ as the ANN architecture;
Define $snn$ as the SNN architecture;
Define $cur\_params$ as Current hyperparameters;
$\tau \leftarrow$ List of time steps;
$\theta \leftarrow$ List of thresholds;
$accuracy_{ANN} \leftarrow$ Accuracy($ann$);
$operations_{ANN} \leftarrow$ Calculate_ops($ann$);
$reliability_{ANN} \leftarrow$ Fi_test($ann$);
STEP 1: Find the candidate list1 of SNN network (accuracy &
  reliability criteria);
for $\tau_i \in \tau$ do
    for $\theta_i \in \theta$ do
        $cur\_params \leftarrow \tau_i, \theta_i$;
        $snn \leftarrow$ Train_model($SNN_i^*, \tau_i, \theta_i, bits$);
        $accuracy_{SNN} \leftarrow$ Accuracy($snn, \tau_i, \theta_i$);
        if $accuracy_{SNN} \geq accuracy_{ANN}$ then
            $reliability_{SNN} \leftarrow$ Fi_test($snn, \tau_i, \theta_i$);
            if $reliability_{SNN} \geq reliability_{ANN}$ then
                $candidate\_list1 \leftarrow candidate\_list1 + cur\_params$;
            end
        end
    end
end
STEP 2: Find a candidate list through $candidate\_list1$ (energy
  criterion);
for $snn\_items \in candidate\_list1$ do
    $cur\_params \leftarrow \tau_*, \theta_*$;
    $operations_{SNN} \leftarrow$ Calculate_ops($snn\_items, \tau_*, \theta_*$);
    if $operations_{SNN} < operations_{ANN}$ then
        $candidate\_list2 \leftarrow$
          $candidate\_list2 + cur\_params,\ operations_{SNN}$;
end
(Optional Part): Select lowest-energy solution from $candidate\_list2$;
$best\_item \leftarrow$ Min($candidate\_list2$ Ops);
if $best\_item \neq 0$ then
    PASS; Found a set of optimized SNN or (Optional) an optimized
      SNN config;
end
else
    FAIL; Change given parameters (Selecting the list of new
      hyperparameters);
end
```

In **STEP 1**, the accuracy of SNN_i^* is compared to that of the original ANN. If accuracy is maintained or improved, the model undergoes reliability assessment. Only configurations meeting accuracy and then reliability thresholds are stored in *candidate_list1*.

In **STEP 2**, The final step evaluates energy consumption. Hyperparameters such as τ_*, θ_* are items from the previous list that are met, so they are used in this stage. Configurations with lower energy usage than the ANN are stored in *candidate_list2*. If optimal networks exist in *candidate_list2*, the framework returns a selection of viable SNN models. Optionally, the user can request the lowest-energy solution. If no configurations meet the criteria, the input parameters must be adjusted again. Thereby, the algorithm back to the start of STEP 0 according to Algorithm 1, and using an automatic or manual mechanism the list of hyperparameters must be expanded or be selected in other ranges.

The framework systematically searches for an optimal SNN while ensuring minimal performance degradation. If a network from STEP 2 is selected, it is guaranteed to outperform the ANN in terms of energy efficiency and reliability while maintaining accuracy.

By applying the order used in checking accuracy, reliability, and energy consumption, many weak cases are eliminated in a short period of time. According to the experiments performed, the accuracy check of an SNN, depending on the selected hyperparameters and with the topologies chosen in this study, is usually under 3 seconds. However, a reliability test may take several minutes to complete. In STEP 1, all cases with unacceptable accuracy are eliminated, and neither reliability nor energy efficiency tests are performed on them. Also, for SNNs, the energy consumption estimation in this algorithm is calculated simultaneously with their accuracy test.

III. Experimental Results

This section presents the results obtained from the proposed framework and its evaluated parameters. The evaluation considers multiple network topologies, ranging from shallow to deep architectures, as summarized in Table II. Fully connected SNNs are often chosen for experiments because of their simplicity and demonstrated effectiveness. Their ability to leverage the inherent sparsity and event-driven processing of spiking computation results in significant reductions in power consumption and computational load [5]. This makes them especially suitable for applications in edge scenarios such as health monitoring [21]. Key hyperparameters such as time steps and neuron threshold voltage, shown in Table III, are explored. To ensure comprehensive evaluation, a combination of the topologies in Table II and configurations in Table III is tested, allowing for the identification of the most energy-efficient and reliable SNN models.

The analysis is performed using two widely used classification datasets, MNIST and Fashion-MNIST, abbreviated as "M" and "F" in the tables, along with network topologies and configurations. Two forms of reliability assessment are conducted: model-wise and layer-wise. In the model-wise method, FI is applied to the entire network simultaneously, while in the layer-wise method, faults are selectively introduced into specific layers to evaluate their individual resilience.

TABLE II. Different Network Topologies Used in This Work

Name	Number of Neurons in layers	Number of Layers
TOP0	32-10	2
TOP1	64-32-10	3
TOP2	128-64-10	3
TOP3	128-64-64-32-10	5
TOP4	512-256-256-128-10	5

TABLE III. The Total SNN Configs Used

Config	Timesteps	Threshold Voltage
C1	10	0.5
C2	10	1.5
C3	30	0.5
C4	30	1.5

As shown in Fig. 2, the first experiment compares a trained and quantized ANN with four SNN variants that share the same topology but differ in configuration. Initially, SNN models are trained with predefined hyperparameters, followed by quantization and comparison with their ANN counterparts. The results indicate that SNN models achieved accuracy levels comparable to their ANN counterparts.

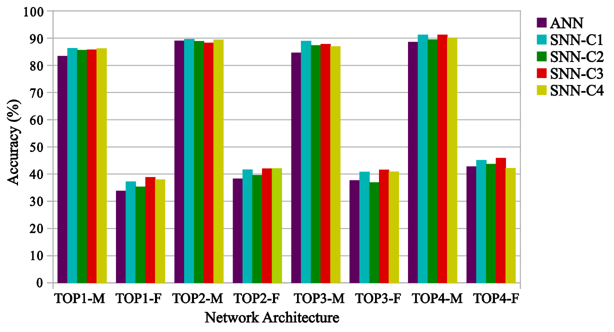
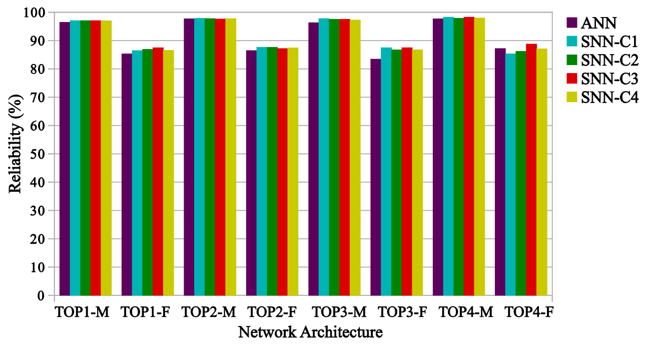


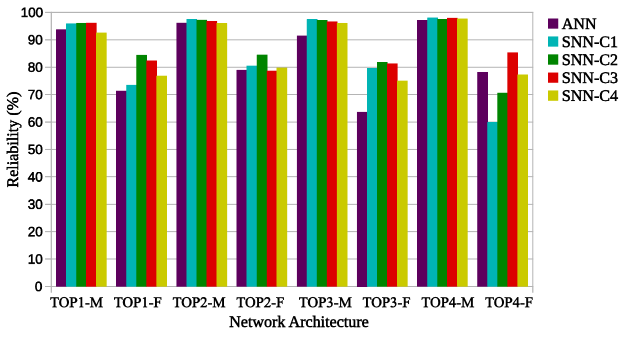
Figure 2. Comparison of accuracy in different architectures

The next study examines the impact of injecting faults into network parameters. For this purpose, four different topologies with four distinct configurations are evaluated, with each graph representing the results for a single BER. As shown in Fig. 3, the experiment covers four BER ranges. The analysis follows a model-wise approach, meaning faults are injected into all hyperparameters of a given model. In each experiment, an ANN is compared with four SNNs of the same topology but different configurations. By analyzing Fig. 3a to 3d, it is evident that networks with different hyperparameters exhibit varying levels of reliability. This underscores the importance of identifying the optimal configuration for an SNN with a given structure. Fig. 3d shows the results of heavy FI as BER equals 0.1, the network has started to lose its parameters, and fault resiliency is unreasonable in this situation.

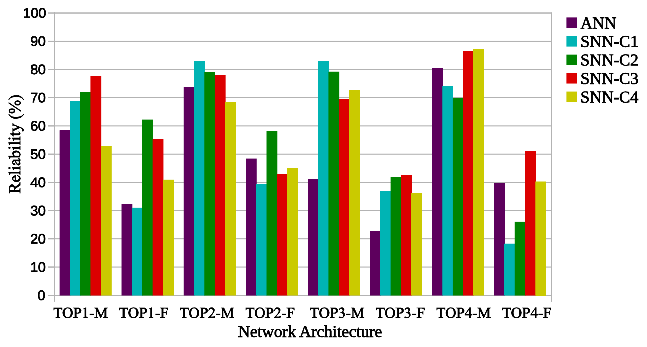
The layer-wise reliability analysis is presented in Fig. 4. Using the proposed framework, a test was conducted across all previously examined cases (various topologies and configurations). After determining the most reliable configuration for each topology, only the best-performing configuration was included in this layer-wise study. This analysis focuses on two topologies: a 3-layer and a 5-layer network. Faults were applied to all layers, and the ANN results



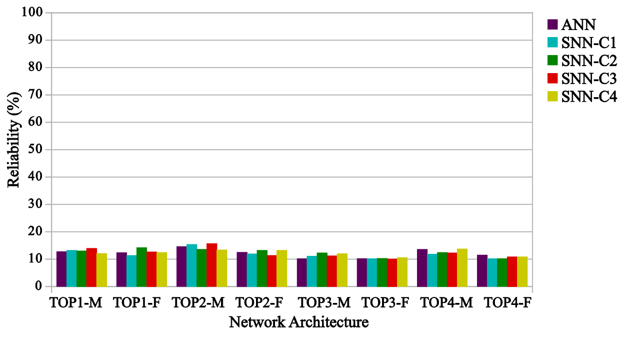
(a) BER = 0.0001



(b) BER = 0.001

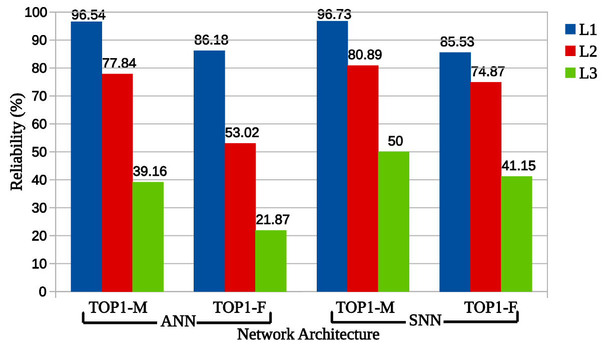


(c) BER = 0.01

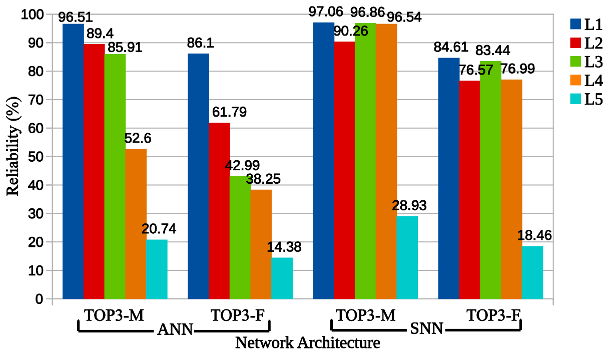


(d) BER = 0.1

Figure 3. Model-wise reliability analysis for some custom network topologies



(a) 3-Layer



(b) 5-Layer

Figure 4. Layer-wise reliability analysis for two 3-Layer and 5-Layer network topologies at BER=0.01

were compared with their corresponding SNNs. In this experiment, only the C2 configuration was analyzed at a BER of 0.01. The results show that SNN layers exhibit greater robustness to faults than their ANN counterparts. For instance, in Fig. 4b, the fourth layer (L4) of the TOP3 SNN achieves 96.54% reliability—1.84× higher than the equivalent ANN topology, which has a reliability of 52.6%.

Accuracy, reliability, and energy consumption trade-offs illustrated in Fig. 5. According to the values in Table I, the figure shows the energy consumption in the two equivalent ANN and SNN topologies. To better highlight differences in energy consumption, two topologies—2-layer and 5-layer networks—are examined, as detailed in Table II. The selected configurations—C2 and C3 for TOP0 and TOP4 respectively—represent the optimal SNN models identified by the proposed framework. As observed, for both topologies and datasets, the accuracy of SNN models remains comparable to their ANN counterparts, while their reliability surpasses that of equivalent ANN models. The figure shows the energy

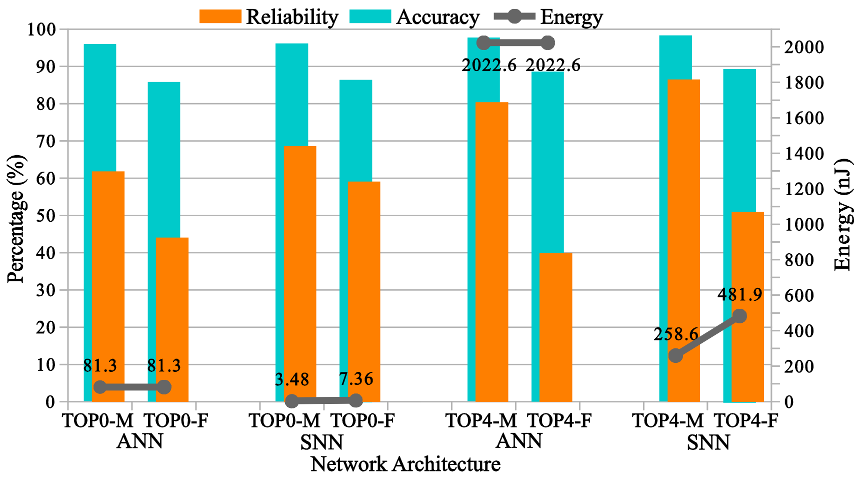


Figure 5. The comparison of accuracy, reliability, and energy for ANN and SNN topologies at BER=0.01

consumption difference between ANN and its equivalent SNN network, especially in a relatively large network. The energy consumption ratio of ANN to SNN in a 2-layer feedforward network (TOP0) is 23.36× for MNIST and 11.05× for Fashion-MNIST dataset. Also in a 5-layer feedforward network (TOP4) is 7.82× for MNIST and 4.2× for Fashion-MNIST dataset. The difference in energy consumption in two SNNs with different datasets is related to the difference in the spike rate of the encoded data of the two datasets, which naturally changes the computational operations and energy consumption.

Based on the data in Fig. 3c and TOP4, if a naïve conversion from ANN to SNN is performed and the proposed framework with three-lateral optimization is not used, the conversion result may end up in one of the configurations such as C1 or C2, which, as is clear from the results, although these configurations meet the accuracy and energy conditions, they deteriorate the reliability in the converted network up to 54.13%. In contrast, the network introduced by the proposed framework, although it meets the accuracy and energy conditions, has also improved its reliability in C3 configuration up to 28.03%.

In some other cases such as TOP3, the proposed framework gives a set of optimizes SNNs, based on Fig. 3c where SNNs showed up to 1.92× and 1.84× better reliability compared to ANNs and lower energy consumption reached up to 27.81× and 15.17× for the MNIST and Fashion-MNIST dataset when using the C2 configuration. Selecting candidate networks without considering reliability may yield better energy efficiency but often lacks fault resilience. Our framework addresses this by balancing all aspects to achieve an optimal trade-off, as reflected in the reported results. Expanding the SNN configuration space could further improve outcomes by offering more design choices.

IV. Conclusion

This paper presented a novel framework for optimizing the conversion of ANNs to SNNs while balancing accuracy, reliability, and energy efficiency. The proposed method systematically explores SNN hyperparameters to identify optimal configurations that maintain accuracy while significantly improving fault tolerance and reducing energy consumption.

Experimental evaluations on MNIST and Fashion-MNIST datasets demonstrated that the optimized SNN models achieved accuracy levels comparable to their ANN counterparts. Moreover, the proposed framework enhanced the reliability of SNNs, as reflected in FI studies, where SNNs showed up to 1.92× and 1.84× lower accuracy degradation under injected faults compared to ANNs in some cases. Additionally, layer-wise reliability assessments confirmed that certain SNN configurations exhibited significantly higher robustness in individual layers than their ANN equivalents.

In terms of energy efficiency, the results showed that SNNs outperformed ANNs by substantial margins. The energy consumption ratio between ANN and SNN reached 27.81× for the MNIST dataset and 15.17× for the Fashion-MNIST dataset in some cases. These findings validate the effectiveness of the proposed approach in achieving energy-efficient and fault-tolerant SNN architectures, making them ideal candidates for edge computing and safety-critical applications.

Acknowledgment

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS“, EU Grant Project 101160182 “TAICHIP“ and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID "458578717".

References

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[3] J. Ding, Z. Pan, Y. Liu, Z. Yu, and T. Huang, "Robust stable spiking neural networks," *arXiv preprint arXiv*:2405.20694, 2024.

[4] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv*:1705.06963, 2017.

[5] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 409662, 2018.

[6] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p.508, 2016.

[7] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, pp. 54–66, 2015.

[8] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding and synaptic weights trained with backpropagation," *Neural Networks*, vol. 143, pp. 100–111, 2021.

[9] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *2022 Design, Automation &Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 370–375.

[10] R. V. W. Putra, M. A. Hanif, and M. Shafique, "Respawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories," in *2021 IEEE/ACM International Conference OnComputer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[11] R. V. Putra, M. A. Hanif, and M. Shafique, "Rescuesnn: enabling reliable executions on spiking neural network accelerators under permanent faults," *Frontiers in Neuroscience*, vol. 17, p. 1159440, 2023.

[12] A. B. Göğebakan, E. Magliano, A. Carpegna, A. Ruospo, A. Savino, and S. Di Carlo, "Spikingjet: Enhancing fault injection for fully and convolutional spiking neural networks," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2024, pp. 1–7.

[13] T. Spyrou, S. Hamdioui, and H.-G. Stratigopoulos, "Spikefi: A fault injection framework for spiking neural networks," *arXiv preprint arXiv*:2412.06795, 2024.

[14] B. Na, J. Mok, S. Park, D. Lee, H. Choe, and S. Yoon, "Autosnn:Towards energy-efficient spiking neural networks," in *International Conference on Machine Learning*. PMLR, 2022, pp. 16 253–16 269.

[15] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[16] S. Venkatesh, R. Marinescu, and J. K. Eshraghian, "Squat: Stateful quantization-aware training in recurrent spiking neural networks," in *2024 Neuro Inspired Computational Elements Conference (NICE)*. IEEE, 2024, pp. 1–10.

[17] S. Barchid, J. Mennesson, J. Eshraghian, C. Djéraba, and M. Bennamoun, "Spiking neural networks for frame-based and event-based single object localization," *Neurocomputing*, vol. 559, p. 126805, 2023.

[18] T. Zhang, S. Xiang, W. Liu, Y. Han, X. Guo, and Y. Hao, "Hybrid spiking fully convolutional neural network for semantic segmentation," *Electronics*, vol. 12, no. 17, p. 3565, 2023.

[19] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[20] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[21] L. Pang, J. Liu, J. Harkin, G. Martin, M. McElholm, A. Javed, and L. McDaid, "Case study—spiking neural network hardware system for structural health monitoring," *Sensors*, vol. 20, no. 18, p. 5126, 2020.

Simply-V: A RISC-V Reconfigurable Playground Soft-SoC for Open Hardware Research and Fast Prototyping

Vincenzo Maisto, Stefano Mercogliano, Manuel Maddaluno, Alessandro Cilardo
Department of Information Technology and Electrical Engineering
University of Naples Federico II, Naples, Italy
{vincenzo.maisto2, stefano.mercogliano, manuel.maddaluno, acilardo}@unina.it

Abstract—**The recent rise of open hardware, mainly driven by the momentum of the RISC-V ecosystem, has sparked significant innovation in the development of open-source CPUs and SoCs. This movement has enabled broad exploration across academia and industry, fostering collaboration and reuse. However, the diversity and openness that empower this space also introduce challenges: academic projects often fall short of industry-grade robustness, lack of standardization, and simulation limitations. To ease the work of researchers some key challenges must be faced in open hardware development: platforms' reconfigurability, ease of integration of third-party IPs, and support for technological heterogeneity. To address these issues, we present Simply-V, a flexible, FPGA-based soft-SoC platform designed for rapid prototyping and open hardware research. Simply-V enables plug-and-play support for multiple CPUs, IPs and accelerators, offers structured configurability across embedded and highperformance profiles, and supports the integration of both RTL and HLS-based components. We demonstrate the SoC's capabilities through platform-fair CPU benchmarking and the iterative development of HLS-designed convolutional accelerators, showcasing simplified fast prototyping, configurability, and heterogeneous IP support on real hardware. Simply-V is openly available at https://github.com/HiSA-Team/Simply-V.git.**

Index Terms—RISC-V, FPGA, Fast-Prototyping, Experimental Research.

I. Introduction

In recent years, open hardware has experienced a remarkable surge, largely fueled by the RISC-V open ISA, which has become a catalyst for research into open-source CPUs and Systems-on-Chip (SoCs) across both academia and industry. On one hand, this rich, diverse, and open environment fosters knowledge sharing and promotes the reusability of hardware solutions. On the other hand, academic projects often fall short of industry-grade standards in areas such as documentation, usability, and long-term maintainability. As a result, open hardware researchers frequently encounter significant challenges, not only in reproducing experimental results, but also in building upon existing work. Most setups are often hard to reproduce, and the inherent heterogeneity can result in inconsistent or non-comparable performance figures. As a mitigation to these challenges, one would wish for a simple, verified and hardware-ready playground platform for open hardware research that is reconfigurable, easy to use and reuse in larger systems. Such an achievement, however, is nontrivial for several reasons. First, validating CPUs in realistic scenarios, such as running full operating systems or benchmarking memory hierarchies, goes beyond basic testbenches. Second, while reusable IP blocks like accelerators, peripherals and protocol bridges are widely available, they often lack consistent interface standardization and toolchain compatibility, shifting focus from open hardware research to low-level troubleshooting. Lastly, seamless configurability remains a major roadblock. Tasks like address mapping, dependencies management, memory and bus resizing, or clock domain crossing (CDC) are often hardcoded or require manual rework, limiting scalability and slowing down design iterations.

To address these challenges, we present Simply-V (pronounced "simplify-ve"), an easy-to-deploy, flexible, and extensible soft-SoC platform for rapid prototyping, open hardware research and development. Simply-V provides a simple, FPGA-based, hardware-ready playground platform for full-stack evaluation on real physical devices in open hardware research. The platform offers a structured reconfiguration flow, is portable across a wide range of FPGA devices, supporting both embedded and high-performance profiles. It enables drop-in integration of multiple CPUs and accelerators, along with a configurable interconnect managed through a high-level flow. We demonstrate the capabilities of Simply-V with a platform-fair benchmarking of set open-source CPUs, across the embedded and high-performance computing (HPC) profiles, in both 32 and 64 bits. Additionally, we validate the integration of custom IPs, support for technological heterogeneity and fast design iterations by deploying a set of incrementally-designed high-level synthesis (HLS) convolutional accelerators.

II. BACKGROUND AND MOTIVTION

A. A Flexible SoC for Open Research and Fast Prototyping

While many reconfigurable RISC-V-based platform designs have been proposed, a significant number of them are either proprietary or not publicly accessible. Conversely, most opensource RISC-V CPUs are distributed with minimal testing environments, valuable for evaluating the processor's functionalities but not meant to serve as hardware-ready SoCs.

Manuscript received April 30, 2025; revised July 23, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025.
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.86

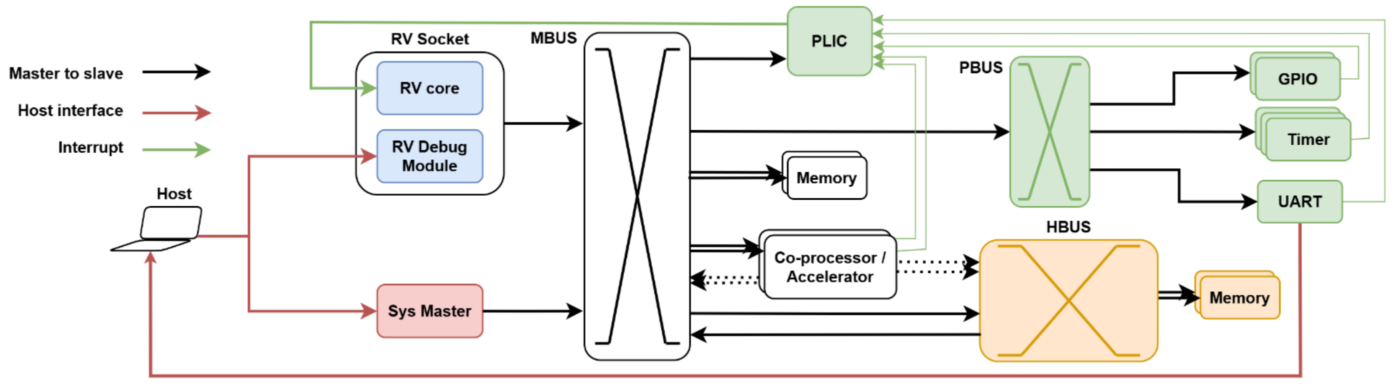


Fig. 1. General architecture and on-chip interconnect of Simply-V. It features a main bus (MBUS), a peripheral bus (PBUS) for low-speed devices and a high-performance bus (HBUS) for high-bandwidth memory accesses, suitable for accelerators and co-processors. On the MBUS, the RV Socket hosts a RISC-V processor and debug module. Finally, the SysMaster grants the host full control of the platform and master access to the SoC interconnect.

More sophisticated open-source SoCs do exist, such as lowRISC's OpenTitan [1], however, these systems lack the right reconfigurability across a wide spectrum of technology needed for open-hardware research. In contrast to fixed-functionalities SoCs, frameworks for SoC generation can offer more flexibility, such as ESP [2] and Chipyard [3]. Similarly to Simply-V, using a configuration-driven design flow, they can generate RTL for a complete SoC, including the CPUs, caches, interconnects, and co-processors. On the other hand, Chipyard specifically focuses on verification of ASIC tape-outs, and ESP is mostly oriented at tiled architectures, integrating third party SoCs and IP cores. Instead, Simply-V is explicitly targets FPGA platforms and SoC architecture, with fast prototyping for academic research as its primary aim.

B. *A Practical Alternative to Simulation*

Simulation frameworks have long been the cornerstone in digital design validation. System emulation platforms such as QEMU, and instruction set simulator like Spike, are clearly limited to functional validation, and cannot be used for performance evaluation. More advanced tools, such as gem5 [4] and event-driven simulators, like GVSoC [5], offer reasonable trade-offs between timing accuracy and simulation time, but tend to be platform-specific and require a reimplementation of the simulated modules. On the other hand, cycle-accurate RTL simulators are often prohibitively slow, namely when simulating long-running programs like booting an OS. Hybrid hardware/software co-simulation approaches attempt to mitigate these issues, but they rely on custom intermediate representations, or emulation on expensive FPGA platforms [6].

Simply-V addresses these challenges by providing the fidelity of hardware execution without the burden of RTL design and platform integration. Such flexibility allows rapid prototyping, fast design iterations, and real-system validation beyond cycle-accurate simulations or FPGA-based emulations.

III. ARCHITECTURE

This section describes the design principles of our Simply-V and design challenges it addresses. To ease the work of researchers and practitioners, we focus on (1) system-wide reconfigurability, (2) ease of integration of custom IPs and (3) technological heterogeneity as pivotal requirements.

MBUS: the Simply-V architecture, depicted in Figure 1, is based on fully parametric and reconfigurable, yet simple, main bus (MBUS) interconnect, based on AMBA AXI4. Most memories in Simply-V, generically encompassing ROMs, onchip SRAMs and external DRAMs, are mapped on the MBUS.

PBUS: Low-end and low-frequency peripherals, which commonly require a limited range of addresses for register file data and control, are collected in the peripheral bus (PBUS), as an additional slave to the MBUS. By design, the PBUS is meant for non-performance-critical bus traffic, hence we opt for an AXI4-lite interconnect.

HBUS: Our platform is also designed of accelerator development and HPC configurations, hence a high-bandwidth bus (HBUS) interconnect is exposed as a further slave of the MBUS. The HBUS offers streamlined, long-word access to more performant memories, such as DDR banks or HBM channels. The HBUS is suitable for high-performance accelerators, which require high-bandwidth to external DRAM memories, such as AI engines or vector co-processors.

PLIC and interrupts: Interrupts are managed by an implementation of the RISC-V standard platform-level interrupt controller, namely PLIC, integrated as a custom unit leveraging our custom IP flow, detailed in Section III-C.

SysMaster: Since Simply-V is primarily designed for research and development, host-side debug is a fundamental feature. We explicitly expose on the MBUS a host-side connection, the SysMaster. It allows the user to directly interact with the SoC, e.g. to inject faults or read-back data over a high-speed link, e.g. PCIe, rather than low-speed JTAG.

Cross-profile UART: A UART peripheral is hosted in the PBUS for both embedded and HPC profiles. For embedded, we leverage a physical UART IP over a PMOD connector. In case of HPC deployment, such as on PCIe acceleration cards, a PMOD connection is typically not available. Therefore, we design a virtual UART module to emulate the same behaviour of its physical counterpart over the PCIe link.

A. *SoC Configuration Flow*

Simply-V provides a lightweight, parameter-based configuration flow to re-shape the platform at build time and restructure the whole SoC to adapt it to their experimental needs.

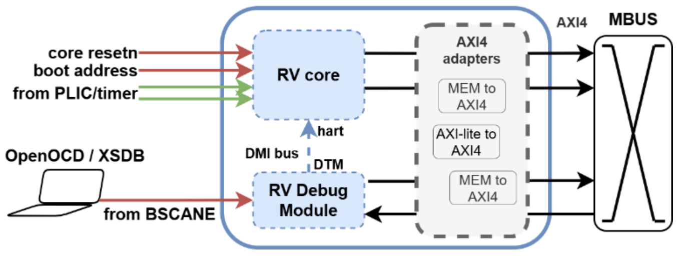


Fig. 3 . Architecture of the RV Socket, with CPU-specific logic enclosed in dashed lines.

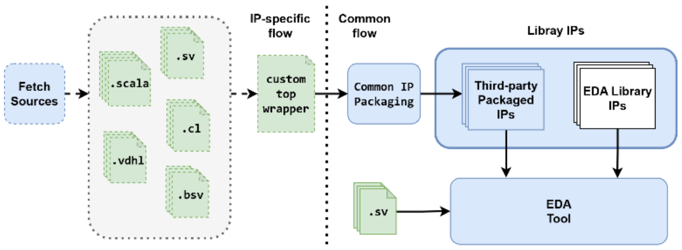


Fig. 2. Custom IP packaging flow. On the left, in dashed lines the IP-specific steps. On the right, the common steps managing packaged IPs as library elements in the target EDA tool.

Four modular input tables are required, namely a systemlevel configuration, e.g. target CPU, RISC-V XLEN, and three configuration files for MBUS, PBUS and HBUS, respectively, for interconnect, address map and CDC management.

Buses are reconfigured in a transparent, automated and verified process. The necessary RTL modifications to remap addresses and interconnections are automated and hidden to the user. Peripheral IPs, such as timers, GPIOs or accelerators, can be optionally instantiated in one or multiple instances and their clock frequency, alongside the CPU clock, can be controlled from configuration files.

B. RV Socket and Debug Support

Simply-V aims at providing a fast deployment flow that targets different processors with a vendor-agnostic plug-in framework. To address such needs, we introduce the RV Socket, a modular CPU wrapper offering a unified interface for the RISC-V cores towards the MBUS for operation and the host for debugging. Figure 2 depicts the architecture of the RV Socket. In the following, we present motivation and detail our design choices.

1) *Unifying CPU Interfaces:* In order to provide a vendor-independent interface for CPUs, we leverage our custom IP flow to provide a packaging framework for compatible and re-usable adapters. Adapters can be either imported or implemented from scratch, and deployed alongside the RISCV CPU to provide a unified interface for all CPUs, allowing plug-in CPU support.

2) *RISC-V External Debugging:* RISC-V CPUs can support a Debug Transport Module (DTM) for external debugging. The RISC-V debug specification defines a Debug Module Interface bus (DMI), but the implementation is left to the designer. Consequently, each RISC-V core comes with a tightly-couples DTM. With our simple configuration flow, the transition between CPUs remains seamless, which transparently enables the right DTM and compatible DMI interconnect.

C. Custom IP Packaging

For Simply-V, we design a custom IP packaging methodology to ease the integration of custom and third-party IPs. We allow users to package RTL or other HDL sources into a selfcontained IP, namely a blunt out-of-context netlist, with no remaining references to its source code. Such a flow is depicted in Figure 3. The first step of the packaging is to provide the necessary sources, resolve internal dependencies, and is IPspecific. Simply-V poses no constraints on this step, other than providing a custom top wrapper module for all IP sources. The second step is unified for all IPs and automatically builds such top module in a library IP element. Consequently, this strategy enables Simply-V to integrate third-party IPs with potential non-compatible code bases, effectively turning third-party IPs in simple and off-the-shelf library elements.

D. Managing Clock Domain Crossing

Managing clock domains can be difficult and deploying a whole SoC in the same domain might be inefficient in performance and power. Our configuration flow allows for the slaves of the main bus to be clocked at different frequencies, with automated and verified CDC bridges deployment. A main clock domain is shared by RV Socket, MBUS, PLIC and BRAM memories, as such modules typically show no advantage in a dedicated domain. The PBUS hosts all of its low-speed peripherals in a single domain, clocked at lower frequency. All MBUS additional peripherals, such as programmable co-processors or specialized accelerators, can be placed in a dedicated domain, allowing fast integration at their natural frequency, or use the MBUS domain. Moreover, the HBUS maximizes integration with DDR and HBM channels by deploying in their high-speed clock domain. Such a domain is available for accelerator deployment for the best performance and integration with the high-speed interconnect.

IV. EXPERIMENTAL VALIDATION

In this section, we empirically validate the capabilities of Simply-V for fast prototyping and research. We configure our SoC CDC with PBUS, HBUS and DRAM memories in dedicated clock domains and build our FPGA designs with Vivado v2024.2. We deploy Simply-V embedded profile on Digilent Nexys A7 Artix-7 board. For validation in HPC profile, we use an AMD Xilinx Alveo PCIe Card.

A. Cross-vendor CPU Benchmarking

We demonstrate a platform-fair comparison of RISC-V CPUs and the plug-and-play support of multiple processors from a diverse pool of vendors, namely CV32E40P from OpenHW, Ibex from lowRISC, MicroblazeV from AMD Xilinx, and finally, we demonstrate RV64 support with CVA6. Leveraging our configuration flow, for given a Simply-V setup we seamlessly plug in and out different CPUs. Additionally, we showcase the transparent profile transition from embedded to

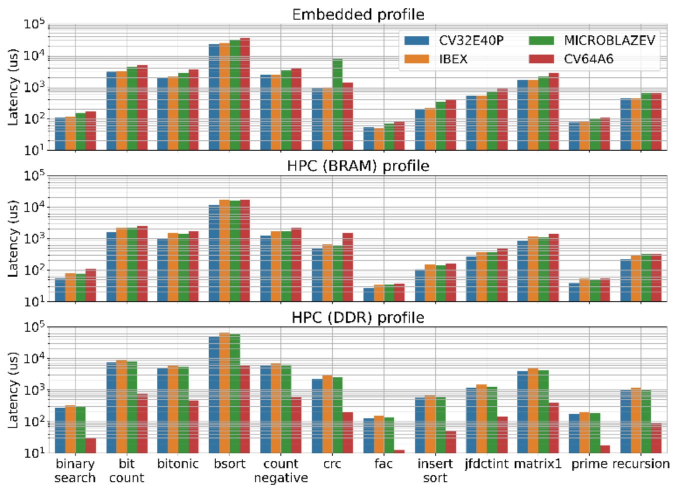


Fig. 4. Latency results of tacle-bench CPU benchmarking across Simply-V profiles and memory devices.

HPC setups, with no overhead from the practitioner. We run the tacle-bench1 benchmark on all CPUs and Simply-V profiles. For the embedded profile, we run software from onchip BRAM memory, in the main clock domain at 50MHz, alongside the CPU. For the HPC profile, we run the software both from BRAM memory and an external DDR bank.

Figure 4 shows the tacle-bench[1] latency results, averaged over 10 iterations. CPUs can be quickly evaluated and compared in the embedded profile, for its fast turn-around time, obtaining a fast, baseline indication with the sophisticated CVA6 core performing worse than simpler cores. Transitioning to a HPC configuration, namely and keeping code in local BRAMs or off-chip DDR, and increasing the target frequency in the main clock domain, a researcher can easily evaluate the differences in performance for the various cores.

B. *Fast Prototyping an HLS-based Convolutional Core*

In this section, we demonstrate the use of Simply-V as a hardware-ready platform for fast prototyping custom IPs, including support for technological heterogeneity with HLS technology. We target an 8-bit 2D convolutional engine, namely CONV2D, as a representative example of modern workloads. We implement a pool of CONV2D engines and integrate each one in Simply-V as an accelerator IP, demonstrating fastprototyping capabilities, both from the IP design and SoC integration perspective. Figure 5 reports the HLS engine's performance across design iterations: (1) *Naive loop-nest*: the baseline prototype of our core is a basic HLS-compatible Ccode, with a single AXI master port for memory access. (2) *AXI bursts*: activating memory coalescing; (3) *Double buffering*: implementing double buffering; (4) *Frequency boost*: leveraging our configuration-based CDC, we boost the IP clock frequency; Such approach improves performance by a only limited amount, suggesting the IP core is bottle-necked by memory accesses; (5) *Split interfaces*: maximizing data-access parallelism with three parallel read and write AXI ports; (6) *HBUS access*: alternatively, leveraging the HBUS interconnect for wider memory accesses, showcasing the best performance.

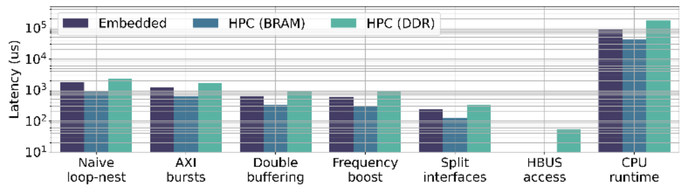


Fig. 5. Demonstrating fast prototyping with the latency evaluation of the multiple design iteration of HLS-CONV2D IP across Simply-V profiles.

V. CONCLUSIONS

In this work, we presented Simply-V, a reconfigurable, hardware-ready soft-SoC platform for fast prototyping and open hardware research. We demonstrated the capabilities of our platform by simplifying platform-fair CPU benchmarking and fast prototyping a HLS-based convolutional engine. Moving forward, we plan support for additional CPUs and IPs for RISC-V extensions and heterogeneous technologies, such as Chisel. We plan to soon boot Linux on Simply-V and deliver a full-fledged platform for experimental and applied research.

ACKNOWLEDGMENTS

This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of ICSC - Centro Nazionale di Ricerca in High-Performance-Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU.

REFERENCES

[1] M. Ciani et al., "Unleashing opentitan's potential: a silicon-ready embedded secure element for root of trust and cryptographic offloading," ACM Transactions on Embedded Computing Systems, 2024.
[2] P. Mantovani et al., "Agile SoC development with open ESP," in Proceedings of the 39th International Conference on Computer-Aided Design, ICCAD '20, (New York, NY, USA), Association for Computing Machinery, 2020.
[3] A. Amid et al., "Chipyard: Integrated design, simulation, and implementation framework for custom socs," Ieee Micro, vol. 40, no. 4, pp. 10–21, 2020.
[4] N. Binkert et al., "The gem5 simulator," ACM SIGARCH computer architecture news, vol. 39, no. 2, pp. 1–7, 2011.
[5] N. Bruschi et al., "GVSoC: A Highly Configurable, Fast and Accurate Full-Platform Simulator for RISC-V based IoT Processors," in 2021 IEEE 39th International Conference on Computer Design (ICCD), pp. 409–416, 2021.
[6] S. Karandikar et al., "FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), pp. 29–42, IEEE, 2018.

[1] https://github.com/tacle/tacle-bench/tree/V1.9

Towards Trustworthy Adaptation of Cyber-Physical Production Systems with Contract-Based Design

Hossein Rahmani[†] Kristof Meixner[†] Stefan Biffl[†]
[†]Institute of Information Systems Engineering, TU Wien
E-Mail: [first].[last]@tuwien.ac.at

***Abstract*—Adapting a Cyber-Physical Production System (CPPS) to different production goals and conditions requires capabilities to validate multi-domain dependencies. Traditional approaches to CPPS adaptation rely on domain experts' implicit knowledge, making reconfiguration prone to error, challenging to validate, and hard to trust. Our research aims at improving the trustworthiness of the CPPS adaptation process regarding effectiveness, risk mitigation, and understandability, with a formal representation of reconfiguration dependencies and conditions. This paper introduces the approach *Trustworthy Adaptation Process for CPPS (TAP-CPPS)* to validate the feasibility of achieving the adaptation goal by reconfiguration. TAP-CPPS is a systematic approach to (i) model the adaptation process using BPMN; and (ii) validate the adaptation process model using *contracts* by verifying explicit reconfiguration pre-/post-conditions in the BPMN model, which is linked to the CPPS configuration variants. We initially evaluate TAP-CPPS with a use case of a CPPS for joining car parts, and derive a research agenda.**

***Index Terms*—Production Systems Engineering, Industry 4.0, Multi-disciplinary reconfiguration, Adaptive production system.**

I. Introduction

Industry 4.0 (I4.0) has envisioned to realize adaptive Cyber-Physical Production Systems (CPPSs) [1] in order to meet the growing demands for flexible and responsive production [2]. In this work, CPPS adaptation refers to the process of adjusting the CPPS's behavior in response to a change in a dynamic production environment [3]. An adaptation process often involves reconfiguring several components, including Product-Process-Resource (PPR) aspects, to achieve the *target adaptation goal*, such as addressing changes in market demands and customer requirements [4], technology [5], or regulations [3], [6].

Traditional approaches for CPPS adaptation and reconfiguration [7] rely on domain experts' implicit knowledge. This tacit knowledge is often fragmented among experts coming from several domains, e.g., mechanical, electrical, system, and software engineering, each of whom has partial or incomplete knowledge required for designing and evaluating adaptation. Hence, traditional CPPS adaptation approaches are prone to error, challenging to validate, and hard to trust [5], [8].

Considering the complex multidisciplinary nature of a CPPS and its adaptation process, structured methods and appropriate models are required to enhance the *trustworthiness of CPPS adaptation* and the underlying reconfigurations [1], [5], [8]. Trustworthiness is an umbrella term for properties including safety, security, reliability, integrity, availability, and understandability [9]–[11]. This work focuses on the integrity and understandability aspects of trustworthiness.

To address the challenges and enhance the *trustworthiness of CPPS adaptation*, this paper proposes a systematic multi-view approach to ensure (i) the target configuration variants are valid; (ii) the target adaptation goal is achievable by the planned reconfigurations; and (iii) the adaptation and reconfiguration processes are understandable and verifiable by humans (various stakeholders) and machines. This paper shall address the Research Question: *What approach can validate whether a target adaptation goal is achievable by planned reconfiguration activities?*

With this aim, we introduce the approach *Trustworthy Adaptation Process for CPPS (TAP-CPPS)* built on the approach *PPR Asset Network with Reconfiguration (PAN+R)* [5]. TAP-CPPS is a systematic approach to (i) model the CPPS adaptation process using the Business Process Model and Notation (BPMN); and (ii) validate the adaptation process model and underlying reconfigurations using *contracts* [12] by explicitly verifying the reconfiguration pre- and post-conditions in the BPMN, which is linked to the CPPS configuration variants.

We illustrate an application of TAP-CPPS to evalutate the reconfiguration of an automated industrial screwdriver, a typical flexible resource in car production.

The remainder of this paper is structured as follows. Section II summarizes the related work. Section III introduces the use case. Section IV introduces our proposed approach, TAP-CPPS, illustrated with data from the use case. Finally, Section V concludes the paper with a research agenda.

II. Related Work

Multi-view Configuration Management (CM) in CPPS engineering, according to the VDI 3695 [7], aims at managing the correct migration between CPPS configurations. A CPPS configuration represents a consistent, validated combination of all required system elements. While the VDI 3695 addresses multidisciplinary CM, it does not address trustworthy CM.

The guideline VDI 3682 [13] provides a formalism for describing the production processes based on the core PPR concepts. Building on PPR, the *PPR Asset Network (PAN)* [14] is an I4.0 asset-based coordination artifact, which can represent PPR dependencies for a specific configuration variant. However, it does not support multiple variants required for reconfiguration. Extending the PAN, the *PAN+R* approach [5] provides (i) knowledge representation required to coordinate CPPS reconfiguration, and (ii) an approach for validating the reconfiguration process based on multidisciplinary pre-

Manuscript received July 15, 2025; revised July 31, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.87

and post-conditions. However, the PAN+R does not consider formal notations for modeling complex adaptation processes.

Trustworthy adaptation of CPPS requires a suitable architecture that supports the modeling and management of multi-domain reconfiguration knowledge and coordination of the adaptation process [15]. MAPE-K [16] defines a reference framework for self-adaptive systems by organizing the adaptation process into four core functions: Monitor, Analyze, Plan, and Execute, along with a central Knowledge component. MAPE-K encourages the principle of separation of concerns, which provides a suitable basis for our proposed approach.

The BPMN standard [17] allows modeling complex business processes that technical and non-technical experts, and machines can interpret. The BPMN can be used to extend the PAN+R approach for modeling complex adaptation processes. Yet, BPMN *per se* does not consider PPR assets or the conditions and data required for the adaptation validation. This work explores linking the adaptation process in BPMN to PPR assets and conditions [5], [18] using *contracts* [12].

A contract for a component is a pair of an assumption and a guarantee. The component guarantees a particular behavior if the environment satisfies the assumption [12]. Contract-based design is a rigorous method for verification, analysis, and abstraction/refinement [12], However, to our knowledge, this method has not been applied to adaptive CPPSs.

The modeling method *procan.do* [19], [20] facilitates understanding, for a process or system of interest, the assets, stakeholders, and data required to analyze multi-domain contributions to a desired or undesired outcome. In this paper, we use procan.do to derive the stakeholders, contract conditions, and data sources required to evaluate the contract conditions.

This paper shall go beyond the state of the art in CPPS adaptation and reconfiguration [5], [7]. We introduce a systematic multi-view approach to validate whether the target adaptation goal is achievable by the planned reconfiguration activities.

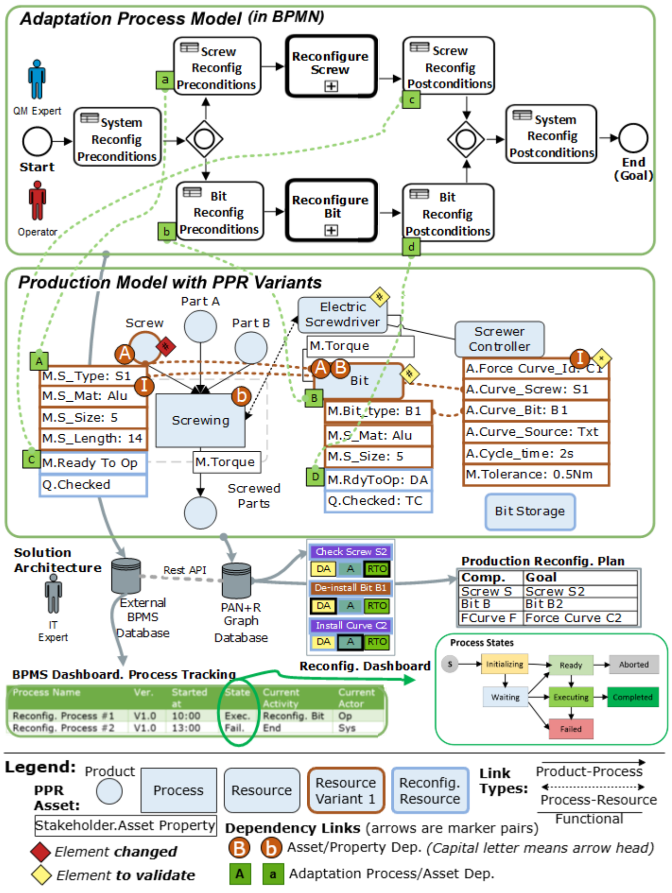
III. Use Case Work Cell Adaptation

Based on a domain analysis of screwing work cells [21], [22], we abstracted the illustrative use case *adaptation of a screwing work cell*. Moreover, we identified the requirements for knowledge representation on reconfiguration. Specifically, we describe components of a robotic work cell equipped with an electric screwdriver. The screwdriver consists of a bit and a screwer controller that uses a force curve to define the screwing process behavior.

In the use case, a quality expert collaborates with process experts and detail planners to define and validate reconfiguration procedures for the operator who conducts the reconfiguration. A representative multidisciplinary reconfiguration task is changing the screw type. This change requires checking and modifying the screwing bit and the force curve of the screwing process. It involves dependencies between all PPR aspects, including mechanical and automation engineering disciplines. For validating a reconfiguration process with PPR change dependencies, we identified three essential modeling requirements: (R1) representation of the PPR change knowledge, (R2) representation of the reconfiguration process, and (R3) linking the reconfiguration process with the PPR model, making dependencies explicit for validation and traceability.

IV. Trustworthy Adaptation Process for CPPS

This section introduces the approach *Trustworthy Adaptation Process for CPPS (TAP-CPPS)* and demonstrates its application using data from the use case *adaptation of a screwing work cell*. We apply *procan.do* [19], [20] to analyze dependencies in the adaptation process. We identify the assets, stakeholders, conditions, and data sources required to evaluate, verify, and validate *contracts* [12] in the adaptation process.



Comp.	Goal
Screw S	Screw S2
Bit B	Bit B2
FCurve F	Force Curve C2

Process Name	Ver.	Started at	State	Current Activity	Current Actor
Reconfig. Process #1	V1.0	10:00	Exec.	Reconfig. Bit	Op
Reconfig. Process #2	V1.0	13:00	Fail.	End	Sys

Fig. 1. *Trustworthy Adaptation Process for CPPS (TAP-CPPS) approach*: Solution Overview.

Fig. 1 illustrates the TAP-CPPS approach consisting of (i) a *Production Model with PPR variants* and the required production change knowledge (cf. Fig. 1, middle), (ii) an *Adaptation Process Model* including the required reconfiguration knowledge (cf. Fig. 1, top), and (iii) *Links between the adaptation process elements and the production assets* (cf. Fig. 1, green dashed lines) for validating the reconfiguration pre- and post-conditions (cf. Tab. I) using contracts. These linked models form a knowledge graph that can be queried to (i) derive and validate an adaptation plan with the underlying reconfigurations, and (ii) inform the operators via a dashboard about the reconfiguration tasks and their current status. The main parts of TAP-CPPS (cf. Fig. 1) are explained as follows.

(i) Production Model with variants. The TAP-CPPS production model builds on the PAN [14] to represent PPR assets and properties (circles and boxes in light blue color), such as the screwing process and functional dependencies between PPR assets (black arrows). The production model also contains variants of the PPR assets and properties (PPR elements with frames in brown color) to represent the production variants, such as screw types, bits, screwing processes, and screwing force curves. The variants of a PPR element are connected by transition dependencies (brown arrows with dashed lines). The change dependencies between (variants of) PPR elements are also represented in the production model by brown dashed lines, e.g., between the screw and the bit.

TAP-CPPS production model properties can represent the reconfiguration states of components, such as *assembly* or *validation* states. For validating a sequence of reconfiguration tasks, the valid states and transitions can be defined using state machines, considering multidisciplinary dependencies. To represent reconfiguration assets or properties required only for coordinating the adaptation, not for production, the production model contains PPR elements in a light blue frame, e.g., *Bit Storage*. The production model uses red and yellow diamonds for marking a changed PPR asset and the related PPR elements to validate. The validation of each element can be addressed by a *contract*, defined as a set of *assumption* pre-conditions and a set of *guarantee* post-conditions, and its evaluation process.

(ii) Adaptation Process Model. An adaptation process consists of reconfiguration tasks with pre- and post-conditions, each leading from a start to a goal state (cf. Fig. 1, top). For instance, the adaptation of the screwing system requires reconfiguration tasks for the screw, bit, and screwing curve. The process expert defines the reconfiguration task conditions considering dependencies and states in the TAP-CPPS production model. A Business Process Management System (BPMS) can track and monitor the execution of the adaptation process for normal or special cases.

(iii) Knowledge graph of the adaptation process linked to the production assets. The domain concepts in the task pre- and post-conditions linked to PPR elements (cf. Fig. 1, green dashed lines) build the foundation to validate these concepts with their dependencies in the TAP-CPPS knowledge graph using *contracts* [12] and an information system for validation.

Evaluation. As an initial feasibility evaluation, we conducted TAP-CPPS for the use case *adaptation of a screwing work cell* (cf. Section III) following the *procan.do* method.

Step 1: The scope of work is the adaptation of the screwing work cell (cf. Fig. 1) with desired and undesired outcomes.

Step 2: Process analysis results in a BPMN process model for adapting the screwing work cell with the required reconfiguration tasks (cf. Fig. 1, top). This adaptation model should lead to desired outcomes, such as completing the customer orders on time, with limited resources. This step identifies high-risk undesired process outcomes, such as completing the customer orders with delays or high unplanned costs.

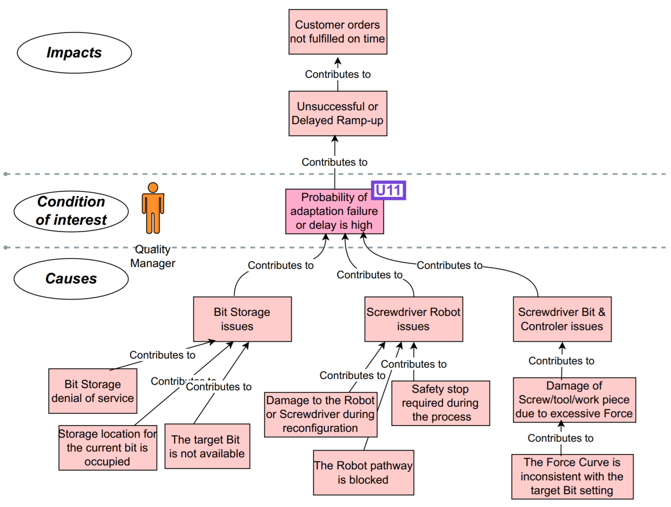

Impacts
Condition of interest
Causes
Quality Manager
Customer orders not fulfilled on time
Contributes to
Unsuccessful or Delayed Ramp-up
Contributes to
U11
Probability of adaptation failure or delay is high
Contributes to
Contributes to
Contributes to
Bit Storage issues
Screwdriver Robot issues
Screwdriver Bit & Controler issues
Contributes to
Bit Storage denial of service
Contributes to
Contributes to
Storage location for the current bit is occupied
The target Bit is not available
Contributes to
Contributes to
Damage to the Robot or Screwdriver during reconfiguration
Safety stop required during the process
The Robot pathway is blocked
Contributes to
Damage of Screw/tool/work piece due to excessive Force
Contributes to
The Force Curve is inconsistent with the target Bit setting

Fig. 2. Undesired conditions in adaptation process of *Screwing Work Cell*.

TABLE I
PRE- AND POST-CONDITIONS OF RECONFIGURATION TASKS (CF. FIG. 1).

Condition Id	Condition Description
Screw.Reconfig. Precondition	'Screw S1'.M.'Ready To Op' == assembled ∨ ready ∧ 'Screw S2'.M.'Ready To Op' == disassembled.
Screw.Reconfig. Postcondition	'Screw S2'.M.'Ready To Op' = assembled ∨ ready ∧ 'Screw S1'.M.'Ready To Op' == disassembled.
Bit.De-install Precondition	'Bit B1'.M.'Ready To Op' == assembled ∨ ready ∧ 'Bit B2'.M.'Ready To Op' == disassembled.

Step 3: Condition analysis starts with analyzing the desired conditions of the "sunshine case" in the BPMN model, coming from Step 2. Then, analysis shall focus on an undesired condition, e.g., *"probability of adaptation failure or delay is high"* (cf. Fig. 2). Domain experts shall identify conditions, including the pre- and post-conditions (cf. Tab. I) to be verified by *contracts* at interfaces between stakeholder modules.

Step 4: Analyze assets, stakeholders, and dependencies related to the conditions and identify the asset properties and data sources required to evaluate these conditions [20].

Step 5: Validate contract conditions evaluates for the steps in a reconfiguration process the fulfillment of pre- and post-conditions (cf. Tab. I) to list the issues, in particular, false post-conditions (guarantee) with true pre-conditions (assumption).

In the evaluation, we identified lines of undesired conditions likely to contribute to an undesired outcome (cf. Fig. 2). The combination of conditions can specify *special cases* that require countermeasures, which may be expressed as a contract for the expected behavior. In the use case context, the TAP-CPPS approach facilitated modeling the adaptation process in BPMN and validating contract conditions on the adaptation process for typical reconfiguration activities, which may result in *special cases*. Therefore, TAP-CPPS was found sufficient to validate the feasibility of the adaptation goal by stepping through typical reconfiguration chains of tasks.

V. Conclusion

The Industry 4.0 vision of adaptable robot work cells [23] requires capabilities for (i) multidisciplinary reconfiguration based on a model with PPR dependencies; (ii) the flexible design of reconfiguration processes according to a production model to accommodate for new products, processes, and production system components; and (iii) coordinating human and machine agents. However, traditional reconfiguration processes are (i) often workflows designed for a specific production system and (ii) unaware of production dependencies.

In this paper, we introduced the TAP-CPPS approach that goes beyond the state of the art [5], [7] by representing PPR asset dependencies in a production model. This representation facilitates validating a flexible reconfiguration process as a foundation for coordinating production reconfiguration. Together, the TAP-CPPS production model and the reconfiguration process model can represent the data required for change planning and monitoring. Further, the TAP-CPPS knowledge graph facilitates queries to PPR elements, their variants, and dependencies [14]. Thus, TAP-CPPS provides the basis for effective change coordination of human and machine agents. An initial evaluation of the TAP-CPPS knowledge graph using the screwing work cell use case showed promising results. This suggests exploring its application in a broader range of production adaptation settings that face trustworthiness challenges to better understand its strengths and limitations.

Overall, TAP-CPPS seems well suited to enhance the trustworthiness of the CPPS adaptation process by supporting the specification and validation of reconfiguration effectiveness and mitigating associated risks. It also offers a formal representation of PPR reconfiguration dependencies and conditions, making it suitable for auditing industrial production processes.

Research agenda. *Towards trustworthy self-adaptive production.* We plan to apply the TAP-CPPS approach for production system reconfiguration by coordinating (i) the PPR reconfiguration process design and validation regarding contracts on dependencies in and across disciplines; and (ii) one or more operators with tool support towards valid reconfiguration with run-time input data. We consider investigating (i) operator assistance with a reconfiguration dashboard (cf. Section IV); and (ii) automating selected reconfiguration tasks towards a self-adaptive CPPS for a suitable scope of reconfiguration.

Empirical studies. We plan to identify applicable metrics and explore the trustworthiness, usability and usefulness, and scalability of the TAP-CPPS approach with domain experts in empirical studies in various production adaptation contexts. Also, we plan to conduct quantitative evaluations to report the results of quantitative performance analysis for TAP-CPPS.

Scalability. We plan to explore how to derive a reconfiguration process from a TAP-CPPS production model for large use cases, such as a robot for flexible use in various work cells and lines that may require dozens of production dependencies and a dozen change variants to the robot configuration.

References

[1] T. Müller, B. Caesar, M. Weiß, S. Ferhat, N. Sahlab, A. Fay, R. Oger, N. Jazdi, and M. Weyrich, "Reconfiguration management in manufacturing: A systematic literature review," *at - Automatisierungstechnik*, vol. 71, no. 5, pp. 330–350, May 2023, literatur Review.

[2] R. Dumitrescu, T. Westermann, and T. Falkowski, "Autonome systeme in der produktion," *Industrie 4.0 Management*, pp. 17–20, 2018.

[3] D. Weyns, I. Gerostathopoulos, N. Abbas, J. Andersson, S. Biffl, P. Brada, T. Bures, A. Di Salle, M. Galster, P. Lago, G. Lewis, M. Litoiu, A. Musil, J. Musil, P. Patros, and P. Pelliccione, "Self-adaptation in industry: A survey," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 18, no. 2, pp. 1–44, May 2023.

[4] K. Meixner, F. Rinker, L. Waltersdorfer, A. Lüder, and S. Biffl, "Organizing reuse for production systems engineering with capabilities and skills: Organisation der wiederverwendung im engineering von produktionsystemen mit capabilities und skills," *at - Automatisierungstechnik*, vol. 71, no. 2, pp. 116–127, Feb. 2023.

[5] S. Biffl, K. Meixner, D. Hoffmann, J. Musil, H. Rahmani, and A. Luder, "Towards coordinating production reconfiguration," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, Sep. 2022, pp. 1–4.

[6] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, and M. Sharaf, *Patterns for Self-Adaptation in Cyber-Physical Systems*. Springer International Publishing, 2017, pp. 331–368.

[7] *VDI Guideline 3695: Engineering of industrial plants - Eval. and opt.* Beuth, 2009.

[8] H. Haddou Benderbal, A. R. Yelles-Chaouche, and A. Dolgui, *A Digital Twin Modular Framework for Reconfigurable Manufacturing Systems*. Springer International Publishing, 2020, pp. 493–500.

[9] J. M. G. Sánchez, N. Jörgensen, M. Törngren, R. Inam, A. Berezovskyi, L. Feng, E. Fersman, M. R. Ramli, and K. Tan, "Edge computing for cyber-physical systems: A systematic mapping study emphasizing trustworthiness," *ACM Transactions on Cyber-Physical Systems*, vol. 6, no. 3, pp. 1–28, Jul. 2022.

[10] R. F. Babiceanu and R. Seker, "Trustworthiness requirements for manufacturing cyber-physical systems," *Procedia Manufacturing*, vol. 11, pp. 973–981, 2017.

[11] Z. Yu, L. Zhou, Z. Ma, and M. A. El-Meligy, "Trustworthiness modeling and analysis of cyber-physical manufacturing systems," *IEEE Access*, vol. 5, pp. 26 076–26 085, 2017.

[12] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, *Multiple Viewpoint Contract-Based Specification and Design*. Springer Berlin Heidelberg, 2008, pp. 200–225.

[13] *VDI Guideline 3682 Formalised Process Descriptions.*, Beuth Verlag Std., 2015.

[14] S. Biffl, J. Musil, A. Musil, K. Meixner, A. Lüder, F. Rinker, D. Weyns, and D. Winkler, "An Industry 4.0 Asset-Based Coordination Artifact for Production Systems Engineering," in *Int. Conf. Busi. Inf.* IEEE, 2021.

[15] T. Müller, S. Kamm, A. Löcklin, D. White, M. Mellinger, N. Jazdi, and M. Weyrich, "Architecture and knowledge modelling for self-organized reconfiguration management of cyber-physical production systems," *International Journal of Computer Integrated Manufacturing*, vol. 36, no. 12, pp. 1842–1863, Sep. 2022.

[16] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[17] Object Management Group, "Business Process Model and Notation (BPMN), Version 2.0," Object Management Group, Tech. Rep. formal/2011-01-03, Jan. 2011.

[18] H. Rahmani, S. Biffl, K. Meixner, D. Hoffmann, A. Lüder, and D. Winkler, "Business risk analysis of production variants considering technical dependencies," in *Int. Conf. Busi. Inf.* IEEE, Sep. 2024, pp. 178–187.

[19] Biffl, S., "Introduction to procan.do," QSE Technical Report QSE 2025-02, TU Wien, 2025.

[20] S. Biffl, S. Kropatschek, K. Meixner, D. Hoffmann, and A. Lüder, "Configuring and validating multi-aspect risk knowledge for industry 4.0 information systems," in *Int. CAiSE*. Springer, 2024, pp. 492–508.

[21] J. Herzog, H. Röpke, and A. Lüder, "Analysis of the reusability of modules in automotive assembly," in *IEEE ETFA*, 2021, pp. 1–8.

[22] K. Meixner, K. Feichtinger, R. Rabiser, and S. Biffl, "A reusable set of real-world product line case studies for comparing variability models in research and practice," in *Int. SPLC. Vol. B*. ACM, 2021.

[23] T. D. Thomas Bauernhansl, Manuel Fechter, *Entwicklung und Demonstration einer wandlungsfähigen Forschungsproduktion*. Springer, 2020.

Enhancing Public Safety Situational Awareness Using Edge Intelligence

Pedro Lira, Stefano Loss, Karine Costa, Daniel Araújo, Aluizio Rocha Neto, Nelio Cacho, Thais Batista, Everton Cavalcante, Frederico Lopes, Eduardo Nogueira
Federal University of Rio Grande do Norte
Natal, Brazil
pedro.varela.117@ufrn.edu.br, momoloss10@gmail.com, {karine.piacentini, daniel, aluizio}@imd.ufrn.br,
neliocacho@dimap.ufrn.br, thaisbatista@gmail.com, everton.cavalcante@ufrn.br, {fred, eduardo}@imd.ufrn.br

Abstract—**Real-time video analytics powered by artificial intelligence (AI) enables public safety agents to effectively perceive and respond to dynamic environments. However, processing large-scale video streams introduces computational and latency challenges. This work presents a framework that combines edge and cloud computing to facilitate efficient AI-based processing of video streams for public safety applications. We evaluated the framework's performance in a face recognition task by comparing edge and cloud processing. Our initial results demonstrate that edge processing achieves lower total latency compared to cloud processing despite higher inference times, primarily due to reduced transmission overhead. The framework also achieves high accuracy in recognition tasks, though with trade-offs in recall.**

Keywords-public safety; situational awareness; edge intelligence; stream analytics

I. Introduction

Situational awareness (SA) refers to the ability to perceive environmental factors, understand their significance, and anticipate future developments [1]. In contrast to traditional methods relying on radio communication and manual reporting, which present significant delays and inefficiencies, modern technologies like surveillance cameras and body-worn devices can enable real-time data collection and dissemination, thereby improving SA and decision-making [2].

Public safety operations require quick detection, analysis, and response to incidents. To comply with these requirements, real-time processing of video streams can be used to detect threats and support decision-making. While artificial intelligence (AI) techniques can significantly enhance this kind of advanced analysis, the large volumes of data to be handled and the high computational demand of intelligent models typically require cloud resources, which introduce latency due to data transmission and depend on reliable connectivity [3]. An alternative to alleviate these issues is processing AI close to the data source through edge intelligence, i.e., the convergence of AI-based task processing and edge computing. This kind of approach can reduce latency and bandwidth usage while ensuring continuity in low-connectivity environments. Nonetheless, the resource constraints of edge devices and bandwidth costs of cloud transmission demand a careful task distribution strategy.

This paper addresses these issues through SAALSA [4], a framework designed to enable efficient, low-latency video analytics by combining edge and cloud computing. We instantiated SAALSA into a public safety scenario, including the real-time identification of individuals based on face recognition resulting from AI-based processing of video streams. We assessed SAALSA's performance for this task by comparing edge and cloud processing in terms of latency and accuracy. Our preliminary findings have demonstrated the potential of edge intelligence to support critical decision-making in public safety operations.

The remainder of this paper is organized as follows. Section II brings an overview of SAALSA. Section III describes a face recognition use case in public safety that we utilize to demonstrate the framework. Section IV reports a preliminary evaluation of SAALSA's performance in AI-driven face recognition regarding edge and cloud-based processing latency and accuracy. Section V points out final remarks and directions for future work

II. A Framework for AI-driven Stream Analytics in the Edge

SAALSA addresses the fundamental challenge of balancing computational efficiency with latency requirements for AI-based stream analytics [4]. Public safety operations often occur under unstable or absent network conditions, requiring solutions that function independently of the cloud. AI-powered video analytics at the edge addresses this need by directly enabling real-time tasks, such as object detection and face recognition, on local devices. This enables the provision of timely insights that support faster and more accurate decision-making.

The SAALSA's architecture, depicted in Fig. 1, allows distributing processing tasks across three tiers. The *Data Source Tier* captures and collects raw data (e.g., audio and video streams and geolocation data) from various devices. The *Edge Tier* handles initial data processing near data sources, reducing latency and minimizing data transmission overhead to the cloud. The *Cloud Tier* handles computationally intensive tasks and provides centralized coordination and storage. Unlike traditional cloud-based architectures that send raw data to remote servers for processing, SAALSA collects data from several sources,

This work is supported by the SPICI project, funded by the Brazilian Innovation Agency – FINEP (grant 2827/22).

Manuscript received May 20, 2025; July 26, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025.
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.88

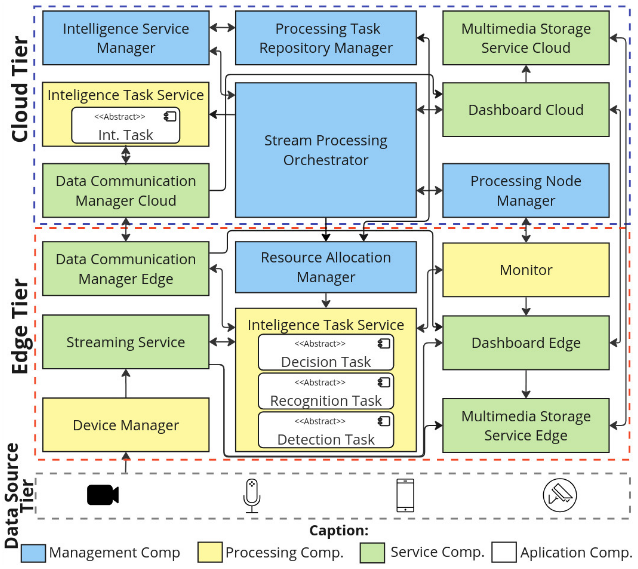


Figure 1. Main components of the SAALSA framework.

processes it at the edge to reduce latency and bandwidth use, and offloads it to the cloud for intensive tasks or long-term storage. Using SAALSA, it is possible to convert real-time data streams into actionable insights in public safety scenarios.

SAALSA comprises processing, management, and service components. Edge nodes handle streaming, detection, and recognition, while the cloud coordinates orchestration and complex analytics. Services include task scheduling, monitoring, multimedia storage, and user interfaces. This modular architecture enables adaptive, distributed video analytics tailored to public safety operations.

Data flows from the *Data Source Tier* to the *Edge Tier*, where the *Device Manager* and *Streaming Service*, built with Kurento,[1] handle media streaming. The *Multimedia Storage Service* records video streams and synchronizes them with the cloud when possible. A shared *Dashboard* provides real-time visualization with geolocation. FogFlow[2] orchestrates tasks between edge and cloud, with the *Resource Allocation Manager* coordinating task distribution between edge and cloud tiers. The *Intelligence Task Service* performs AI-driven task processing by using GStreamer[3] and DeepStream,[4] while Qdrant[5] supports face recognition. Finally, the *Data Communication Manager* handles inter-tier communication and data synchronization.

SAALSA implements dynamic task distribution based on computational requirements and network conditions. Initial processing tasks, such as detection and tracking, occur at the edge to minimize latency. Conversely, intensive tasks, including feature extraction and database queries, can be offloaded to the cloud when network conditions permit and computational demands exceed the edge's capabilities.

III. Face Recognition Use Case in Public Safety

We implemented a face recognition pipeline to demonstrate the feasibility of SAALSA. This use case represents a common public safety requirement where officers need to rapidly identify individuals in the field. Detected faces are first matched locally; if no match is found, embeddings are sent to the cloud. Results are annotated on-screen, enabling efficient, low-latency recognition adapted to hardware constraints.

The pipeline comprises three sequential stages that process video frames for accurate face recognition. Face detection employs the NVIDIA FaceNet model[6] to extract faces within bounding boxes [5]. Next, the FaceNet convolutional neural network [6] performs feature extraction, generating a feature vector (embedding) that maps each face to a compact Euclidean space. Finally, face recognition is achieved by comparing the extracted embedding to those stored in a vector database. Considering the incident response scenario, we adopted high-performance models that strike a balance between robustness and low latency, meeting the resource constraints of edge environments and the variability of public safety conditions.

Given the unpredictable nature of public safety scenarios, we also implemented an accumulation strategy to improve recognition accuracy. The strategy operates by assigning unique identifiers to each detected face, extracting embeddings, and identifying the closest match in the database for each frame. Recognition results are stored for N frames, after which the most frequent recognition is selected, and an average distance is computed for the final decision. This strategy prioritizes precision over recall, a crucial feature for public safety applications where false positives can have severe consequences.

IV. Evaluation

Experimental setup. We evaluated SAALSA using two configurations representing typical deployment scenarios. The edge configuration utilized an NVIDIA Jetson Nano[7] 4 GB (ARM Cortex-A57, 128 CUDA cores). The cloud configuration utilized a server equipped with an Intel i5-9300H processor, 64 GB of RAM, and a GTX 1650 GPU.

The evaluation considered streaming a recorded video from a simulated device to the edge and the cloud (see Fig. 2). The stream, sent via GStreamer using the RTSP protocol, was processed by a DeepStream-based face recognition pipeline (Section III), which used Qdrant for vector-based identity retrieval. Both setups used identical versions of DeepStream (6.0) and Qdrant (1.12.1), with consistent configurations and quantization levels: `INT8` for detection and `FP32` for embedding. We implemented tracking using NVIDIA's discriminative correlation filter. While we are aware that the hardware used in our evaluation does not currently represent the state-of-the-art of

[1] https://kurento.openvidu.io/
[2] https://fogflow.readthedocs.io/
[3] https://gstreamer.freedesktop.org/
[4] https://developer.nvidia.com/deepstream-sdk
[5] https://qdrant.tech/
[6] https://catalog.ngc.nvidia.com/orgs/nvidia/teams/tao/models/facenet
[7] https://developer.nvidia.com/embedded/jetson-nano

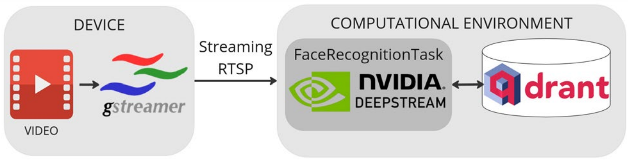


Figure 2. Face recognition setup used in the evaluation.

specialized AI hardware, we aimed to provide valuable insights into trade-offs between edge and cloud processing.

In addition to edge-only and cloud-only experiments, we evaluated a hybrid configuration where detection is performed at the edge, and only cropped faces are sent to the cloud for recognition. We measured transmission times for both full frames (1920 × 1080, RGB) and cropped faces (160 × 160), encoded in `base64`. We also established a secure Tailscale[8] network to simulate realistic cloud latency.

It is worth mentioning that our study focused on the overall pipeline processing rather than the latency due to transmitting data, as network conditions can introduce variability that must be accounted for in real-world deployments. Future work can explore controlled environments with dedicated network configurations to provide a more precise evaluation of transmission delays in edge and cloud scenarios.

Data sets. We constructed the dataset used in the evaluation from two publicly available face recognition datasets: CelebA [7] and Labeled Faces in the Wild (LFW) [8]. CelebA comprises 10,177 individuals, while LFW includes 5,749 individuals. For this study, we randomly selected 5,000 individuals from CelebA and 800 individuals from LFW, ensuring each individual had one to four images. This selection resulted in a total of 5,800 individuals and 9,199 images. Additionally, four volunteers from our research group contributed three images each, captured from different face angles (frontal and both sides profiles), adding 12 more images to the experimental dataset. We used two videos: Video V1 (duration 1’45”) features a single individual per frame for accurate timing analysis, and Video V2 (duration 3’10”) features the four volunteers in dynamic outdoor settings to assess the model’s robustness under real-world conditions.

A. *Processing Time Assessment*

The first experiment evaluated the computational cost of each stage in the face recognition pipeline, aiming to identify performance bottlenecks. This experiment utilized Video V1, which includes one person per frame, enabling consistent measurement across all frames. We timed four tasks: (i) face detection, (ii) face tracking, (iii) embedding extraction, and (iv) database query, on both edge and cloud environments. As shown in Table I, all stages were faster in the cloud. Embedding extraction was the most expensive task, especially on the edge (288.37 ms per frame), due to a non-optimized ONNX model[9] that did not fully exploit DeepStream’s acceleration. In contrast, face detection used a native DeepStream model, enabling much faster inference. Database queries also exhibited higher latency on the edge, primarily due to slower communication between DeepStream and Qdrant, limited memory bandwidth, and lower processing capacity.

TABLE I
AVERAGE PROCESSING TIME FOR EACH PIPELINE OPERATION

Operation	Cloud	Edge
Face detection	4.73 ± 1.02 ms	47.27 ± 1.96 ms
Face tracking	2.47 ± 2.31 ms	10.48 ± 11.07 ms
Embedding extraction	8.85 ± 4.14 ms	288.37 ± 10.58 ms
Database query	7.05 ± 1.26 ms	29.87 ± 3.20 ms

TABLE II
TOTAL PROCESSING TIME PER FRAME

Metrics	Edge	Hybrid	Cloud
Average frame processing time	375.99 ms	65.74 ms	23.20 ms
Average frame transmission time	108.85 ms*	606.75 ms	3,019.42 ms
Total processing time per frame	484.84 ms	672.49 ms	3,042.62 ms

*Time to transmit a Full HD frame via Gigabit Ethernet

Table II summarizes the total frame processing time. Edge processing had lower overall latency than cloud, despite slower inference. The hybrid approach, which balanced computation and communication, was slightly slower than the edge-only approach. These results reflect the trade-offs between computation and transmission, as well as the benefits of minimizing unnecessary data transfer.

B. *Recognition Assessment*

The second experiment evaluated the face recognition performance by using Video V2. To analyze the accuracy of the system, we employed precision and recall metrics, considering two parameters: the number of accumulated frames used to confirm an identity and a Euclidean distance threshold T that defines a valid match. Recognition was considered correct (true positive, *TP*) when the average distance between the detected face and its closest match in the database was below T, and the predicted identity matched the ground truth. Conversely, a false positive (*FP*) occurred when the distance was acceptable, but the identity was incorrect, while a false negative (*FN*) indicated a correct identity with a distance above the threshold. Precision is computed using Equation 1, while recall is computed using Equation 2:

$$Precision = \frac{TP}{TP + FP} \quad (1) \qquad Recall = \frac{TP}{TP+FN} \quad (2)$$

According to the results shown in Fig. 3, precision reached 100% upon accumulating $N = 110$ frames and $T = 0.8$, indicating no false positives. However, recall remained at 0.39 due to a high false negative rate. This configuration prioritizes accuracy over coverage and is suitable for public safety applications where minimizing false alarms is crucial. We observed that recall improved significantly when we relaxed the threshold; however, this came at the cost of reduced precision, with an increase in false positives. This trade-off illustrates the balance between identifying as many individuals as possible and maintaining a high level of confidence in each recognition. These findings

[8] https://tailscale.com/

[9] https://onnx.ai/

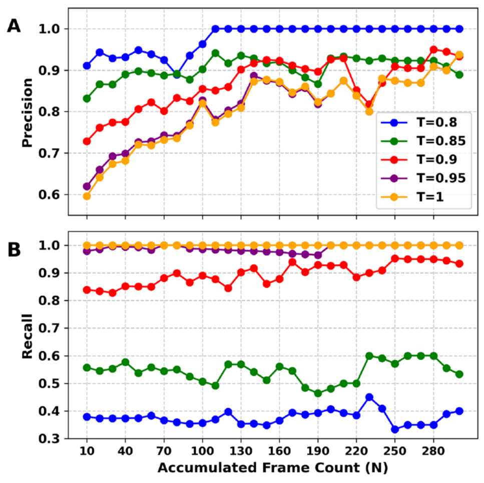


Figure 3. Precision and recall results for face recognition for different accumulation frame counts (*N*) and Euclidean distance thresholds (*T*).

suggest that in public safety scenarios, where reliability is paramount, using stricter parameters can help ensure that only highly confident identifications are acted upon, even if fewer matches are detected overall.

V. Conclusion

This work presented preliminary results for an edge-cloud video analytics framework tailored to public safety applications. Our initial experiments demonstrated the feasibility of edge processing for real-time video analytics, with edge deployment achieving the lowest total latency despite computational constraints. The face recognition use case illustrated the framework's potential while highlighting key trade-offs between accuracy and performance. The accumulation strategy successfully eliminates false positives, which is crucial for public safety applications, though at the cost of reduced recall.

The preliminary evaluation revealed several key insights into the performance of edge-cloud video analytics. Despite computational constraints, edge processing achieves lower total latency due to reduced transmission overhead, challenging the assumption that cloud processing is always superior for AI-driven tasks. The accumulation strategy effectively eliminates false positives but at the cost of increased false negatives, representing a critical trade-off in public safety applications. Additionally, selective offloading of computationally intensive tasks shows promise but requires careful consideration of network conditions and latency requirements.

Our future work will address current limitations through comprehensive evaluation and optimization, considering modern hardware, such as NVIDIA Jetson Orin and Raspberry Pi 5, in the evaluation. Future evaluation will include large-scale testing with realistic public safety scenarios and dynamic task distribution algorithms for optimal edge-cloud task allocation. An energy efficiency analysis will also examine the implications for power consumption and battery life.

References

[1] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," *Human Factors*, vol. 37, no. 1, pp. 32–64, Mar. 1995.

[2] D. Minoli, A. Koltun, and B. Occhiogrosso, Situational awareness for law enforcement and public safety agencies operating in smart cities – Part 2: Platforms, in S. Rani, V. Sai, and R. Maheswar, Eds. *IoT and WSN based smart cities: A machine learning perspective*. Switzerland: Springer International Publishing, 2022, pp. 139–162.

[3] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 200–10 232, Oct. 2020.

[4] S. Loss et al., "A framework for live situational awareness in stream-based 5G applications," in *1st IEEE Latin American Conference on Internet of Things*. USA: IEEE, 2025, to appear.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition*. USA: IEEE, 2016, pp. 779–788.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition*. USA: IEEE, 2015, pp. 815–823.

[7] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep Learning Face Attributes in the Wild," in *2015 IEEE International Conference on Computer Vision*. USA: IEEE, 2015, pp. 3730–3738.

[8] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, *Labeled Faces in the Wild: A database for studying face recognition in unconstrained environments*. University of Massachusetts, Amherst, USA, Tech. Rep. 07-49, October 2007.

Efficient Neural Network Reduction for AI-on-the-edge Applications through Structural Compression

Adriano Puglisi, Flavia Monti, Christian Napoli and Massimo Mecella
Department of Computer, Control, and Management Engineering Antonio Ruberti
Sapienza Università di Roma, Rome, Italy
Email: {puglisi, monti, napoli, mecella}@diag.uniroma1.it

***Abstract*—Modern neural networks often rely on overparameterized architectures to ensure stability and accuracy, but in many real-world scenarios, large models are unnecessarily expensive to train and deploy. This is especially true in Internet of Things (IoT) and edge computing scenarios, where computational resources and available memory are severely limited. Reducing the size of neural networks without compromising their ability to solve the target task remains a practical challenge, especially when the goal is to simplify the architecture itself, not just the weight space. To address this problem, we introduce ImproveNet, a simple and general method that reduces the size of a neural network, without compromising its ability to solve the original task. The approach does not require any pre-trained model, specific architecture knowledge, or manual tuning. Starting with a standard-sized network and the standard training configuration, ImproveNet verifies the model's performance during training. Once the performance requirements are met, it reduces the network by resizing feature maps or removing internal layers, thus making it ready for AI-on-the-edge deployment and execution.**

***Index Terms*—IoT, Edge AI, Deep Model Optimization, Neural Network Compression**

I. Introduction

The Internet of Things (IoT) concept, first introduced by Ashton in 1999, describes a system in which physical objects equipped with sensors are connected to the Internet and used to collect data from the environment. Since then, the idea has evolved rapidly and today includes billions of devices that can communicate with each other and exchange information in real time. There are currently over seven billion IoT devices in the world, and this number is expected to exceed twenty billion in the coming years.

As the number of connected devices increases exponentially, the amount of data generated is also growing. Although this data may contain useful information, it is often affected by noise, redundancies or errors [1]. As a result, traditional processing methods are no longer sufficient and increasing machine learning techniques are being used to extract knowledge from collected data. However, running machine learning models directly on IoT devices is extremely complex. These systems, also called *edge devices*, are characterized by limited resources, limited memory, low computational power, and stringent energy constraints. For these reasons, running large models locally (*on-device*) is often impractical.

An alternative solution could be to send the data to a remote server (*cloud*), where heavier models can be run without hardware constraints. However, in many real-world scenarios this option is limited or unacceptable, either for latency reasons (such as in real-time applications) or for privacy reasons (in healthcare, industrial, or personal contexts). In these scenarios, keeping the computation local is the only sustainable choice, provided that the model is light enough to be run safely and efficiently on the device.

To address this trend towards *AI-on-the-edge* (a.k.a. Edge AI), we propose ImproveNet, a simple method that reduces the size of a neural network directly during training, while ensuring that the model maintains the required performance. The approach starts with complete architecture, which is then progressively reduced as training progresses.

Block removal and filter reduction are the two structural alterations that lead to reduction. Removing a block means eliminating entire sequential portions of the network, each composed of one or more convolutional or linear layers, with a direct impact on the depth of the model. Filter reduction, on the other hand, consists in decreasing the number of output channels in the convolutional or dense layers, with the effect of reducing the width of the intermediate representations. Both operations allow us to simplify the architecture in a controlled way, keeping the capacity of the model within acceptable thresholds.

In the following sections, we describe in detail how this strategy works and analyze its application in different scenarios. In particular, Section II presents the main existing works dedicated to the reduction of neural networks through pruning, quantization or architectural simplification. Section III introduces the logic of ImproveNet and how reduction operations are integrated into the training cycle. Finally, in Section IV, we report the experimental results obtained by applying the method to the ESA-ADB dataset, using two autoencoders, one linear and one fully convolutional.

II. Related Works

In our application context, oriented to industrial scenarios and constrained by efficiency and compatibility requirements with edge systems, requirements analysis has led to narrowing the focus to fully convolutional or linear networks. Despite this choice, the comparison with existing compression techniques

Manuscript received May 19, 2025; revised August 27, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025.
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.89

TABLE 1 - COMPARISON OF MODEL COMPRESSION TECHNIQUES

Property	**ImproveNet**	**Structured Pruning** [7] [8]	**Unstructured Pruning** [9] [10]	**Distillation** [11]
Reduction type	Structural	Structural	Sparse	Knowledge transfer
Granularity	Blocks, Channels, Neuron	Filters	Weights	-
Inference time	Reduced	Reduced	Same	Reduced
Memory footprint	Reduced	Reduced	Same	Reduced
Compression ratio	High	High	Same	Reduced
Performances retention	Preserved	Not guaranteed	Not guaranteed	Teacher-dependent
Loss stability	Controlled	Requires retraining	Requires retraining	Regularized
Training Time	High	High	Low	Low
Architecture agnostic	Yes	No	No	Yes
Self contained	Yes	Yes	Yes	No
Repeatability	Yes	[7] Available in Caffe (Python) [8] Available in PyTorch	[9] Not available [10] Official not available (3rd part)	[11] Official not available (3rd part)
Neural Network Supported	Linear & Fully Convolutional	CNN	CNN	Any

will be conducted in a fair way, highlighting for each approach the context of validity and the reference architectures to which it applies. Several methods have been proposed to reduce the size and complexity of neural networks, particularly in applications with limited computational or memory resources, such as embedded or IoT devices. Most existing techniques are based on static approaches, including quantization [2], pruning [3], and knowledge distillation [4] [5], and are typically applied after training. Table 1 summarizes the main differences between ImproveNet and other model compression techniques, i.e., Structured Pruning, Unstructured Pruning, and Distillation.

III. IMPROVENET

Unlike traditional techniques based on neural importance, induced sparsity in weights, or post-training strategies, ImproveNet takes a completely different approach. The method acts directly during the training process, progressively reducing the network only when performance reaches predefined thresholds. This reduction occurs without requiring a fully trained model or the use of external heuristics.

The method takes as input the initial model together with all the components needed for training, such as the dataset, the optimizer, the metrics estimator, the loggers, and the performance constraints to be achieved such as the loss $\mathcal{L}_{target}$ and the accuracy $\mathcal{M}_{target}$. At regular intervals, the optimizer checks whether the current model M satisfies the target requirements. These requirements are expressed as constraints on global quantities, such as the loss $\mathcal{L}(M)$ and the accuracy metric $\mathcal{M}(M)$, as formalized in the following equation

$$\mathcal{L}(M) \leq \mathcal{L}_{target} \wedge \mathcal{M}(M) \geq \mathcal{M}_{target}$$

If the conditions are satisfied, ImproveNet applies a structural transformation to the network, choosing between reducing the number of channels and removing an internal block.

The type of reduction applied is managed dynamically based on the size of the current model compared to the initial one. The first two attempts apply a channel reduction and a simple block removal respectively. After this, the method computes the reduction ratio between the current network and the original one. If this ratio is still higher than the first threshold, priority is given to block removal, alternating every three attempts with a filter reduction. When the ratio drops below the first but above the second threshold, the two types of reduction are alternated more frequently (once every two attempts). Finally, only channel reduction is performed below the second threshold, preventing further excessive structural eliminations.

Using this technique, the model can gradually reduce while preserving its structural balance and avoiding excessive compression in subsequent training phases. To avoid repeated or harmful activities, the system also considers the number of reductions performed previously.

An additional protection mechanism is activated in case the network starts to stagnate. If the model does not converge within a certain number of iterations and the reduction attempts exceed half of the maximum expected number, ImproveNet performs a controlled reallocation of the architecture. The goal is to prevent excessive compression from trapping the model in non-ideal local minima.

Finally, if at the end of a reduction cycle the model fails to stably maintain the convergence conditions, the system restores

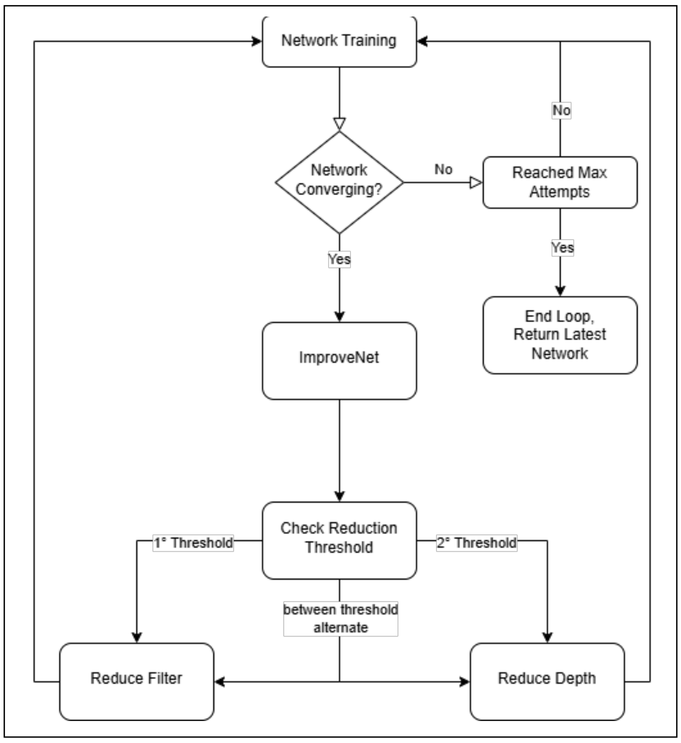


Figure 1 - Schematic representation of the ImproveNet workflow.

the last effective configuration, i.e. the last architecture that satisfied the performance constraints. An alternative strategy is then adopted, for example switching from block removal to filter reduction, or vice versa. This rescue mechanism ensures that the compression process does not irreversibly compromise the optimization capacity of the network.

The described structural transformations are based on the reduction of the number of channels (i.e. of input and output channels) and on the removal of entire blocks, which include sequences of convolutional or linear layers. Both operations take into account both the number of attempts already made and the size of the current model compared to the original one, Figure 1 shows the full workflow underlying the ImproveNet procedure, highlighting how the system monitors the training progress and applies structural reductions when the target conditions are met.

IV. Preliminary Results

To evaluate the performance of ImproveNet, one approach could have been a comparison with the methods in Table 1, but since most of them did not provide official source code or were not implemented in PyTorch or the code was available but did not work properly, we decided to test our method using ESA-ADB dataset [6], a recognized benchmark for multivariate time-series anomaly detection based on real data from space missions. We chose this dataset because it is representative of an edge environment where there is a need for small and compact models, suitable for resource constrained environments and a real-time operational context. The dataset is composed of three missions from which we selected Mission 1, composed of 76 channels, 58 of which are target channels and are splitted into 4 subsystems. Mission 1 includes 200 annotated events where 118 are anomalies, 78 nominal rare events (atypical but expected telemetry variations), and 4 communication gaps. We conducted experiments on a lightweight subset consisting of channels 41 to 46 as suggested by [6]. These channels were selected because they contain interesting but manageable anomalies, are useful for monitoring the health of the satellite, are relatively easy to visualize and analyze manually, and are independent of other channels or subsystems. The data were normalized in the range [0, 1] using a Min-Max scaling channel by channel, to ensure uniformity between the signal scales and avoid distortions in the calculation of the loss function.

The data were split respecting the temporal order of the observations where 70% was used for training, while the remaining 30% was divided into equal parts for validation and testing. The anomalous pattern is present exclusively in the test set, to train the model on the reconstruction of normal conditions. The time series were then transformed into fixed-length windows of 50 samples, with a stride of 50, obtaining sequences of the type (batch, 50, 6), where 6 represents the number of channels.

Training was conducted for a maximum of 100 epochs, using the Adam optimizer with a learning rate of 0.0001. The objective function used is a weighted combination of mean squared error (MSE) and mean absolute error (MAE), defined as

$$\mathcal{L} = \alpha \cdot MSE + (1 - \alpha) \cdot MAE$$

where ($\alpha = 0.5$) represents the balance between the two components. This formulation allows to penalize both large point errors (that the MAE effectively intercepts) and moderate diffuse errors (captured by the MSE), resulting particularly suitable for anomaly detection tasks.

We considered a Fully Convolutional Autoencoder (FCAE) and a Linear Autoencoder (LAE). ImproveNet was applied to these models, which operated during training by progressively reducing their structural complexity through functional criteria, generating compressed versions capable of maintaining comparable performance in terms of predictive accuracy.

All the experiments were run using an Intel core I5-13600KF, 32 GB of RAM and an RTX 3060 with 12 GB of VRAM. The results obtained show a significant reduction in the size of the models, in the case of the convolutional autoencoder, as shown in Table 2, the number of parameters goes from 130,886 in the original version to only 6,734 in the compressed network, with a reduction ratio of 94.85%, a reduction in the inference time from 3.4 ms to about 1 ms (3.4x faster) and a reduction in memory footprint of 94.07%. Similarly, in the linear model, as shown in Table 2, the parameters drop from 244,972 to 15,284 (93.76% reduction), and the inference time is reduced from 1.56 ms to about 0.15 ms (10.4x faster) and a memory footprint reduced of 93.37%.

Figure and Figure display anomaly detection results on a test sequence for linear autoencoder and convolutional models, respectively. In both situations, it is noted that the smaller

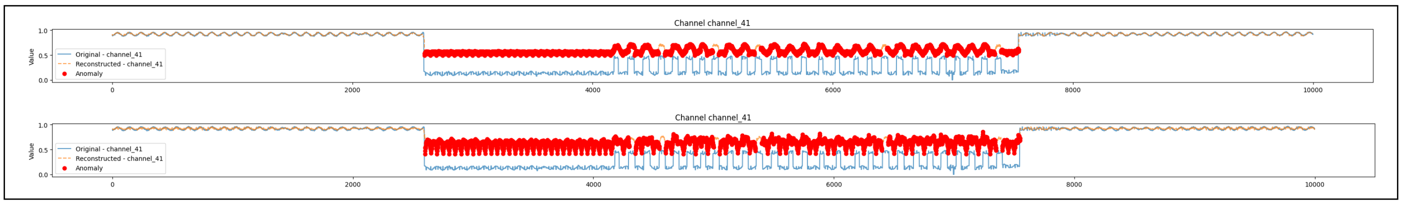


Figure 2 - Anomaly detection on a test sequence by the original convolutional model (the upper one) and reduced one (the lower one).

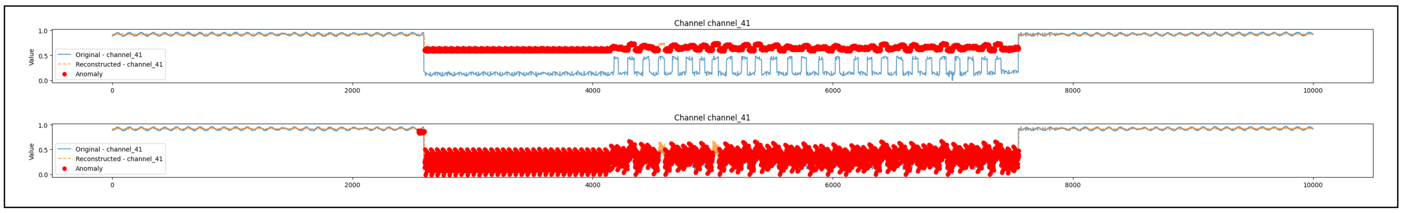


Figure 3 - Visual comparison between the original linear autoencoder (the upper one) and the reduced linear autoencoder model (the lower one).

TABLE 2 – COMPARISON BETWEEN THE LARGE AND THE REDUCED FULLY CONVOLUTIONAL AUTOENCODER (FCAE) AND LINEAR AUTOENCODER (LINEAR AE) ARCHITECTURES

	Total Param #	**CPU Inference Time (ms)**	**Size (MB)**
FCAE	130,886	3.4073	0.5046
Reduced FCAE	6,734	1.019	0.0299
Linear AE	244,972	1.5676	0.9397
Reduced Linear AE	15,284	0.1508	0.0623

version of the network (the lower time series in both images), parameters. For the convolutional model, the compressed obtained using ImproveNet, retains the ability to accurately detect abnormal patterns, despite the reduction in the number of network shows a slightly lower reconstruction quality than the original network, but the ability to detect anomalies remains unchanged, with comparable predictive performances. Similarly, in the case of the linear autoencoder, a slight degradation of the reconstruction is observed, but the anomaly is still correctly identified.

V. CONCLUSIONS

In this paper, we demonstrated the effectiveness of our approach for convolutional and linear networks in AI-on-the-edge scenarios where the size of the model is a central constraint. The ability of ImproveNet to progressively reduce architectural complexity while maintaining stable performance makes it particularly suitable for use in systems where the trade-off between accuracy and computational efficiency is essential.

In addition to the space/satellite, similar applications are found in sectors such as autonomous robotics, distributed industrial monitoring systems, wearable biomedical devices, and IoT infrastructures, all of which share the need to run neural models under limited computation and energy constraints.

A future development consists of directly integrating the compressed models generated by ImproveNet into real devices, evaluating their behavior on embedded hardware and low-power microcontrollers, in unsimulated operating conditions.

ACKNOWLEDGMENT

The work of Adriano Puglisi is supported by Thales Alenia Space and Regione Lazio, through the fellowships (CUP B83C23001060009), (SIGEM 22066DP000000035) *AI-based digital industrial processes for space systems production workflow.*

REFERENCES

[1] H. N. W. R. C. W. W. H. Z. Z. &. V. A. V. Dai, "Big data analytics for large-scale wireless networks: Challenges and opportunities," *ACM Computing Surveys (CSUR),* pp. 1-36, 2019.

[2] B. K. S. C. B. Z. M. T. M. H. A. .. &. K. D. Jacob, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 2704-2713, 2018.

[3] H. K. A. D. I. S. H. &. G. H. P. Li, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710,* 2016.

[4] T. G. I. &. S. J. Chen, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641,* 2015.

[5] R. C. a. A. N.-M. C. Buciluˇa, "Model compression," *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining,* p. 535–541, 2006.

[6] K. H. C. A. J. R. B. N. J. L. D. .. &. D. C. G. Kotowski, "European space agency benchmark for anomaly detection in satellite telemetry," *arXiv preprint arXiv:2406.17826,* 2024.

[7] J. H. Z. H. Z. H. Y. X. C. W. W. J. &. L. W. Luo, "ThiNet: Pruning CNN filters for a thinner net," *IEEE transactions on pattern analysis and machine intelligence,* vol. 41, no. 10, pp. 2525-2538, 2018.

[8] Y. K. G. D. X. F. Y. &. Y. Y. He, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866,* 2018.

[9] S. &. B. R. V. Srinivas, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149,* 2015.

[10] S. P. J. T. J. &. D. W. Han, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems,* vol. 28, 2015.

[11] G. V. O. &. D. J. Hinton, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531,* 2015.

Towards Optimized Arithmetic Circuits with MLIR

Louis Ledoux, Pierre Cochard, Florent de Dinechin
INSA Lyon, Inria, CITI, UR3720, 69621 Villeurbanne, France
{louis.ledoux, pierre.cochard, florent.de-dinechin}@insa-lyon.fr

***Abstract*—Numerical programs are typically conceived with real numbers in mind. However, programming languages operate at a lower abstraction level with fixed-width machine arithmetic. This abstraction gap limits the scope of legal arithmetic optimizations in compilers, in particular when targetting hardware.**

This work introduces a set of MLIR dialects that explicitly separate concerns between real-valued computation and low-level arithmetic representation. The RealArith dialect captures mathematical intent, enabling algebraic rewrites and approximation-aware transformations. The FixedPointArith dialect expresses quantized arithmetic with fine-grained control over bit widths. This separation enables arithmetic optimizations beyond those supported by conventional compilers. An example end-to-end lowering flow performs polynomial approximation, then generates fixed-point Horner-form architectures tailored for hardware synthesis. Early hardware results on signal processing benchmarks demonstrate the potential of this approach.

***Keywords*—MLIR, arithmetic optimization, fixed-point, polynomial approximation, high-level synthesis**

I. Introduction

Numerical programs are often written under the implicit assumption that operations behave like those over real numbers. However, modern compiler infrastructures, including MLIR, typically operate on machine-level formats such as fixed-width integers and floating-point numbers (IEEE754). These formats impose rigid evaluation semantics, limiting the set of legal arithmetic transformations.

For example, while addition is associative in real arithmetic, it is not in floating point. Similarly, expression fusion or algebraic rewrites that are mathematically valid may become unsafe or imprecise when executed with finite-precision types. These constraints hinder possible optimizations, particularly in domains like signal processing, scientific computing, and machine learning, where computations follow well-defined mathematical patterns such as matrix multiplication (GEMM), quantization, sparse accumulation, activation functions, or transcendental operations. These patterns are often amenable to algebraic simplification or approximation before being lowered to low-level arithmetic circuits, where further hardware-specific rewrites such as operator specialization and bitwidth tuning can be applied.

When compiling to hardware, such limitations can and should be relaxed. Hardware offers the freedom to implement arithmetic operators with arbitrary bit widths and optimized datapaths. Given the ability to tune precision and layout at the circuit level, designers can trade off accuracy, and area according to application needs. To fully exploit this flexibility, compiler flows must reason not only about bits, but also about the mathematical semantics of computation.

The Multi-Level Intermediate Representation (MLIR) [3] offers a promising foundation for building such flows. Originally developed to improve the compilation of machine-learning models, MLIR now also serves as a foundation for hardware-oriented compiler projects such as CIRCT[1] and Dynamatic[2]. Its extensible infrastructure enables modular modeling of programs across abstraction levels. In MLIR, these levels are described as *dialects*, each of which defines an Intermediate Representation (IR) with its own operations, types, and transformation rules. However, existing MLIR dialects remain tightly bound to machine arithmetic and lack a systematic way to express real-valued computation or reason about approximation.

This work introduces an arithmetic-aware MLIR flow that bridges high-level mathematical intent and hardware-oriented representation. Our contributions include two dialects with transformation passes and lowerings for arithmetic optimization and hardware generation:

- **RealArith** represents computations over real numbers, enabling semantic-preserving rewrites and symbolic approximation control.
- **FixedPointArith** expresses quantized arithmetic with precise control over fixed-point types and is designed to target hardware synthesis.

Building on these dialects, we implement a full lowering pipeline that transforms real-valued expressions into hardware-oriented fixed-point arithmetic. The pipeline performs approximation using external tools such as Sollya [1] and FloPoCo [2], generating Horner-form evaluators with precision-tuned datapaths. This enables the synthesis of optimized arithmetic accelerators directly from high-level mathematical IR. Our contributions are as follows:

(1) We design and implement the `RealArith` and `FixedPointArith` MLIR dialects to support multi-level arithmetic reasoning and transformation.
(2) We develop an approximation-aware lowering pipeline that translates real expressions into fixed-point arithmetic using Sollya and FloPoCo.
(3) We generate precision-tuned, pipelined Horner architectures, suitable for RTL and HLS-based synthesis.
(4) We demonstrate early hardware results on signal processing benchmarks, where our approach enables trade-offs between memory footprint and arithmetic complexity under an accuracy budget.

[1]https://circt.llvm.org/
[2]https://dynamatic.epfl.ch/

Manuscript received July 14, 2025; revised July 28, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.90

II. Background

A. Multi-Level Intermediate Representation (MLIR)

MLIR provides an extensible infrastructure for building compiler pipelines with multiple abstraction levels. Its core abstraction, the *dialect*, allows different computational models to coexist, enabling progressive lowering from high-level semantics to hardware-ready representations. Projects such as CIRCT and Dynamatic extend MLIR for hardware synthesis, motivating our use of dialects to capture arithmetic intent across levels of precision and abstraction.

B. Functional Audio Stream (FAUST)

Faust [5] is a domain-specific language for real-time digital signal processing, making it well-suited to capture the associated mathematical intent. It adopts a functional programming model and supports multiple backends, including C++, LLVM IR, and hardware targets. To enable FPGA deployment, the Syfala toolchain [6] compiles Faust-generated C++ via Vitis HLS, producing hardware for real-time audio. More recently, an MLIR backend has been introduced, enabling lowering of Faust programs to hardware-oriented IR flows.

C. Floating-Point Cores (FloPoCo)

FloPoCo is an open-source tool for generating parameterized arithmetic cores, particularly optimized for FPGA targets. Internally, FloPoCo is structured around a clear separation of concerns between high-level arithmetic modeling, such as real-valued polynomial and piecewise approximations, low-level datapath construction, including components like bit heaps, and target-specific mapping to FPGA resources. While this philosophy guides its architecture, these layers are currently intertwined in implementation and not explicitly exposed, making them difficult to access or reuse from external tooling.

These internal abstractions align naturally with MLIR's dialect model. Our work seeks to expose each layer as an explicit dialect, making FloPoCo's arithmetic reasoning and circuit synthesis capabilities more accessible and interoperable within MLIR-based hardware flows.

D. Polynomial Approximations and Horner Architectures

Polynomial approximation is a classical technique that enables efficient evaluation of functions using only additions and multiplications. This topic has been well studied in the literature: textbooks [2], [4] detail both the mathematical foundations and implementation strategies, including range reduction and hardware-oriented considerations.

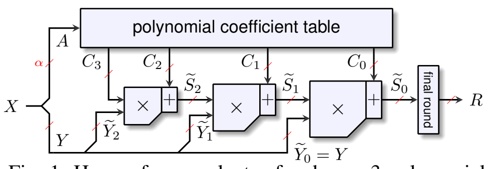


Fig. 1: Horner-form evaluator for degree-3 polynomials.

A univariate polynomial p of degree d over a real variable X has real-valued coefficients $C_i \in \mathbb{R}$. The Horner evaluation scheme is often used, since it involves only one multiplication per coefficient:

$$p(X) = C_0 + X(C_1 + \cdots + X(C_{d-1} + XC_d))). \quad (1)$$

Figure 1 represents a piecewise polynomial evaluator the fixed-point architecture for evaluating degree-3 polynomials using Horner's method. The input X is decomposed into two parts: the most significant α bits, denoted A, addresses a coefficient table holding 2^α polynomials. The remaining $w_X - \alpha$ bits, denoted Y, serve as the local offset for evaluation within the selected sub-interval.

In a fixed-point implementations (Figure 1), the smallest possible format of each coefficient C_i can be derived from the function and the accuracy constraints. Similarly, each Horner step may use a truncation $\tilde{Y}_i$ of Y to minimize the size of the corresponding multiplier [2] – this is implemented in FloPoCo.

The segmentation parameter α introduces a trade-off: increasing α reduces the required polynomial degree and hence arithmetic cost, but exponentially increases memory usage due to the 2^α coefficient sets. This trade-off must be co-optimized based on implementation constraints. MLIR provides a suitable infrastructure to capture this trade-off explicitly, enabling lowering strategies or automated heuristics to tune the arithmetic / memory balance in the generated hardware.

III. End-to-End Compilation Flow

A. System Integration Overview

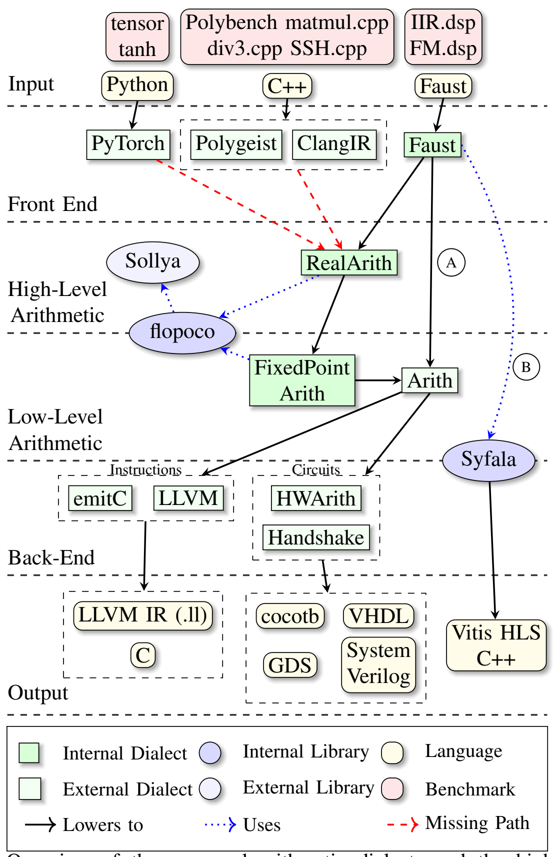
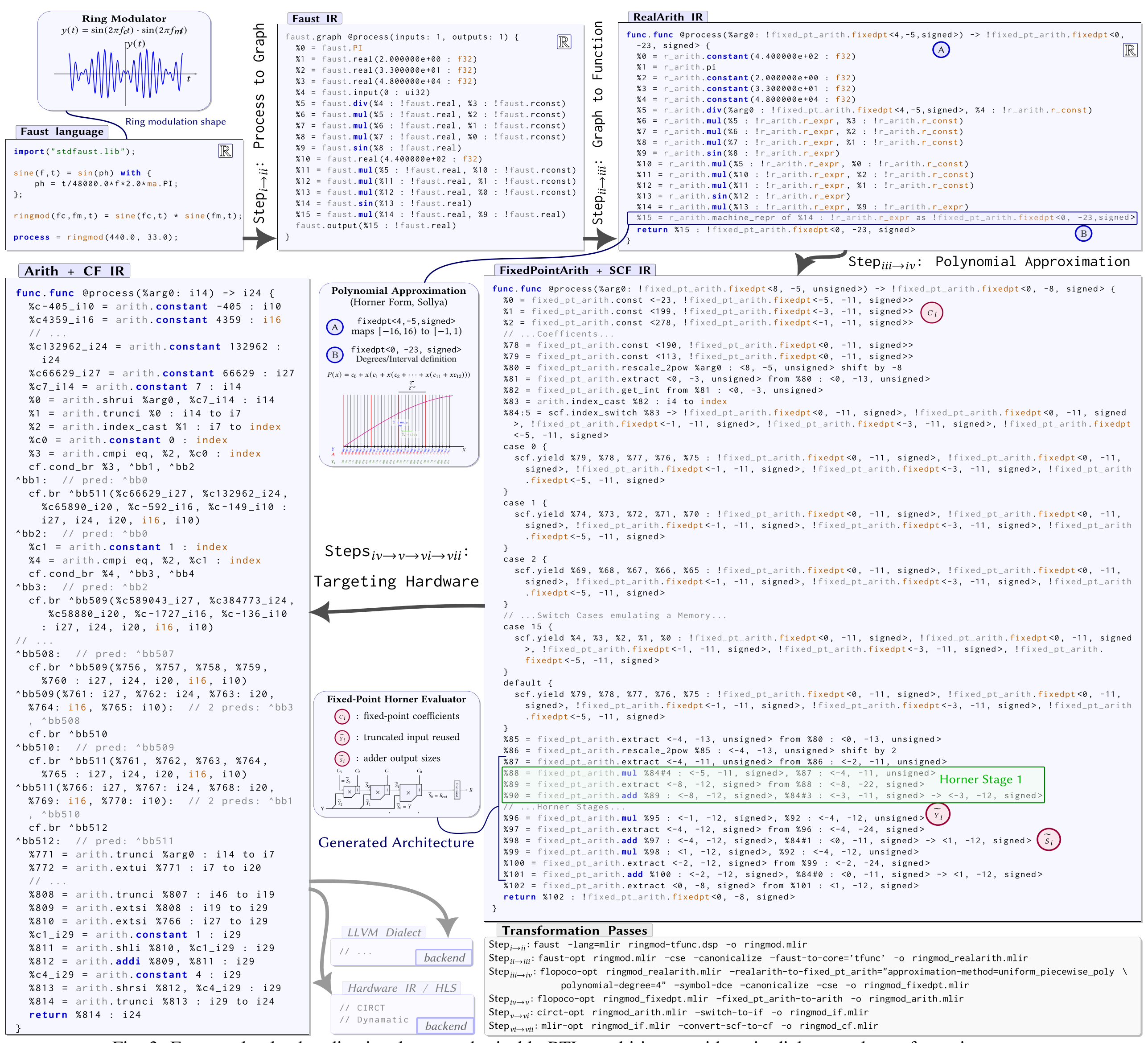


Fig. 2: Overview of the proposed arithmetic dialects and the high-level synthesis toolchain.


Ring Modulator
$y(t) = \sin(2\pi f_c t) \cdot \sin(2\pi f_m t)$
$y(t)$
t
Ring modulation shape
Step$_{i\to ii}$: Process to Graph
Step$_{ii\to iii}$: Graph to Function
Step$_{iii\to iv}$: Polynomial Approximation
Steps$_{iv\to v\to vi\to vii}$: Targeting Hardware
Polynomial Approximation
(Horner Form, Sollya)
A fixedpt<4,-5,signed> maps $[-16,16)$ to $[-1,1)$
B fixedpt<0, -23, signed> Degrees/Interval definition
$P(x) = c_0 + x(c_1 + x(c_2 + \cdots + x(c_{11} + xc_{12})))$
Fixed-Point Horner Evaluator
c_i : fixed-point coefficients
$\widetilde{Y}_i$: truncated input reused
$\widetilde{S}_i$: adder output sizes
Generated Architecture
Horner Stage 1
LLVM Dialect
backend
Hardware IR / HLS
backend

Faust language

```
import("stdfaust.lib");

sine(f,t) = sin(ph) with {
    ph = t/48000.0*f*2.0*ma.PI;
};

ringmod(fc,fm,t) = sine(fc,t) * sine(fm,t);

process = ringmod(440.0, 33.0);
```

Faust IR

```
faust.graph @process(inputs: 1, outputs: 1) {
  %0 = faust.PI
  %1 = faust.real(2.000000e+00 : f32)
  %2 = faust.real(3.300000e+01 : f32)
  %3 = faust.real(4.800000e+04 : f32)
  %4 = faust.input(0 : ui32)
  %5 = faust.div(%4 : !faust.real, %3 : !faust.rconst)
  %6 = faust.mul(%5 : !faust.real, %2 : !faust.rconst)
  %7 = faust.mul(%6 : !faust.real, %1 : !faust.rconst)
  %8 = faust.mul(%7 : !faust.real, %0 : !faust.rconst)
  %9 = faust.sin(%8 : !faust.real)
  %10 = faust.real(4.400000e+02 : f32)
  %11 = faust.mul(%5 : !faust.real, %10 : !faust.rconst)
  %12 = faust.mul(%11 : !faust.real, %1 : !faust.rconst)
  %13 = faust.mul(%12 : !faust.real, %0 : !faust.rconst)
  %14 = faust.sin(%13 : !faust.real)
  %15 = faust.mul(%14 : !faust.real, %9 : !faust.real)
  faust.output(%15 : !faust.real)
}
```

RealArith IR

```
func.func @process(%arg0: !fixed_pt_arith.fixedpt<4,-5,signed>) -> !fixed_pt_arith.fixedpt<0,
  -23, signed> {
  %0 = r_arith.constant(4.400000e+02 : f32)
  %1 = r_arith.pi
  %2 = r_arith.constant(2.000000e+00 : f32)
  %3 = r_arith.constant(3.300000e+01 : f32)
  %4 = r_arith.constant(4.800000e+04 : f32)
  %5 = r_arith.div(%arg0 : !fixed_pt_arith.fixedpt<4,-5,signed>, %4 : !r_arith.r_const)
  %6 = r_arith.mul(%5 : !r_arith.r_expr, %3 : !r_arith.r_const)
  %7 = r_arith.mul(%6 : !r_arith.r_expr, %2 : !r_arith.r_const)
  %8 = r_arith.mul(%7 : !r_arith.r_expr, %1 : !r_arith.r_const)
  %9 = r_arith.sin(%8 : !r_arith.r_expr)
  %10 = r_arith.mul(%5 : !r_arith.r_expr, %0 : !r_arith.r_const)
  %11 = r_arith.mul(%10 : !r_arith.r_expr, %2 : !r_arith.r_const)
  %12 = r_arith.mul(%11 : !r_arith.r_expr, %1 : !r_arith.r_const)
  %13 = r_arith.sin(%12 : !r_arith.r_expr)
  %14 = r_arith.mul(%13 : !r_arith.r_expr, %9 : !r_arith.r_expr)
  %15 = r_arith.machine_repr of %14 : !r_arith.r_expr as !fixed_pt_arith.fixedpt<0, -23,signed>
  return %15 : !fixed_pt_arith.fixedpt<0, -23, signed>
}
```

Arith + CF IR

```
func.func @process(%arg0: i14) -> i24 {
  %c-405_i10 = arith.constant -405 : i10
  %c4359_i16 = arith.constant 4359 : i16
  // ...
  %c132962_i24 = arith.constant 132962 :
    i24
  %c66629_i27 = arith.constant 66629 : i27
  %c7_i14 = arith.constant 7 : i14
  %0 = arith.shrui %arg0, %c7_i14 : i14
  %1 = arith.trunci %0 : i14 to i7
  %2 = arith.index_cast %1 : i7 to index
  %c0 = arith.constant 0 : index
  %3 = arith.cmpi eq, %2, %c0 : index
  cf.cond_br %3, ^bb1, ^bb2
^bb1:  // pred: ^bb0
  cf.br ^bb511(%c66629_i27, %c132962_i24,
    %c65890_i20, %c-592_i16, %c-149_i10 :
    i27, i24, i20, i16, i10)
^bb2:  // pred: ^bb0
  %c1 = arith.constant 1 : index
  %4 = arith.cmpi eq, %2, %c1 : index
  cf.cond_br %4, ^bb3, ^bb4
^bb3:  // pred: ^bb2
  cf.br ^bb509(%c589043_i27, %c384773_i24,
     %c58880_i20, %c-1727_i16, %c-136_i10
    : i27, i24, i20, i16, i10)
// ...
^bb508:  // pred: ^bb507
  cf.br ^bb509(%756, %757, %758, %759,
    %760 : i27, i24, i20, i16, i10)
^bb509(%761: i27, %762: i24, %763: i20,
  %764: i16, %765: i10):  // 2 preds: ^bb3
  , ^bb508
  cf.br ^bb510
^bb510:  // pred: ^bb509
  cf.br ^bb511(%761, %762, %763, %764,
    %765 : i27, i24, i20, i16, i10)
^bb511(%766: i27, %767: i24, %768: i20,
  %769: i16, %770: i10):  // 2 preds: ^bb1
  , ^bb510
  cf.br ^bb512
^bb512:  // pred: ^bb511
  %771 = arith.trunci %arg0 : i14 to i7
  %772 = arith.extui %771 : i7 to i20
  // ...
  %808 = arith.trunci %807 : i46 to i19
  %809 = arith.extsi %808 : i19 to i29
  %810 = arith.extsi %766 : i27 to i29
  %c1_i29 = arith.constant 1 : i29
  %811 = arith.shli %810, %c1_i29 : i29
  %812 = arith.addi %809, %811 : i29
  %c4_i29 = arith.constant 4 : i29
  %813 = arith.shrsi %812, %c4_i29 : i29
  %814 = arith.trunci %813 : i29 to i24
  return %814 : i24
}
```

FixedPointArith + SCF IR

```
func.func @process(%arg0: !fixed_pt_arith.fixedpt<8, -5, unsigned>) -> !fixed_pt_arith.fixedpt<0, -8, signed> {
  %0 = fixed_pt_arith.const <-23, !fixed_pt_arith.fixedpt<-5, -11, signed>>
  %1 = fixed_pt_arith.const <199, !fixed_pt_arith.fixedpt<-3, -11, signed>>
  %2 = fixed_pt_arith.const <278, !fixed_pt_arith.fixedpt<-1, -11, signed>>
  // ...Coefficents...
  %78 = fixed_pt_arith.const <190, !fixed_pt_arith.fixedpt<0, -11, signed>>
  %79 = fixed_pt_arith.const <113, !fixed_pt_arith.fixedpt<0, -11, signed>>
  %80 = fixed_pt_arith.rescale_2pow %arg0 : <8, -5, unsigned> shift by -8
  %81 = fixed_pt_arith.extract <0, -3, unsigned> from %80 : <0, -13, unsigned>
  %82 = fixed_pt_arith.get_int from %81 : <0, -3, unsigned>
  %83 = arith.index_cast %82 : i4 to index
  %84:5 = scf.index_switch %83 -> !fixed_pt_arith.fixedpt<0, -11, signed>, !fixed_pt_arith.fixedpt<0, -11, signed
    >, !fixed_pt_arith.fixedpt<-1, -11, signed>, !fixed_pt_arith.fixedpt<-3, -11, signed>, !fixed_pt_arith.fixedpt
    <-5, -11, signed>
  case 0 {
    scf.yield %79, %78, %77, %76, %75 : !fixed_pt_arith.fixedpt<0, -11, signed>, !fixed_pt_arith.fixedpt<0, -11,
      signed>, !fixed_pt_arith.fixedpt<-1, -11, signed>, !fixed_pt_arith.fixedpt<-3, -11, signed>, !fixed_pt_arith
      .fixedpt<-5, -11, signed>
  }
  case 1 {
    scf.yield %74, %73, %72, %71, %70 : !fixed_pt_arith.fixedpt<0, -11, signed>, !fixed_pt_arith.fixedpt<0, -11,
      signed>, !fixed_pt_arith.fixedpt<-1, -11, signed>, !fixed_pt_arith.fixedpt<-3, -11, signed>, !fixed_pt_arith
      .fixedpt<-5, -11, signed>
  }
  case 2 {
    scf.yield %69, %68, %67, %66, %65 : !fixed_pt_arith.fixedpt<0, -11, signed>, !fixed_pt_arith.fixedpt<0, -11,
      signed>, !fixed_pt_arith.fixedpt<-1, -11, signed>, !fixed_pt_arith.fixedpt<-3, -11, signed>, !fixed_pt_arith
      .fixedpt<-5, -11, signed>
  }
  // ...Switch Cases emulating a Memory...
  case 15 {
    scf.yield %4, %3, %2, %1, %0 : !fixed_pt_arith.fixedpt<0, -11, signed>, !fixed_pt_arith.fixedpt<0, -11, signed
      >, !fixed_pt_arith.fixedpt<-1, -11, signed>, !fixed_pt_arith.fixedpt<-3, -11, signed>, !fixed_pt_arith.
      fixedpt<-5, -11, signed>
  }
  default {
    scf.yield %79, %78, %77, %76, %75 : !fixed_pt_arith.fixedpt<0, -11, signed>, !fixed_pt_arith.fixedpt<0, -11,
      signed>, !fixed_pt_arith.fixedpt<-1, -11, signed>, !fixed_pt_arith.fixedpt<-3, -11, signed>, !fixed_pt_arith
      .fixedpt<-5, -11, signed>
  }
  %85 = fixed_pt_arith.extract <-4, -13, unsigned> from %80 : <0, -13, unsigned>
  %86 = fixed_pt_arith.rescale_2pow %85 : <-4, -13, unsigned> shift by 2
  %87 = fixed_pt_arith.extract <-4, -11, unsigned> from %86 : <-2, -11, unsigned>
  %88 = fixed_pt_arith.mul %84#4 : <-5, -11, signed>, %87 : <-4, -11, unsigned>
  %89 = fixed_pt_arith.extract <-8, -12, signed> from %88 : <-8, -22, signed>
  %90 = fixed_pt_arith.add %89 : <-8, -12, signed>, %84#3 : <-3, -11, signed> -> <-3, -12, signed>
  // ...Horner Stages...
  %96 = fixed_pt_arith.mul %95 : <-1, -12, signed>, %92 : <-4, -12, unsigned>
  %97 = fixed_pt_arith.extract <-4, -12, signed> from %96 : <-4, -24, signed>
  %98 = fixed_pt_arith.add %97 : <-4, -12, signed>, %84#1 : <0, -11, signed> -> <1, -12, signed>
  %99 = fixed_pt_arith.mul %98 : <1, -12, signed>, %92 : <-4, -12, unsigned>
  %100 = fixed_pt_arith.extract <-2, -12, signed> from %99 : <-2, -24, signed>
  %101 = fixed_pt_arith.add %100 : <-2, -12, signed>, %84#0 : <0, -11, signed> -> <1, -12, signed>
  %102 = fixed_pt_arith.extract <0, -8, signed> from %101 : <1, -12, signed>
  return %102 : !fixed_pt_arith.fixedpt<0, -8, signed>
}
```

```
// ...
```

```
// CIRCT
// Dynamatic
```

Transformation Passes

Step$_{i\to ii}$: `faust -lang=mlir ringmod-tfunc.dsp -o ringmod.mlir`
Step$_{ii\to iii}$: `faust-opt ringmod.mlir -cse -canonicalize -faust-to-core='tfunc' -o ringmod_realarith.mlir`
Step$_{iii\to iv}$: `flopoco-opt ringmod_realarith.mlir -realarith-to-fixed_pt_arith="approximation-method=uniform_piecewise_poly \`
`polynomial-degree=4" -symbol-dce -canonicalize -cse -o ringmod_fixedpt.mlir`
Step$_{iv\to v}$: `flopoco-opt ringmod_fixedpt.mlir -fixed_pt_arith-to-arith -o ringmod_arith.mlir`
Step$_{v\to vi}$: `circt-opt ringmod_arith.mlir -switch-to-if -o ringmod_if.mlir`
Step$_{vi\to vii}$: `mlir-opt ringmod_if.mlir -convert-scf-to-cf -o ringmod_cf.mlir`

Fig. 3: From real-valued audio signals to synthesizable RTL: multi-intent arithmetic dialects and transformation passes.

Figure 2 illustrates the integration of our arithmetic-aware flow within a high-level synthesis toolchain. While the diagram emphasizes signal processing use cases, the flow is designed to accommodate a broader range of inputs, such as AI models or polyhedral C++ code. Figure 2-(A) and -(B) denote baseline that bypass the proposed optimizations.

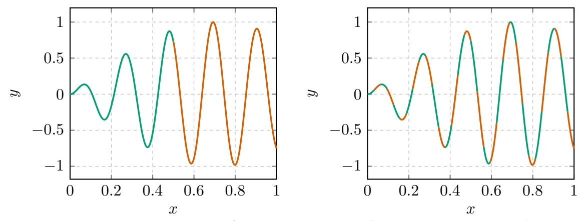
B. Multi-Level Arithmetic Dialects

We introduce two MLIR dialects that reflect distinct levels of arithmetic abstraction. The `RealArith` dialect operates over mathematical real numbers and supports symbolic expressions with both algebraic and transcendental operations. As illustrated by Figure 3-(B), it introduces the `machine_repr` operation, which defines the transition point from infinite-precision real arithmetic to a concrete fixed-point format suitable for implementation. The `FixedPointArith` dialect encodes quantized fixed-point arithmetic with operations. This dialect serves as an intermediate representation amenable to hardware synthesis and lowers directly to the core MLIR dialects (`Arith`).

C. Transformation and Pass Pipeline

Figure 3 illustrates the transformation pipeline across intermediate representations in our flow. The process begins from high-level real-valued expressions written in DSLs like Faust, which are then mapped to the `RealArith` dialect. Approximation is triggered by the insertion of a `machine_repr` operation, introduced by the `faust-opt` pass (Figure 3-(B)). This operation marks the boundary between real-valued computation and fixed-point implementation, as indicated by the absence of the $\mathbb{R}$ badge. The LSB of its return type determines the desired output precision.

This request is handled by a transformation pass that performs symbolic approximation using the Sollya library. The result is a fixed-point polynomial architecture expressed in Horner form. The corresponding evaluator is emitted with our `FixedPointArith` dialect, paired with `SCF` (Structured



(a) $\alpha = 1$, Degree=16 (b) $\alpha = 5$, Degree=4

Fig. 4: Uniform piecewise polynomial approximations.

Control Flow) Dialect to implement a case-based memory access over 2^α coefficient sets.

The generated architecture corresponds to the fixed-point Horner evaluator depicted in Figure 1. The fixed-point coefficients c_i are defined by the values yielded in each `scf.index_switch` case. Since each of the 2^α segments requires a polynomial of degree d, each case yields $d + 1$ constants, totaling $(d + 1) \cdot 2^\alpha$ coefficients. The truncated inputs $\widetilde{Y}_i$ are propagated through the stages, visible in the IR as shared `extract` and `rescale` operations. Adder output sizes $\widetilde{S}_i$ result from precision growth at each stage and correspond to the bitwidths seen in the intermediate addition results on lines `%88`–`%90`. Eventually, to target hardware synthesis tools, the IR is lowered to core dialects: `Arith`, `CF`, and `Builtin`. This includes scaling fixed-point into integers and converting control constructs into the `CF` dialect required by CIRCT or Dynamatic. Once in this form, downstream tools such as `dynamatic-opt` and `export-hdl` can be used to generate synthesizable HDL.

IV. Results Discussions

A. Experimental Setup and Methodology

We evaluate our compilation flow on a ring modulation algorithm expressed in Faust. The computation consists of the product of two sine waves, $y(t) = \sin(2\pi f_1 t) \cdot \sin(2\pi f_2 t)$, chosen for its relevance to real-time audio processing and the presence of a nonlinear transcendental function. Two baselines are considered (see two first rows of Table I). The first uses Syfala, which compiles Faust-generated C++ through Vitis HLS to produce synthesizable RTL (see Figure 2-Ⓑ). The second bypasses our proposed optimizations by lowering Faust-generated MLIR directly to the `arith` dialect in floating point, and applies the `mlir-opt -test-math-polynomial-approximation` pass to expand transcendental functions into `f32`-based polynomial approximations(see Figure 2-Ⓐ). Figure 4 shows two of our polynomial approximation configurations after range reduction to $[0, 1)$ of a subset of a full period. The visible alternation of segments reflects the piecewise scheme. The degree-16 case needs to store $2^1 \cdot (16+1) = 34$ coefficients, likely wider than the $2^5 \cdot (4 + 1) = 160$ narrower ones of the degree-4 case.

B. Hardware Results

Table I reports hardware usage across methods on the xc7z020-1clg400c FPGA. DSP usage increases with polynomial degree due to deeper pipelines and wider coefficients, which result in more arithmetic stages and facilitate automated DSP inference. Lower-degree piecewise configurations trade arithmetic for memory by increasing the number of segments. Both 32-bit floating-point baselines show high DSP usage due to mantissas being mapped to dedicated multipliers.

TABLE I: Resource usage across methods on xc7z020 FPGA.

Method (output precision)	**Poly. degree**	**Hardware resources**		
		LUT	**FF**	**DSP**
faust-syfala (32 bits)	*	3,765	3,142	32
faust-mlir (32 bits)	*	14,638	8,226	27
Uniform Piecewise Poly. Approx. (10 bits)	4	2,868	4,172	4
	5	3,209	4,460	5
	14	1,252	1,157	50
	18	1,531	1,369	63
Uniform Piecewise Poly. Approx. (24 bits)	3	127,441	209,121	5
	4	32,042	56,004	8
	5	17,608	30,399	11
	6	9,712	16,489	15
	8	4,965	7,886	24
	9	5,228	8,229	27

* *Not applicable in this case.*

While these early results demonstrate the feasibility of our flow, we note that baseline paths required manual construction due to gaps in existing MLIR hardware lowering support. A detailed and systematic evaluation of baseline strategies, as well as more precise comparisons across numeric formats, will be the subject of future work.

V. Conclusions and Future work

This work presents a multi-level arithmetic-aware MLIR flow that connects high-level mathematical semantics to low-level hardware representations in an end-to-end pipeline.

Preliminary results on a signal processing benchmark show promising trade-offs between memory footprint and arithmetic complexity. However, the current state of end-to-end MLIR hardware support poses challenges for establishing robust baselines. With the dialect and lowering infrastructure now in place, future work will focus on introducing hardware-level optimizations – such as bitheap-based arithmetic synthesis [2, ch. 7] – as well as supporting a wider range of approximation schemes, including table-based methods and non-uniform segmentation strategies. We also plan to extend evaluation to larger workloads in signal processing, linear algebra, and AI, where the benefits of semantic-aware arithmetic compilation are expected to be more pronounced.

References

[1] S. Chevillard, M. Joldeş, and C. Lauter, “Sollya: An environment for the development of numerical codes,” in *International Congress on Mathematical Software*, vol. 6327. Heidelberg, Germany: Springer, September 2010, pp. 28–31.

[2] F. de Dinechin and M. Kumm, *Application-Specific Arithmetic*. Springer, 2024.

[3] C. Lattner *et al.*, “MLIR: Scaling compiler infrastructure for domain specific computation,” in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14.

[4] J.-M. Muller, *Elementary functions, algorithms and implementation, 3rd Edition*. Birkhaüser Boston, 2016.

[5] Y. Orlarey, D. Fober, and S. Letz, “FAUST : an Efficient Functional Approach to DSP Programming,” in *NEW COMPUTATIONAL PARADIGMS FOR COMPUTER MUSIC*, E. D. FRANCE, Ed., 2009, pp. 65–96.

[6] M. Popoff, “Audio DSP to FPGA Compilation: The Syfala Toolchain Approach,” Ph.D. dissertation, Univ Lyon, INSA Lyon, Inria, May 2023.

Towards the evaluation of the Arrowhead SoA in ITS

Luís Ribeiro, Tiago Costa, Ricardo Severino and Luis Lino Ferreira
INESC-TEC/ISEP, Instituto Politécnico do Porto
Porto, Portugal
{lfsro, 1201329, rar, llf}@isep.ipp.pt

Abstract— **The evolution of autonomous driving is reshaping the automotive landscape into a highly cooperative and interconnected system, where vehicles and infrastructure exchange data to improve safety, efficiency, and responsiveness. In this context, service-based architectures are becoming essential to support the modular, scalable, and dynamic nature of automotive applications such as Cooperative Perception, demanding robust mechanisms for real-time communication, service discovery, interoperability, and secure data handling. This work aims at investigating the suitability of the Arrowhead Framework—a service-oriented architecture initially designed for industrial automation—as a middleware to enable and manage services in the context of cooperative autonomous driving. By integrating Arrowhead into a multi-dimensional co-simulation framework, encompassing the simulation of realistic vehicle models, control and communications, we evaluate its effectiveness in supporting service orchestration, system integration, and interoperability in different scenarios. In parallel, we aim to demonstrate how co-simulation environments can facilitate the rapid prototyping and deployment of distributed autonomous driving services.**

Keywords-ITS; service-oriented architecture; co-simulation;

I. Introduction

The rapid advancement of autonomous driving technologies is steering the automotive industry toward a cooperative and intelligent ecosystem. Vehicles are no longer isolated systems but integral nodes in a broader and more complex Intelligent Traffic System (ITS), in which they collaborate with the roadside infrastructure, traffic control systems, and even pedestrians. One of the most promising paradigms in this domain is Cooperative Perception, which relies on the fusion of sensor data shared among various entities to enhance the awareness and safety of automated driving systems. To realize this level of cooperation, future automotive applications must be highly modular, distributed, and interoperable—qualities well-aligned with service architectures. Services allow developers to break down complex functionalities into independently deployable units, enabling scalability, fault isolation, and easier updates. Particularly in ITS, services architectures hold the potential to enable a multitude of services while reusing and repurposing the roadside infrastructure and sensors. Roadside deployed sensors such as cameras or radars can support cooperative highway merging applications, while enabling traffic emergencies detection, and helping authorities enforce traffic rules, by monitoring vehicles' speed. However, deploying services in automotive contexts introduces several challenges, including low-latency requirements, dynamic service discovery, orchestration, data standardization, and secure communications.

In response to these challenges, our work explores the use of the Arrowhead Framework, an open-source service-oriented architecture originally developed for industrial automation, as a service orchestration platform tailored for automotive use cases. We present an integration of Arrowhead into a Co- Simulation Framework, designed to simulate cooperative driving scenarios in its multi-dimensional perspective, encompassing realistic vehicle dynamics, control and communications, to validate the service interactions in a controlled virtual environment. This setup enables the development, testing, and evaluation of automotive-grade services without the high cost and complexity of real-world testing.

The contributions of this work are threefold:

- Assessment of Arrowhead's applicability to cooperative autonomous driving, focusing on its support for multiple automotive service registration, discovery, and secure communication.
- Demonstration of service-based integration for Cooperative Driving applications, leveraging the modularity and extensibility of the Arrowhead architecture.
- Development of a unified co-simulation framework, which allows seamless integration of services across Autonomous Driving and Smart City scenarios, facilitating experimentation and iterative development.

II. Related Work

A. Service-Based Vehicular Networks

Service-based architectures for Vehicular Networks promise to deliver high flexibility, modularity and scalability for designing and implementing ITS applications. This architecture allows the breakdown of specific services into smaller functionalities, which can be further developed and improved on their own, allowing for more efficient, reliable, and fault tolerant applications. Moreover, it facilitates the integration of security, enabling finer-grained access control and monitoring, and reducing the impact of attacks on specific services. Nevertheless, it is fundamental to address the specific cyber-physical requirements such as bounded latency, prevalent in many ITS applications, particularly those involving Advanced Driving Assistance Systems (ADAS) or cooperative driving.

The authors in [1] proposed a service-based architecture that integrates modular V2X services such as Cooperative Awareness with autonomous vehicle systems. The framework developed relies on the Data Distribution Service (DDS) and

Manuscript received July 14, 2025; revised July 31, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.91

introduces a bridge between the V2X services and the autonomous navigation stacks of the vehicles, the Vehicle Programming Interface (VPI). It's capable of dynamic handling of emergency notifications and traffic light states, and its effectiveness was validated by the use of a hybrid experimental setup that combines real world simulation and real-world V2X infrastructure using ITS-G5 communications. This being said, the validation relies on hybrid small-scale tests. Large-scale deployments or edge cases with dense traffic or adverse weather conditions are not addressed. Moreover, security aspects are not addressed.

Another service-based architecture was proposed in [2], promising to handle reliability and latency requirements of 5G and upcoming 6G networks. This solution integrates multiple layers of computing with both fog and mobile edge computing (MEC) and software-defined networking (SDN) that enables adaptive resource management and seamless communication in highly dynamic vehicular environments. Furthermore, the architecture incorporates efficient task offloading, service migration, and handover mechanisms that ensure continuity of service operation under high-speed vehicular mobility. Unlike in our work, the authors used ns-3 for network simulation, using C-V2X. Tests were done with Sumo and ns-3, using the C-V2X as the foundation technology for communications. While the work offers valuable contributes for scaled networks that have a need for the use of handover, task offloading and service migration mechanisms, it also fails to address a few security concerns, such as authentication and encryption and relies heavily on C-V2X/NR and SDN, therefore compatibility with legacy systems such as ITS-G5 is unclear.

Din et al [3] presented a multilayered framework combining services and Named Data Networking (NDN) for efficient in-network computation in autonomous vehicular systems. Their architecture is composed of physical edge servers, and cloud infrastructure to support distributed, computation-intensive tasks. In comparison with more monolithic architecture, through the use of hybrid simulation, the system was capable of enabling efficient offloading of tasks, reducing latency, and optimizing bandwidth usage.

All these works provide a meaningful foundation for the development and improvement of Service-based architectures. While we develop an architecture of our own, we intend to provide an expansible framework that offers the tools to develop these types of architectures.

B. *Arrowhead Framework*

The Arrowhead Framework is a Service-Oriented Architecture (SOA) designed to enable IoT interoperability in-between almost any IoT elements [4]. The framework provides a well-defined structure for managing loosely coupled, event-driven automation systems through core services such as Service Registry, Authorization, and Orchestration [5]. These services enable dynamic discovery, secure access control, and more efficient coordination of distributed systems, making them suitable for industrial automation, IoT, and smart infrastructures. The framework also supports local cloud deployments and inter-cloud collaboration for broader scenarios, which is important to ensure the low-latency and scalability requirements needed for autonomous driving applications. Arrowhead has been applied to a few related scenarios with success, however, none with the stringent requirements imposed by ADAS services. For instance, Joniken et al. [6] demonstrated the capabilities of the use of the Arrowhead Framework for smart city service integration, implementing and connecting two distinct urban infrastructure systems (a street lighting system and a car engine block heating system) into a unified, collaborative automation environment. In this work each system was wrapped with an Arrowhead-compliant interface, exposing RESTful services for monitoring and control, and integrated using Arrowhead's core systems. The control system, acting as a consumer, was capable of dynamically discovering and orchestrating services based on environmental sensor data, such as temperature and luminosity. Passerone et al. [7] presented a comprehensive design methodology for secure and safety-compliant communication in autonomous vehicle systems, specifically those who rely on V2V communication. They introduced a contract-based design approach to formalize and verify system requirements and used the Arrowhead Framework to manage secure service-oriented communications. Verification of system behaviors against formalized safety assertions is done through Contract Analysis Tool (CAT) and 3D simulations in Blender, and it was finally validated through a prototype implementation involving real-time control of wheeled robot platoons. Arrowhead's token-based authorization was able to enable secure communication with latencies of up to 40ms, which addresses the trade-off between security and overhead.

III. Architecture

In this section, we describe the architecture of the Co-Simulation setup and its integration with the Arrowhead Framework. Moreover, we will delve into the case scenarios that will be developed for testing Vehicular Networks.

A. *System's Architecture*

Figure 1 displays a simplified architecture of our Co-Simulation Framework and its integration with Arrowhead. It integrates both ROS 2 and Gazebo with the Omnet++ and Arrowhead Frameworks. Omnet++ is a network simulation tool, and we use it together with the libraries from the Artery framework which allows us to simulate Vehicular Communications with protocols such as ITS-G5.

Our Framework leverages ROS2 and the underlying DDS as the foundation for the interaction between the different tools that compose our application. The Omnet++ Interface is composed by a Omnet++ transmitter that creates a publisher, whose topics are subscribed by the Omnet++ module, which then relays the information to the correct nodes within the Omnet++ simulation environment. The interface also has an Omnet++ receiver which captures the packets from the Omnet++ simulation environment and publishes them into the respective ROS topics.

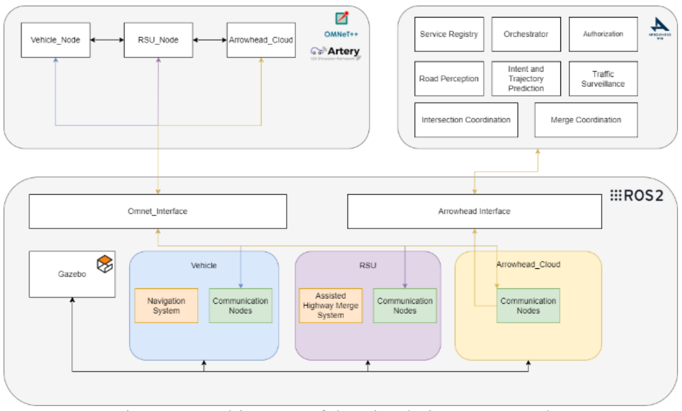


Figure 1: Architecture of the Simulation Framework.

In the context of simulation for connected vehicles, integrating real-time communications with simulation time can be quite challenging due to the Event-Driven nature of network simulation tools, while Simulation tools like Gazebo operate on real-time progression. To ensure that our Network Simulation tool, in this case Omnet++, is on par with real-time events, it's necessary to build a scheduler capable of synchronizing event times with the simulation times. To ensure both Omnet++ and Gazebo are on par with each other, we are using an event scheduler previously made for Omnet++ in [8].

The Arrowhead Interface is a module that sends and handles RESTful requests to the Arrowhead Framework from the nodes within our simulation environments. Within our network, Arrowhead allows for service discovery and authentication. As already mentioned, we intend to test the capabilities of Arrowhead, evaluating the impact services delays may have on the performance of cooperative driving tasks. In our network architecture we decompose services into smaller services capable of interoperating together in many scenarios. In this case, we have Arrowhead Core Services, Service Registry, Orchestrator and Authorization along with some services that are going to be provided by Roadside Units (RSU). The following services are deployed at Edges and discovered and orchestrated by Arrowhead's core services:

- **Road Perception –** Detects and classifies static and dynamic road agents, such as vehicles and pedestrians.
- **Intent and Trajectory Prediction –** Predicts the behavior and trajectory of nearby road agents using past motion and context information.
- **Traffic Surveillance –** Monitors the road and detects any unsafe or irregular behaviors or events, such as stalled vehicles or illegal maneuvers.
- **Merge Coordination** – Dedicated to the highway merge scenario, its function is to coordinate highway merging by calculating and suggesting safe gaps and speeds for entering vehicles.
- **Intersection Coordination –** Dedicated to the urban intersection scenario, its function is to coordinate and assign priorities to vehicles approaching and traversing intersections.

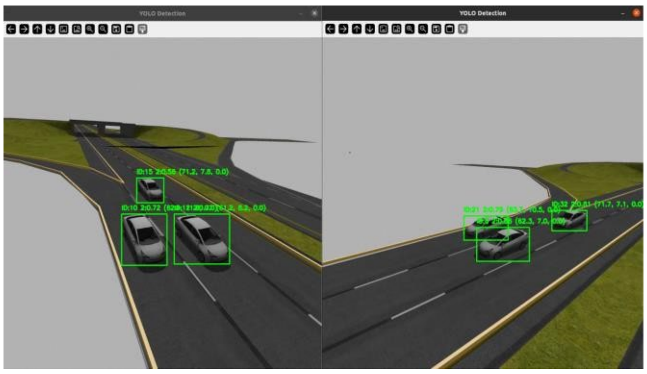


Figure 2: Vehicle Detection in Gazebo.

The Road Perception service relies on cameras that are distributed across the road positioned in a Bird's Eye View (BEV) configuration within the Gazebo simulation environment. YOLOv8 [9] was used for object detection and tracking, and trilateration was applied to fuse the different cameras' perspectives to provide more precise estimations of the positions of the road agents as shown in Figure 2.

B. Test Scenarios

To validate the proposed solution and assess the applicability of Arrowhead Framework within the context of cooperative autonomous driving, we design and implement two different test scenarios, High-Speed Highway Merge and Intersection Assistance. The interaction between vehicle and infrastructure requires the vehicle to first communicate with the RSU's requesting which services are available to use in that region. The RSU connected to an Edge forwards this request to an Arrowhead node deployed at a considerable distance, which will reply with the list of services available in that region. After receiving the list, the vehicle can finally choose what services it wants to subscribe to, through the same process, where then the Arrowhead will verify the vehicle's authenticity to consume those services.

The Highway Merge scenario serves as the most demanding test case, since it imposes lower latencies upon the performance of our services. In this setup, multiple vehicles will be deployed randomly, at speeds up to 120 km/h, simulating real world traffic. The cameras detect and track the vehicles' positions and speed, which will be used to inform incoming vehicles wanting to merge onto the highway. The Highway Merge service identifies suitable gaps and assigns them to incoming vehicles along with the recommended speeds for merging. Our primary goal is to evaluate the latency bounds and responsiveness of the Arrowhead Framework at higher speed cooperative scenarios. Through pushing the framework's capabilities, we can identify limitations and explore possible enhancements that could make Arrowhead more appropriate for automotive applications where low latency is a critical aspect.

The Urban Intersection Assistance scenario, while less sensitive to higher latencies than our previous scenario, offers a controlled environment that can be used to test communications under mixed traffic conditions and serve as a test bed for service scalability, orchestration and integration with other types of services, specifically Smart City applications. Furthermore, it provides an opportunity to explore intersections as convergence points between cooperative driving and urban infrastructure services, namely traffic monitoring, pedestrian safety, and emergency management. Figure 3 presents a more general and hierarchical view of the deployment scenarios. The vehicles, sensors, and RSU's are distributed throughout the environment. The RSUs are directly connected to the Edge nodes, providing them with sensorial data required for service execution. The Edges communicate with Arrowhead nodes that are deployed further from the urban environment usually in the form of Fog's deployed in the city where these services are provided.

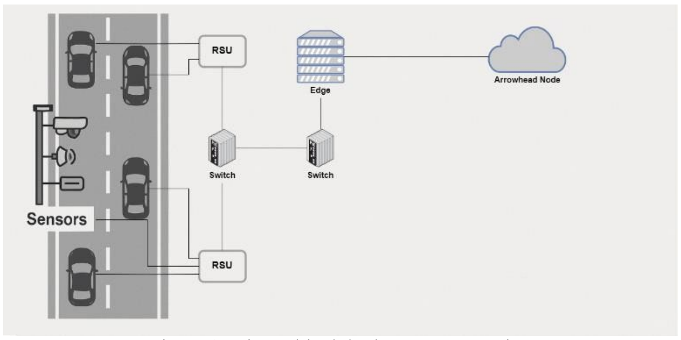


Figure 3: Hierarchical deployment Scenario.

IV. CONCLUSION AND FUTURE WORK

This work aims at assessing the performance of the Arrowhead Framework as a middleware in enabling and managing services in the context of cooperative autonomous driving. To this end, we integrated the Arrowhead Framework over a co-simulation framework relying on ROS 2, Gazebo, and OMNeT++ and devised a set of cooperative autonomous driving scenarios. By leveraging Arrowhead's service-oriented architecture, we enable dynamic service discovery, orchestration, and secure communication between distributed vehicular and infrastructure nodes. Our initial implementation focuses on validating the feasibility and responsiveness of Arrowhead microservices in two representative scenarios: high-speed highway merging and urban intersection coordination. These scenarios allow us to examine the framework's behavior under both strict latency constraints and complex, scalable service environments. Early results suggest that while Arrowhead offers promising mechanisms for structuring and managing ITS services, its performance in real-time vehicular contexts—especially under high-speed conditions—must be thoroughly assessed. The proposed co-simulation setup provides a flexible and extensible environment to iteratively develop, test, and refine distributed services for automotive and smart city applications. It serves as a valuable platform to simulate real-world conditions encompassing multiple cyber-physical dimensions, while maintaining controlled test parameters for the intended evaluation.

Ongoing and future work will focus on profiling service delays, optimizing orchestration logic, and exploring enhancements to the Arrowhead Framework to better meet the stringent demands of vehicular networks and general QoS demanding ITS services. Ultimately, this research contributes to the broader goal of enabling safer and more efficient cooperative autonomous systems by evaluating service-oriented solutions within realistic, simulation-driven development pipelines.

ACKNOWLEDGMENT

This research is supported by the OPEVA project funded within the Chips Joint Undertaking (Chips JU) from the European Union's Horizon Europe Programme and the National Authorities (France, Czechia, Italy, Portugal, Turkey, Switzerland), under grant agreement 101097267. It is also funded by the EU Chips-JU project Arrowhead fPVN with grant agreement no. 101111977 and the national funding authorities from the countries involved. Views and opinions expressed are however those of the authors only and do not reflect those of the European Union or Chips JU.

REFERENCES

[1] J. Amaral, A. Figueiredo, P. Rito and S. Sargento, "Microservice-based Approach to Integrate V2X with Autonomous Mobility," 2024 IEEE Future Networks World Forum (FNWF), Dubai, United Arab Emirates, 2024, pp. 221-225, doi: 10.1109/FNWF63303.2024.11028714.

[2] Zarie, Mira M., Abdelhamied A. Ateya, Mohammed S. Sayed, Mohammed ElAffendi, and Mohammad Mahmoud Abdellatif. 2024. "Microservice-Based Vehicular Network for Seamless and Ultra-Reliable Communications of Connected Vehicles" Future Internet 16, no. 7: 257. https://doi.org/10.3390/fi16070257

[3] Muhammad Salah Ud Din, Muhammad Atif Ur Rehman, Muhammad Imran, Byung Seo Kim, Evaluating the impact of microservice-centric computations in internet of vehicles, Journal of Systems Architecture, Volume 150, 2024, 103119, ISSN 1383-7621, https://doi.org/10.1016/j.sysarc.2024.103119.

[4] Eclipse Foundation, "The Arrowhead Framework," https://projects.eclipse.org/projects/iot.arrowhead.

[5] Varga, Pal & Blomstedt, Fredrik & Ferreira, Luis & Eliasson, Jens & Johansson, Mats & Delsing, Jerker & Martínez de Soria, Iker. (2016). Making system of systems interoperable – The core components of the arrowhead framework. Journal of Network and Computer Applications. 81. 10.1016/j.jnca.2016.08.028.

[6] J. Jokinen, T. Latvala and J. L. Martinez Lastra, "Integrating smart city services using Arrowhead framework," IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, Italy, 2016, pp. 5568-5573, doi: 10.1109/IECON.2016.7793708.

[7] Passerone, Roberto & Cancila, Daniela & Albano, Michele & Mouelhi, Sebti & Plósz, Sándor & Jantunen, Erkki & Ryabokon, Anna & Emine, Laarouchi & Hegedus, Csaba & Varga, Pal. (2019). A Methodology for the Design of Safety-Compliant and Secure Communication of Autonomous Vehicles. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2937453.

[8] Teper, Harun & Bayuwindra, Anggera & Riebl, Raphael & Severino, Ricardo & Chen, Jian-Jia & Chen, Kuan-Hsun. (2022). AuNa: Modularly Integrated Simulation Framework for Cooperative Autonomous Navigation. 10.48550/arXiv.2207.05544.

[9] Yaseen, Muhammad. (2024). What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector. 10.48550/arXiv.2408.15857

Human Activity Recognition Using SVM-based on micro-Doppler Radar Data Classification

Claire Béranger[1], Alexandre Bordat[1,2], Petr Dobias[1], Ngoc-Son Vu[1],
Julien Le Kernec[1,3], David Guyard[2], Olivier Romain[1,3]
[1]*ETIS Laboratory UMR 8051, CY Cergy Paris Université, ENSEA, CNRS, F-95000 Cergy, France*
[2]*BlueLinea, 12 Parvis Colonel Arnaud Beltrame, 78000 Versailles, France*
[3]*James Watt School of Engineering, University of Glasgow, Glasgow, United Kingdom*
smartgaitlab@etis-lab.fr

Abstract—**In response to the challenges posed by an ageing population, radar-based fall detection is gaining attention as a valuable tool for clinical monitoring and teleassistance. Once the radar signals are processed, they can be visualised as spectrograms that capture the dynamic signatures of human activity. In this work, we propose an approach that leverages image processing techniques to extract descriptive features, such as *Area*, *Perimeter* or *Orientation* from these activity signatures. These features are then fed into a Support Vector Machine (SVM), a lightweight yet effective classification model. Our method achieves an accuracy of 88.85%, providing a resource-efficient alternative that matches or exceeds more complex state-of-the-art solutions.**

Keywords-Radar Spectrogram, Human Activity Recognition, Classification, SVM, Clinical Context

I. Introduction

Many developed countries face an ageing population, increasing the risk of motor impairments and falls. In France, 20% of the population is at risk, leading to 10,000 deaths and more than 136,000 hospital admissions each year [1]. In 2022, global recommendations were established for elderly fall prevention [2]. The standard approach involves physician referrals for clinical gait and balance assessments. However, overcrowded services often lack sufficient staff and time.

Wearable sensors, such as gyroscopes and accelerometers in necklaces or watches, can be used to recognise activities like "Walk" [3], but they are restrictive. Cameras offer alternatives, yet remain intrusive as well [4]. Radar provides a non-intrusive way to detect micro-movements without visual imaging. The resulting micro-Doppler spectrograms reflect activity-specific limb movements [5]. Classification models like Support Vector Machine (SVM), K-Nearest Neighbors (KNN) and GoogLeNet achieved recognition accuracies from 74% to 94%.

The study [6] achieved 87.10% accuracy with ResNet-18, without modifying the model or input data. This suggests potential for improvement using basic image processing and a lighter model. This paper proposes a lightweight classifier for human activity recognition based on spectrogram features, and explores simple image processing techniques.

Section II reviews radar pre-processing, classification methods, and image processing techniques used. Then, Section III presents the spectrogram generation, feature extraction, and classification applied. Section IV describes and compares results with [6], while Section V offers an overall analysis.

Manuscript received May 20, 2025; revised July 23, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.92

II. Related Work

In this section, we present the radar data extraction method and the recognition techniques developed from it.

A. *Radar Data Extraction*

Radar (Radio Detection And Ranging) uses radio waves to detect and track objects. For activity recognition, its key advantage is analysing Doppler signatures [6]. Doppler signatures reflect frequency changes due to movement, while micro-Doppler are small variations caused by finer movements, such as arm motion during walking.

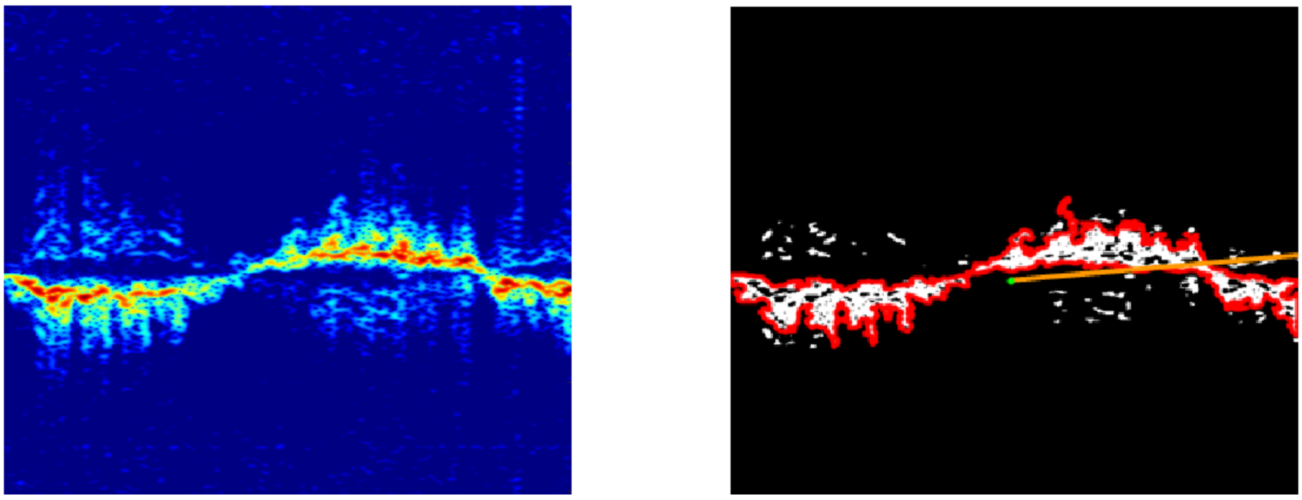
The radar emits signals and measures reflections. Most studies use Frequency Modulated Continuous Wave (FMCW) radars [7]. Analysis of the received signal provides Doppler frequency and time delay (beat frequency). Raw data undergoes Fast Fourier Transform (FFT) to extract distance and time, followed by filtering to remove static elements. Then, a Short Time Fourier Transform (STFT) generates velocity-time representations, highlighting Doppler variations.

The pre-processed data are spectrograms, visually representing movement speed over time. As shown in Figure 1a, spectrograms display signal energy distribution during an activity. These representations serve as the basis for recognition algorithms, which distinguish activities by their unique signatures.

For FMCW radar, range resolution depends on bandwidth, while Doppler resolution depends on observed signal duration in the STFT. In this study, we use a range resolution of 37.5 cm and Doppler resolution of 1.25 Hz (0.03 m/s), enabling accurate velocity analysis and classification. Using these parameters, spectrograms like Figure 1a are generated.

B. *Activity Classification*

Current research aims to improve image classification using radar data from animals or humans [9], [10]. Common models like SVM, KNN and ResNet-18 [6], [11], [12] require sufficient data for training. Experiments on the *Radar Signatures of Human Activities* dataset [8] show spectrograms are effective for recognition. Studies [11], [12] used AlexNet for feature extraction and transfer learning to SVM and KNN, achieving accuracies of 78.25% and 77.15%. Transfer learning uses knowledge from pre-trained models to solve related tasks.



(a) Spectrogram of "Walk" [8]

(b) Parameters Extraction (*Area*, *Perimeter*, *Orientation*, *Centroid*)

FIGURE I. Spectrogram Analysis, from the Original to the Parameters Extraction

ResNet-18, an 18-layer network, achieves 70% to 90% accuracy without optimisation, and suits resource-limited systems [6]. The study [6] reported 87.10% accuracy with unmodified ResNet-18. While deep learning models perform well, SVMs remain a viable lightweight alternative for embedded applications due to fast training and low resource needs.

SVMs are effective in human activity recognition by constructing optimal hyperplanes between classes. The study [13] showed SVMs can reach 92.8% accuracy on Continuous Wave (CW) radar and 95.4% on FMCW, though performance varies with pre-processing. Based solely on micro-Doppler data, the accuracy drops to around 80%. Kernel choice, hyperparameters, and pre-processing are critical. SVMs remain a strong option for balancing performance with hardware constraints, especially when combined with feature extraction.

C. *Image Processing, Parameters Extraction & PCA*

Studies [12], [13] on human activity recognition from spectrograms aim to improve performance by processing data. Some approaches [14] adjust parameters during data acquisition and pre-processing, like filters or biases. Others [15] modify the data before classification, with common classifiers including Convolutional Neural Network (CNN) or SVM.

Most datasets offer opportunities to enhance accuracy with different processing strategies. These datasets address human activity recognition in diverse contexts, based on spectrograms from radar or simulated sensors. The *Radar signatures of human activities* dataset [8], widely used with over 5,000 downloads, has been extensively tested in pre-processing, spectrogram manipulation, and classification. Techniques like binarisation and masks help extract relevant activity features.

Classical processing methods bring only slight improvements. Studies [16] show that Principal Component Analysis (PCA) effectively reduces dimensionality by selecting key spectrogram variables. This enhances model accuracy and lowers computational cost.

III. METHODOLOGY

This section describes the input data generation method and the classification model employed to improve accuracy.

A. *Spectrogram Generation & Parameter Extraction*

The *Radar signatures of human activities* [8] dataset, chosen for its completeness and relevance, contains 1,753 images across 6 activity classes. We follow the pre-processing chain of authors. After generating spectrograms, activity signatures are extracted by binarising the images, unlike [6], which used raw spectrograms. To reduce storage and computational costs, classical shape-based features are preferred.

We extract 8 shape features, *Area*, *Perimeter*, *Orientation*, *Major*, *Minor*, *Centroids X and Y*, and *Excentricity*, using mathematical methods and the *skimage*[1] Python library. Activity signatures are isolated by identifying connected regions in the binarised spectrograms. Morphological and geometric descriptors are computed via the *regionprops* function, while contours are extracted using *measure.find_contours* with a 0.9 threshold and *fully_connected = " high "* option enabled to have full diagonal connectivity.

$$\theta = \begin{cases} \theta + 90, & \text{if } \theta < 0 \\ \theta - 90, & \text{else} \end{cases} \qquad (1)$$

Contours oriented clockwise are reversed to counterclockwise to maintain geometric consistency. Orientation θ is converted to degrees and adjusted for the image coordinate system, where y increases downward, as described in Equation 1.

B. *Hyperparameter & Parameter Selection*

The *Radar signatures of human activities* [8] dataset contains two similar activities: "Pick" and "Drink". The paper [6] showed this confusion. To address this issue, we propose combining these activities under a single label, "Other". As a

[1] https://scikit-image.org/

result, we work with 5 activity classes: "Walk", "Sit", "Stand", "Other", "Fall".

The optimal SVM kernel was chosen based on the five highest classification accuracies from spectrogram parameters. Among common kernels (Linear, RBF, Sigmoid, Polynomial), Linear and Sigmoid were excluded due to the nonlinearity data. RBF and Polynomial kernels, which performed well in initial tests, were compared to identify the most suitable kernel and hyperparameters for human activity recognition.

For the Polynomial kernel, key hyperparameters include *C* (error penalty), *degree* (model complexity), and *coef0* (importance of lower *degree* terms, especially when *degree* > 1). Hyperparameter tuning was done using *GridSearchCV* (exhaustive search with cross-validation) and *Randomized-SearchCV* (random sampling within defined bounds), both ensuring model robustness and generalisation.

Finally, we apply PCA to reduce data dimensionality, capture key variations, and compare results with and without this technique. The goal is to optimise model parameters and hyperparameters by selecting the most relevant features from spectrogram signatures.

C. Classification Method

The study [6] has highlighted the feasibility of recognising activities using a simple and easily implementable recognition method. ResNet-18 performed initial training directly on the spectrograms obtained at the output of a radar recording. However, we are now exploring an alternative approach for interpreting the signatures, while still aiming to maintain a simple and lightweight solution.

An SVM is used to learn from spectrograms after feature extraction. It offers a lightweight, fast, and effective solution, especially for small datasets, and requires minimal storage. The *SVM implementation* from the Python library *sklearn*[2] is used, with the kernel type selected based on extracted data results. Hyperparameter tuning, detailed in Section III-B, identifies optimal settings.

Training is conducted on the *Radar signatures of human activities* dataset [8], using the same train and test splits as [6] for fair comparison with the ResNet-18 model. Despite the imbalance of the dataset, particularly fewer samples for the "Fall" activity, this does not hinder performance evaluation, as noted in [6]. The model is trained on features extracted from spectrograms stored in text files. From eight initial features, the most relevant ones are selected, as described in Section III-B, along with suitable kernels and hyperparameters.

IV. RESULTS & DISCUSSION

This section outlines the results, from parameter extraction to model training.

A. Hyperparameter Selection

The results presented here focus on the Polynomial kernel. To identify optimal hyperparameters, we applied two search techniques: *GridSearchCV*, which exhaustively explores a predefined set, and *RandomizedSearchCV*, which samples a fixed number of combinations randomly. We tuned the hyperparameters *C*, *degree*, and *coef0*. Both methods agreed on *coef0* = 1.0, but highlighted two promising values for *C* (100, 1000) and *degree* (3, 4).

TABLE I. ACCURACY FOR ALL HYPERPARAMETERS COMBINATIONS

	C=100	**C=1000**
degree=3	87.45%	87.45%
degree=4	87.80%	86.41%

All four combinations were tested, revealing up to a 10% accuracy difference between *degree* = 3 with *C* = 1000 and *degree* = 4 with *C* = 100. The highest accuracy was achieved with *degree* = 4 and *C* = 100, as shown in Table I. The next step involves extracting and selecting the most relevant features from the spectrogram data.

B. Parameter Extraction and Selection

To extract the characteristic data, the spectrogram image is first binarised as shown in Figure 1b, followed by contour extraction to recover the activity signature. This enables the computation of eight previously described parameters, including *Area*, *Perimeter*, and *Orientation*. In Figure 1b, the *Area* is outlined in red, the *Orientation* indicated by the orange line, and the *Centroid* marked by the green dot.

These parameters serve as SVM inputs, but their relevance must be assessed to determine whether the full set is necessary. PCA is then applied: the data are centred and standardised, and the explained variance of each principal component is calculated. As shown in Figure 2, at least 5 components are needed to preserve 90% of the total variance, indicating robust data representation.

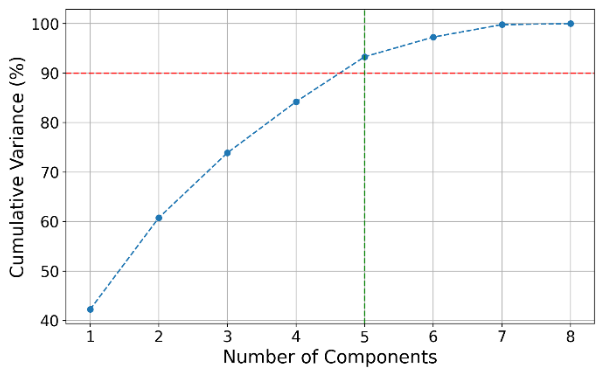


FIGURE II. Explained Variance per PCA

Subsequently, all parameter combinations were evaluated. Subsets of 5 to 6 features proved sufficient to achieve recognition rates of approximately 80% or higher. Among the top-performing subsets, the most consistently relevant features, identified using the polynomial kernel with tuned hyperparameters, are *Orientation*, *Major*, *Minor*, *Centroid X*, *Centroid Y*, and *Excentricity*.

C. SVM Application and Its Advantage over ResNet-18

Using the selected feature set, the SVM with a Polynomial kernel achieved an accuracy of 88.85%. This configuration

[2] https://scikit-learn.org/stable/

included the features *Orientation*, *Major*, *Minor*, *Centroid X*, *Centroid Y*, and *Excentricity*. The confusion matrix in Figure 3 confirms strong classification performance, with clear diagonals and minimal confusion between classes.

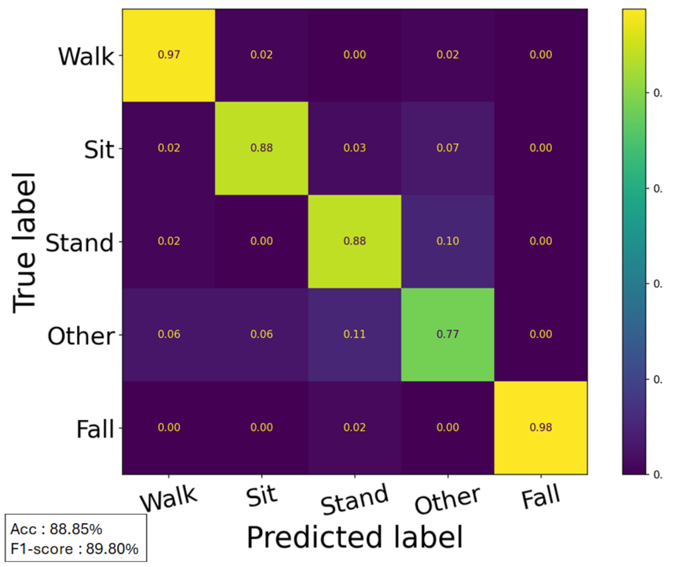


FIGURE III. SVM Normalised Confusion Matrix

The use of SVM improved activity recognition performance by 1.75% compared to the ResNet-18 of [6]. Grouping similar activities helped reduce confusion, while learning from image-extracted parameters led to further gains, exceeding a 1% improvement. These results emphasise the value of clearly defined spectrogram signatures, as more distinctive features allow for more accurate classification. In [6], an accuracy of 87.10% was achieved using dataset [8]. Our approach demonstrates that a lightweight, resource-efficient method focused on the most relevant features can still deliver strong performance.

V. CONCLUSION

This paper proposes a method for recognising human activity using Frequency Modulated Continuous Wave (FMCW) radar data. Spectrograms are generated through a preprocessing chain from an open-source dataset [8]. Key features such as *Area*, *Perimeter*, and *Orientation* are extracted from these spectrograms and used to train a Support Vector Machine (SVM). Using Principal Component Analysis (PCA), the approach achieved an accuracy of up to 88.85%, demonstrating that complex radar data can be effectively analysed with simple, informative features.

Future work will focus on exploring lightweight, efficient activity recognition methods and enhancing existing techniques. Once detection and recognition reach satisfactory performance, the next step will involve developing and evaluating a real-time solution for fall risk prediction. The long-term objective is to create a real-time embedded system suitable for real-world deployment.

REFERENCES

[1] T. Marion et al., Chutes des personnes agees a domicile. Caracteristiques des chuteurs et des circonstances de la chute. Volet ≪ Hospitalisation ≫ de l' enquete ChuPADom, 2018. Etudes et enquetes,2020. https://www.santepubliquefrance.fr/maladies-et-traumatismes/traumatismes/chute/documents/enquetes-etudes/chutes-des-personnesagees-a-domicile.-caracteristiques-des-chuteurs-et-des-circonstancesde-la-chute.-volet-hospitalisation-de-l-enquete-chupadom

[2] H. Blain et al., "Anti-fall plan for the elderly in France 2022-2024: objectives and methodology," Geriatrie et Psychologie Neuropsychiatrie du Vieillissement, vol. 21, no. 3, pp.286-294,Sep.2023.http://www.john-libbey-eurotext.fr/medline.md?doi=10.1684/pnv.2023.1122

[3] S. Iloga, et al., "Human Activity Recognition Based on Acceleration Data From Smartphones Using HMMs," IEEE Access, vol. 9, pp. 139 336-139 351, 2021. https://ieeexplore.ieee.org/document/9557268/

[4] A. Bordat, P. Dobias et al., "Towards Real-Time Implementation for the Pre-Processing of Radar-Based Human Activity Recognition," in 2022 IEEE 31st International Symposium on Industrial Electronics (ISIE). Anchorage, AK, USA: IEEE, Jun. 2022, pp. 635-638. https://ieeexplore.ieee.org/document/9831677/

[5] S. Vishwakarma, "Learning Algorithms For Micro-Doppler Radar Based Detection, Classification and Imaging of Humans in Indoor Environments," Ph.D. dissertation, Indraprastha Institute of Information Technology,Delhi,2020.https://repository.iiitd.edu.in/xmlui/handle/123456789/800

[6] C. Béranger et al., "Radar-based human activity acquisition, classification and recognition towards elderly fall prediction," in 2023 26th Euromicro Conference on Digital System Design (DSD). Golem, Albania: IEEE, Sep. 2023, pp. 95-102. https://ieeexplore.ieee.org/document/10456803/

[7] S. Hu et al., "Radar-Based Fall Detection: A Survey [Survey]," IEEE Robotics & Automation Magazine, vol. 31, no. 3,pp.170-185,Sep.2024. https://ieeexplore.ieee.org/document/10420485/

[8] D. F. Fioranelli et al., "Radar sensing for healthcare: Associate Editor Francesco Fioranelli on the applications of radar in monitoring vital signs and recognising human activity patterns," Electronics Letters, vol. 55, no. 19, pp. 1022-1024, Sep. 2019. https://researchdata.gla.ac.uk/848/

[9] A. Khawani, "Human/Animal Activity Recognition Data Analysis – Classification and Identification," 2024. https://essay.utwente.nl/100968/1/Khawani_BA_EEMCS.pdf

[10] D. Park et al., "Radar-Spectrogram-Based UAV Classification Using Convolutional Neural Networks," Sensors, vol. 21, no. 1, p. 210, Dec. 2020. https://www.mdpi.com/1424-8220/21/1/210

[11] S. A. Shah et al., "Human Activity Recognition : Preliminary Results for Dataset Portability using FMCW Radar," in 2019 International Radar Conference (RADAR). TOULON, France: IEEE, Sep. 2019, pp. 1-4. https://ieeexplore.ieee.org/document/9079098/

[12] W. Taylor et al., "Radar Sensing for Activity Classification in Elderly People Exploiting Micro-Doppler Signatures Using Machine Learning," Sensors, vol. 21, no. 11, p. 3881, Jun. 2021. https://www.mdpi.com/1424-8220/21/11/3881

[13] Z. Li, "Radar sensing for Ambient Assisted Living application with Artificial Intelligence," Ph.D. dissertation, James Watt School of Engineering, University of Glasgow, Glasgow, United Kingdom, 2024. https://theses.gla.ac.uk/84245/1/2024LiZhenghuiPhD.pdf

[14] N. B. Nguyen et al., "Enhance micro-Doppler signatures-based human activity classification accuracy of FMCW radar using the threshold method," Journal of Military Science and Technology, vol. 95, no. 95, pp. 20-28, May 2024. https://en.jmst.info/index.php/jmst/article/view/1142

[15] Youngwook Kim et al., "Human Activity Classification Based on Micro-Doppler Signatures Using a Support Vector Machine," IEEE Transactions on Geoscience and Remote Sensing, vol. 47, no. 5, pp. 1328-1337, May 2009. http://ieeexplore.ieee.org/document/4801689/

[16] Y. Zhao et al., "Human Activity Recognition Based on Non-Contact Radar Data and Improved PCA Method," Applied Sciences, vol. 12, no. 14, p. 7124, Jul. 2022. https://www.mdpi.com/2076-3417/12/14/

Multi Hardware-Attack Dataset and ML-based Detection Using Processor Stress Patterns on x86

1st David Andreu
Department of Computer Architecture
Universitat Politècnica of Catalunya
Barcelona, Spain

david.andreu.gerique@estudiantat.upc.edu

2nd Beatriz Otero
Department of Computer Architecture
Universitat Politècnica of Catalunya
Barcelona, Spain

beatriz.otero@upc.edu,

3rd Ramon Canal
Department of Computer Architecture
Universitat Politècnica of Catalunya
Barcelona, Spain

ramon.canal@upc.edu,

***Abstract*—Hardware attacks exploit the vulnerabilities discovered in state-of-the-art CPUs. As an example, attacks such as Meltdown and Spectre have made the headlines. To benefit from the vulnerabilities, hardware attacks stress tremendously some section/s of the processor, usually the branch-prediction unit and the different cache levels. This gives us a recognizable pattern and a way to implement a system capable of detecting the presence of these attacks while monitoring the computer. In this paper, we describe the set of hardware attacks under focus, then we describe how we create the dataset and, finally, the use of machine learning to detect the attacks in three scenarios (i.e. training on both benign applications and attacks, training on only benign applications and training only on attacks) and two x86 CPUs (Intel and AMD). The techniques proposed are capable of achieving over 99% detection rate with a machine learning model. This provides a run-time solution to quickly identify the attack as it starts running and take remedial actions.**

***Index Terms*—Security, hardware attack, Spectre, Meltdown, Fallout, machine learning**

I. Introduction

In today's world, computers are integral to our daily lives, from work desktops to personal smartphones. We trust these devices to securely store our data, but this trust is often misplaced, as we are continually at risk of cyber attacks. Most modern attacks exploit vulnerabilities in operating systems, with *privilege escalation* being the most common goal. These software-based vulnerabilities can often be quickly fixed through updates. In contrast, *hardware-based attacks*, like Meltdown [1] and Spectre [2], target vulnerabilities in the microprocessor itself. These hardware attacks typically stress parts of the processor, such as the branch-prediction unit and caches, creating recognizable patterns that can uncover them. Modern solutions such as KPTI [3] are effective in mitigating many hardware attacks. However, the objective is to develop a solution that can also address future attacks. By employing machine learning's pattern recognition capabilities, similar behaviours in new exploits can be identified. The objective of this work is to develop, train, and fine-tune a machine learning (ML) model capable of detecting both current and future hardware attacks by learning their characteristic patterns.

Specifically, in this work, we will refer to hardware attacks as a set of attacks falling under the category of hardware-based attacks known as cache side-channel attacks. Cache side-channel attacks are a type of attack that exploit unintentional information leaks within the processor's cache system. These attacks use variations in cache access times, storage patterns, or eviction behaviours to infer sensitive information such as cryptographic keys or private user data. These attacks leverage the subtle changes in how data is stored, accessed, or evicted across different cache levels, enabling attackers to deduce the operations performed by a program without directly accessing the target data. This poses a significant and sophisticated security threat to modern computing systems.

There are various types of attacks within this category. In this work, we focus on the following cache side-channel attacks: Spectre V1, Spectre V2, Spectre V4, Meltdown, ZombieLoad, Fallout and Crosstalk.

II. Hardware attacks in x86

A. Threat model

This work assumes an *unprivileged attacker* (i.e., without kernel-level access) and the *absence of kernel-level mitigations* against microarchitectural vulnerabilities such as Kernel Address Space Layout Randomization (KASLR) (e.g. KAISER [3] LAZARUS [4], or FLARE [5]) —which are designed for x86 architectures and rely on platform-specific features —are not considered active. This assumption is realistic as we are considering a platform-independent mechanism to detect the attacks.

B. Related Work

Hardware attacks create distinctive performance anomalies, such as an unusual frequency of branch mispredictions or excessive cache evictions. These patterns are often consistent across different executions of the attack, enabling ML models to generalize and detect them reliably. Previous approaches to detecting hardware attacks using hardware performance counters and ML models have shown great performance [6], [7], [8], [9]. However, these efforts focus only on a single attack and do not share their training datasets, models, or instructions for generating similar models locally. This lack of extensive analysis and reproducibility in the area has brought some authors [10] to believe that ML is not appropriate, even disregarding previous work and evidence.

Other studies, like Carnà et al. [7], provide examples of binaries used in attacks, they lack details on data recording

Manuscript received May 19, 2025; revised July 30, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Regular
DOI: doi.org/10.64552/wipiec.v11i1.94

conditions and methods, as well as specifics on benign programs, making exact replication difficult.

The primary goal of this work is to build and release [11] a comprehensive dataset comprising multiple known attacks and describing the creation methodology. With this, we develop a novel ML model for detecting hardware attacks using hardware performance counters (HPCs). This effort aims to facilitate replication and foster further research within the community.

The ML model is designed to enable rapid, real-time analysis of HPC data streams, making it possible to scale the detection mechanism across multiple systems without sacrificing responsiveness or reliability. In summary, our work makes the following contributions:

- **Build a reliable and reproducible dataset**. The dataset must include relevant samples from both hardware attacks and benign programs, correctly labeled and compatible across different machines and architectures. It should be reproducible under the same conditions on other machines, involving:
  - Identify hardware attack binaries and benign programs for data collection.
  - Record HPC data and sample rate.
- **Develop a ML model**. The model should classify input samples as either malicious or benign, and also distinguish between known attacks and benign programs. The ML model will be broken down in the following steps:
  - Identify the optimal ML model for the task.
  - Preprocess data for the selected model and training it.
  - Optimize parameters.

III. ML Models

Machine learning is preferred over deep learning because the dataset is too small. Deep learning models require hundreds of thousands to millions of samples, while the current dataset only has 28,000 samples from 14 different programs. This size is insufficient for deep learning, making traditional ML methods more suitable and recommended for smaller datasets.

This paper uses Naive Bayes, Decision Tree, Random Forest, and Support Vector Machines classifiers for multi-class classification tasks. Each method is briefly described below.

Naive Bayes is a simple and efficient probabilistic classifier based on Bayes' theorem, assuming feature independence [12].

Decision trees are widely used supervised learning algorithms for classification [13]. They have a hierarchical tree-like structure, where the internal nodes represent decisions based on the feature values, the branches represent the decision results, and the terminal nodes represent the classification categories.

Random Forest (RF) is a classification algorithm that builds multiple independent decision trees using bootstrap sampling of the training data [14]. For each split, it randomly selects a subset of features. In classification, each tree votes on the class, and the majority vote determines the final prediction.

Support Vector Machine (SVM) is a supervised learning algorithm for classification tasks, such as distinguishing between benign and malicious samples and identifying specific attacks [15]. SVM uses support vectors, the critical data points closest to the decision boundary (hyperplane), to maximize the margin between classes, improving generalization to new data. The margin can be determined using linear or non-linear functions like polynomial or radial basis functions (RBF).

One-Class SVM, a variant of SVM, is used for anomaly detection by learning the majority class space and identifying deviations as anomalies. This is useful for detecting cyberattacks in datasets with mostly benign or few attack samples by training the model on benign patterns and identifying deviations as potential attacks. This method will be applied to unbalanced datasets where the samples are entirely benign or malicious.

IV. System setup and dataset creation

The platforms selected are two x86 CPUs from different manufacturers: Intel i5-8250U and an AMD Ryzen 7 3700X. In our analysis, some older architectures are not vulnerable to certain attack types. This ensures the ability to create a consistent dataset for each platform with multiple attacks. The hardware attacks selected run successfully in the host machines (thus, we guarantee we log ”real” traces). We use Lesimple's **spectre-meltdown-checker** script available on GitHub [16] for this purpose. This script analyzes computer characteristics and available mitigations and provides a list of successful hardware-based attacks on the computer. Section IV-C describes the attacks in detail.

The selection of benign programs is performed to ensure reliable and reproducible execution behavior that mirrors common workloads. Various benchmarks with different focuses will be chosen to maximize dataset coverage. Section IV-D describes the attacks in detail.

A. Selection of HPCs

The selected HPCs should accurately represent the patterns exploited by hardware-based attacks, enabling the detection of anomalies when compared to benign executions. The selected counters must also be generic enough to avoid dependence on specific architectures, ensuring the solution's portability across a wide range of computers. Experimentally, we found that there is a soft limit of four counters before some samples are lost in our system. Therefore, monitoring will be limited to four counters.

Some hardware attacks, like those in the Spectre family, exploit speculative execution, triggered when the branch predictor predicts the outcome of a branch instruction. Both `branch instructions` and `branch misses` are generic *perf* events, providing the ratio between the total branches and those where the predictor missed. This selection is supported by previous work, such as Congmiago Li et al. [6]. Additionally, many hardware-based attacks use side channels to extract information, which heavily stress the computer's cache memory. A high count of cache misses on the last-level cache (LLC) memory may indicate the presence of a FLUSH+RELOAD side-channel attack, known as the most effective and popular among hardware-based attacks. The first-level cache is also a common target in other attacks, as used by Stefano Carnà et al. [7]. Thus, the other two HPCs to be analyzed will be `LLC-load-misses` and `L1-dcache-load-misses`.

For dataset generation, the *perf* tool will be utilized. This tool enables recording of multiple HPCs during binary execution. The `perf stat` command will output HPC counts in a csv file format. To streamline operations, only one CPU core will be utilized, achieved using the `taskset` Linux tool, ensuring collected HPC data remains unaffected by workload distribution across cores.

B. Sample rate

Another decision to make is the sampling rate. Previous works have used sampling rates ranging from 1 ms per sample to 100 ms per sample. Congmiago Li et al. [6] even dynamically change the sample rate to prevent evasive malware. To generate a large number of samples for the ML model, the aim was to use the lowest possible sample rate. However, experimentally it was found that anything under 10 ms caused anomalies in *perf*, such as some samples not being recorded. Therefore, a 10 ms sample rate was chosen.

C. Selected hardware attacks

The selected hardware attacks are:

- **Meltdown**: among the most notorious hardware attacks, operates uniquely. While modern computers typically have the KPTI/KAISER mitigation against it, analyzing its behavior could be beneficial for the dataset. The Meltdown code was extracted from the IAIK GitHub repository [17].
- **Spectre V1, V2, and V4**: the infamous companion of Meltdown, has seen several versions released to date, with minor changes between them. A functional proof of concept (PoC) for Spectre V3 was not found, so it was skipped. Codes for the first [18], second [19], and fourth [20] versions of PoCs have been obtained from GitHub repositories.
- **ZombieLoad [21]**: similar to Meltdown, it captures sensitive data accessed by a user on a machine, and has been shown to work even on Meltdown-safe computers. The PoC by IAIK can be found on their GitHub repository [22].
- **Fallout [23]**: akin to Meltdown, leaks data from the CPU pipeline's store buffer and is classified under Microarchitectural Data Sampling (MDS) attacks along with RIDL [24]. The PoC code for the Fallout attack is sourced from Tristan Hornetz's GitHub repository [25].
- **Crosstalk [26]**: another MDS attack, aims to leak information between CPU cores, making it unique as it utilizes multiple cores, unlike other attacks. The source code for the proof of concept used is also obtained from Tristan Hornetz's GitHub repository [27].

These attacks bring the total number of malicious programs to 7. Other hardware attacks, like RIDL, Foreshadow [28], and ForeshadowNG [29], among others, were not selected because they rely on features not present in the laptop's architecture, such as Intel TSX [30].

D. Selection of benignware

The other half of the dataset will be generated using benign programs to contrast the behavior of the hardware attacks. To maintain balance, an equal number of benign programs (7) have been chosen:

- **Matrix multiplier**: A simple C program that multiplies large amounts of integer numbers to stress the computational sections of the CPU.
- **stress -c**: This is from the Debian `stress` tool, which stresses the CPU computing unit by repeatedly performing square roots of random numbers [31].
- **stress -m**: Also from the same tool, this option stresses the memory unit by repeatedly running malloc() and free() [31].
- **MiBench Bitcount**:A benchmark from the MiBench suite under the automotive category [32] available on Embecosm's Github repository [33]. It performs a bitcounting benchmark algorithm that stresses the CPU.
- **STREAM**: The STREAM benchmark [34], known for measuring memory bandwidth, will be used to stress the memory unit. The source code is available in Jeff Hammond's Github repository [35].
- **bzip2**: This is a high-quality lossless data compressor, chosen for both computational and memory workloads [36]. It will compress a fixed file, specifically the FreeBSD ISO image, to ensure replicability [37].
- **FFmpeg**: A multipurpose audio and video tool, used as a benchmark and example of a common mixed workload [38]. In this case, it will decode the "Big Buck Bunny" animation [39], commonly used for video testing [40], [41], [42].

The benchmarks selected are chosen to have similar execution profiles as the attacks listed. They either stress the memory unit, the CPU computational units or a mix. This ensures that the model can reliably distinguish malicious memory usage from memory-intensive workloads; and similarly for computational units or a mix.

V. Experiments and results

The first architecture used for testing is based on the Intel Core i5-8250U processor. The system runs Debian 11 (Bullseye) with the Linux kernel version 5.10.0. The processor operates at a maximum clock frequency of 3.4 GHz and features a 4-core / 8-thread configuration. This CPU belongs to Intel's Kaby Lake R (8th generation) family and is built using a 14nm process. It includes the following cache hierarchy:

- L1 Data Cache: 128 KiB
- L2 Cache: 1 MiB
- L3 Cache: 6 MiB

TABLE I
DATASETS, SCENARIOS AND SAMPLES

Dataset name	Representative scenario	%Samples/Type	Total samples
Balanced	Both representative benign applications and attacks are available for training. System administrator knows representative applications and attacks. The ML method has both information on what is benign and malign to make its prediction.	50% benign 50% malicious	28,000
Benign-only	Only representative benign applications are available for training. System administrator only knows the representative applications running on the system. Any other application will be deemed malign. The ML method turns into an anomaly detection setup.	100% benign	14,000
Malicious-only	Only representative malign applications are available for training. System administrator only knows the representative malign applications that can target the system. Any other application will be deemed benign. The ML method turns into an anomaly detection setup.	100% malicious	14,000

TABLE II
PARAMETERIZATION OF THE METHODS USED IN THE BALANCED DATASET

Method	Kernel	Parameters
Naive Bayes	Gaussian	n.a
	Multinomial	
Decision Tree	n.a	entropy, max_depth=10, min_samples_leaf=1, min_samples_split=5
		entropy, max_depth=None, min_samples_leaf=1, min_samples_split=2
Random Forest	n.a	Same optimal parameters of decision tree with 100 decision tree estimators
SVM	Lineal	C=1000
	Polinomial (second degree)	C=100
	RBF (Detection)	C=10, γ=10
	RBF (Classification)	C=100, γ=10

As mentioned in subsections IV-C and IV-D, we have a total of 14 programs: 7 benign and 7 malicious. From each of these programs, we obtain 2,000 samples, totaling 14,000 benign plus 14,000 malign samples. To conduct the experiments, we group these samples into three datasets: benign-only, malicious-only and balanced (both benign and malign samples). Table I shows for each dataset, its intended representative scenario and the proportion and type of samples. In all cases, 80% of the samples from each dataset were used for model training and 20% for testing.

A. Balanced dataset: Attack detection

For the balanced dataset, all the methods described in the previous section have been studied, except for the One-Class method (as it does not apply). For each case, hyperparameter tuning was performed using GridSearchCV [43], [44]. Table II shows the resulting hyperparameters for each method and variants or kernels studied in each case in the balanced dataset.

Table III shows the accuracy of each method studied. As shown in the table, Naive Bayes is the worst performing unless as it fails to detect benign samples correctly. We analyzed the case an it is caused by the non-independence between the counters used. The other 3 ML methods perform similarly (above 99% accuracy) being the SVM with RBF kernel the method that gives the best results for detection (i.e. benign/malign decision). Table IV shows the accuracy, recall, precision and F1-Score of this case (together with the best performing mechanisms of the next subsections).

TABLE III
PERFORMANCE OF MACHINE LEARNING MODELS ON THE BALANCED DATASET (INTEL CORE I5-8250U)

Method	Kernel	Metric
Naive Bayes	Gaussian	90% to detection and classification, but with the presence of false positives in detection, which impairs the for FFmpeg samples.
	Multinomial	53.69% to detection with problems to detect benign samples. By modifying the dataset to eliminate non independent HPCs, accuracy improves up to 90,89%.
Decision Tree	n.a	99.85% (Detection)
		99.79% (Classification)
Random Forest	n.a	99.94% (Detection)
		99.89% (Classification)
SVM	Lineal	99% detection and classification
	Polinomial (second degree)	99.8% detection and classification
	RBF	99.9% detection and classification

B. Balanced dataset: Classification

Beyond detecting if our system is under attack, we may want to know what kind of attack are we suffering to take specific remedial actions. As listed in Section IV-C, the attacks under study focus on different parts of the CPU and specific actions could be taken in each case.

We used the same ML methods as in the previous section to evaluate the effectiveness of detecting each different malign attack and benign application. Table III shows the accuracy results and, again, SVM with the RF Kernel is the most accurate. Figure 4 shows the confusion matrix for the classification using this method. Classification is nearly perfect (just 10 out of 24000 samples are misclassified and 5 being between Spectre V1 and Spectre V2).

C. Benign-only and malign-only

For the completely unbalanced datasets (benign and malicious), we used the One-Class SVM method with parameter values $\gamma = 1$ and $\nu = 0.01$ (obtained through GridSearchCV [43], [44]). Figures 2 and 3 show the confusion matrices of each dataset for detection. In the scenario where One-Class

SVM is used to detect benign samples, there are many false positives because the model encounters unseen samples during training and misclassifies them as malware. Similarly, in the case of using One-Class SVM to detect malicious samples, there are also numerous false positives due to the model's inability to accurately identify samples from the FFmpeg program. Overall, the F1-score is still over 90% in both cases, but not 99,9% as it is for the balanced dataset.

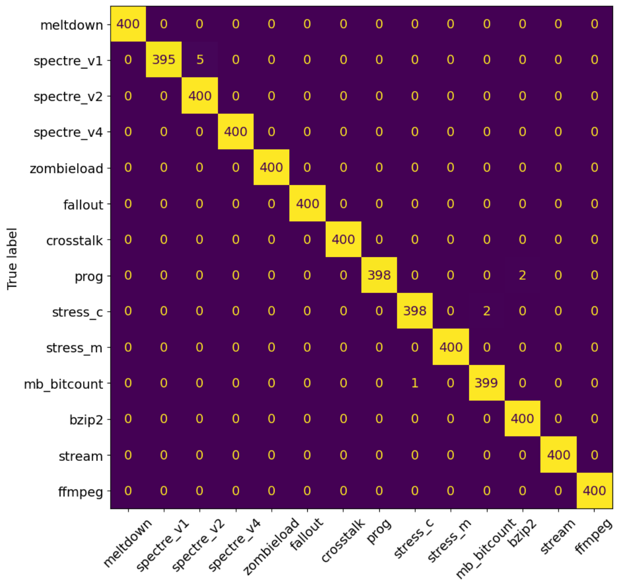


Fig. 1. Confusion matrix for classification using a RBF kernel SVM

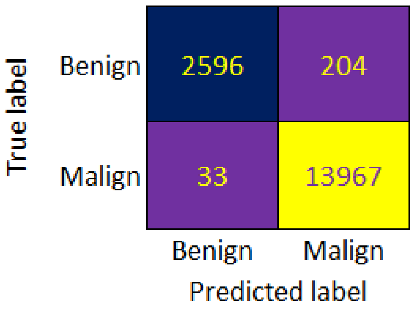


Fig. 2. Confusion matrix using a One-Class method and a benign dataset

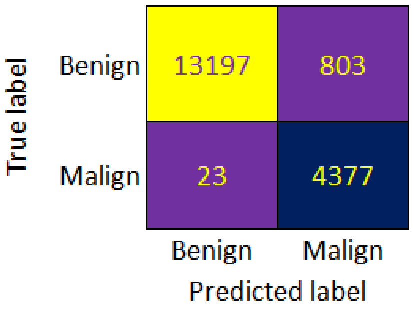


Fig. 3. Confusion matrix using a One-Class method and malicious dataset

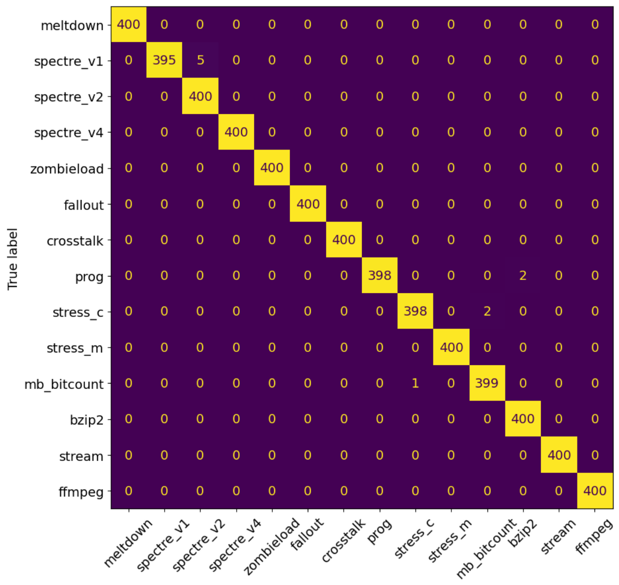


Fig. 4. Confusion matrix for classification using a RBF kernel SVM

TABLE IV
EVALUATION SUMMARY OF SVM-BASED MODELS ON INTEL CORE I5-8250U

Dataset	Method	Accuracy	Recall	Precision	F1
Balanced	SVM RBF (Detection)	99.96[a]%	99.92%	100%	99.96%
	SVM RBF (Classification)	99.91%	99.91%	99%	99.91%
Benign	One-Class SVM	98.5%	92.71%	98.74%	95.63%
Malicious	One-Class SVM	95.51%	99.48%	84.5%	91.38%

D. Summary of results on Intel Core i5-8250U

Table IV displays accuracy, recall, precision and F1-score for both the SVM RBF method and the One-Class method. For the balanced dataset, all values are above 99,9%, indicating very precise predictions. In contrast, for the only-benign and only-malign datasets, the recall and precision values indicate the a higher presence of false positives. Thus, clearly performing behind the balanced dataset.

E. Cross-Architecture Experimental Validation

To assess the portability of our approach across different hardware platforms, we replicated the dataset generation process on a machine with a different architecture. The second architecture is based on the AMD Ryzen 7 3700X processor. This system also runs Debian 11 (Bullseye) but with a more recent Linux kernel version 6.1.0. The CPU has a base clock frequency of 3.6 GHz and can boost up to 4.4 GHz. It features 8 cores and 16 threads, offering significantly more parallel processing capabilities than the first architecture. The processor is part of AMD's Zen 2 architecture, manufactured using a 7nm process. Its cache configuration is as follows:

- L1 Data Cache: 256 KiB
- L2 Cache: 4 MiB
- L3 Cache: 32 MiB

The only required modification involved adapting one hardware performance counter (HPC): the original counter for last-level cache load misses, LLC-load-misses, was unavailable on the AMD processor. Instead, we substituted it with an equivalent counter, l3_comb_clstr_state.request_miss [45], which provides analogous information regarding cache miss behavior on this architecture.

Using this setup, we generated a new dataset equivalent in structure and size to the original. We then evaluated the previously trained models: RBF SVM and the malign-trained One-Class SVM (without retuning the hyperparameters), maintaining consistency with the original system. As in the baseline experiments, we applied the same post-processing steps for prediction smoothing and result refinement.

Table V presents the results obtained using the same methods previously applied to the Intel-based architecture (Table IV).The results confirm that the models exhibit similar behavior to that observed on the Intel platform.

TABLE V
SUMMARY OF SVM-BASED MODELS ON AMD RYZEN 7 3700X

Dataset	Method	Accuracy	Recall	Precision	F1
Balanced	SVM RBF (Detection)	99.96%	100%	99.92%	99.95%
	SVM RBF (Classification)	98.92%	98.72%	99.2%	98.95%
Benign	One-Class SVM	98.32%	92.6%	97.41%	94.94%
Malicious	One-Class SVM	94.41%	99.96%	75.1%	85.76%

The two architectures employed in this study exhibit substantial differences in computational capabilities and target design. The Intel Core i5-8250U is a low-power, 8th-generation mobile processor featuring 4 cores and 8 threads, optimized for energy-efficient operation in portable devices. Conversely, the AMD Ryzen 7 3700X is a high-performance desktop processor with 8 cores and 16 threads, manufactured using a more advanced 7nm process. It offers higher base and boost frequencies, as well as significantly larger cache capacities—most notably a 32 MiB L3 cache compared to 6 MiB in the Intel counterpart. These architectural distinctions position the Ryzen 7 3700X as more suitable for compute-intensive and parallelizable workloads, while the i5-8250U is better aligned with lightweight, general-purpose computing in mobile environments.

Despite these disparities, the experimental findings indicate that reproducing the complete workflow—including dataset generation and model evaluation—on an alternative hardware platform yields consistent and reliable results. The models under evaluation (RBF SVM and One-Class SVM) attained comparable levels of accuracy, even though they were initially trained and hyperparameter-tuned on the Intel-based system. Although a minor degradation in performance was observed, it was largely mitigated through post-processing techniques. This performance gap is attributed to the hardware-specific nature of the original hyperparameter optimization, which was tailored to the Intel architecture.

VI. CONCLUSIONS

This work builds a reliable and reproducible dataset by using hardware counters to generate samples through the execution of 14 programs in two x86 CPUs (Intel and AMD). It then evaluates various ML models to determine the most effective model for detecting and classifying hardware attacks. Among the models evaluated, the SVM with RBF kernel showed superior performance in detecting and classifying attacks. With an accuracy and F1-score over 99.9% for both detection and classification tasks.

We also analyzed two scenarios where only the benign applications are known and only the malign applications are known. In these scenarios, the One-Class ML model was used and was capable of achieving an F1-score above 90% in both cases. Yet, significantly below the 99,9% of the balanced dataset. The larger amount of false positives reduced the F1-score accordingly.

REFERENCES

[1] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium*, 2018.

[2] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy*, 2019.

[3] J. Corbet, *Kaiser: Hiding the kernel from user space*, https://lwn.net/Articles/738975/, Accessed: 13-05-2024.

[4] D. Gens, O. Arias, D. Sullivan, C. Liebchen, Y. Jin, and A.-R. Sadeghi, "Lazarus: Practical side-channel resilient kernel-space randomization," in *Research in Attacks, Intrusions, and Defenses*, Springer International Publishing, 2017, pp. 238–258.

[5] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "Kaslr: Break it, fix it, repeat," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 481–493. DOI: 10.1145/3320269.3384747.

[6] C. Li and J.-L. Gaudiot, "Detecting spectre attacks using hardware performance counters," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1320–1331, 2022. DOI: 10.1109/TC.2021.3082471.

[7] S. Carnà, S. Ferracci, F. Quaglia, and A. Pellegrini, "Fight hardware with hardware: Systemwide detection and mitigation of side-channel attacks using performance counters," *Digital Threats*, vol. 4, no. 1, Mar. 2023. DOI: 10.1145/3519601.

[8] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016. DOI: https://doi.org/10.1016/j.asoc.2016.09.014.

[9] S. Bhattacharya and D. Mukhopadhyay, “Who watches the watchmen?: Utilizing performance monitors for compromising keys of rsa on intel platforms,” in *Cryptographic Hardware and Embedded Systems*, Springer Berlin Heidelberg, 2015, pp. 248–266.

[10] W. Kosasih, Y. Feng, C. Chuengsatiansup, Y. Yarom, and Z. Zhu, “Sok: Can we really detect cache side-channel attacks by monitoring performance counters?” In *19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 172–185. DOI: 10.1145/3634737.3637649.

[11] B. Otero Calviño, D. Andreu Gerique, and R. Canal Corretger, *Replication Data for: Hardware Attack detectoR via Performance counters analYsis Dataset (HARPY Dataset)*, version V1, 2025. DOI: 10.34810/data1982.

[12] C. D. Manning, P. Raghavan, and H. Schütze, “Introduction to information retrieval,” 2008.

[13] L. Rokach and O. Maimon, “Top-down induction of decision trees classifiers - a survey,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005. DOI: 10.1109/TSMCC.2004.843247.

[14] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, “Numerical Recipes 3rd edition: The Art of Scientific Computing,” 2007.

[16] S. L. (speed47), *Spectre-meltdown-checker*, https://github.com/speed47/spectre-meltdown-checker, 2023.

[17] I. of Applied Information Processing and C. (IAIK), *Meltdown*, https://github.com/IAIK/meltdown.

[18] R. C. (crozone), *Spectrepoc*, https://github.com/crozone/SpectrePoC.

[19] A. C. (Anton-Cao), *Spectrev2-poc*, https://github.com/Anton-Cao/spectrev2-poc.

[20] Y. S. (mmxsrup), *Cve-2018-3639*, https://github.com/mmxsrup/CVE-2018-3639.

[21] M. Schwarz *et al.*, “ZombieLoad: Cross-privilege-boundary data sampling,” in *CCS*, 2019.

[22] I. of Applied Information Processing and C. (IAIK), *Zombieload*, https://github.com/IAIK/ZombieLoad.

[23] C. Canella *et al.*, “Fallout: Leaking data on meltdown-resistant cpus,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019.

[24] S. van Schaik *et al.*, “RIDL: Rogue in-flight data load,” in *S&P*, May 2019.

[25] T. H. (tristan-hornetz), *Fallout*, https://github.com/tristan-hornetz/fallout.

[26] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, “CrossTalk: Speculative Data Leaks Across Cores Are Real,” in *S&P*, Intel Bounty Reward, May 2021. [Online]. Available: https://download.vusec.net/papers/crosstalk_sp21.pdf.

[27] T. H. (tristan-hornetz), *Crosstalk*, https://github.com/tristan-hornetz/crosstalk.

[28] J. Van Bulck *et al.*, “Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution,” in *Proceedings of the 27th USENIX Security Symposium*, See also technical report Foreshadow-NG [29], Aug. 2018.

[29] O. Weisse *et al.*, “Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution,” *Technical report*, 2018.

[30] *Intel® transactional synchronization extensions (intel® tsx) memory and performance monitoring update for intel® processors*, https://www.intel.com/content/www/us/en/support/articles/000059422/processors.html, Accessed: 28-05-2024, 2023.

[31] R. O. S. Projects, *Stress*, https://github.com/resurrecting-open-source-projects/stress.

[32] U. of Michigan, *Mibench version 1.0*, https://vhosts.eecs.umich.edu/mibench/, Accessed: 14-05-2024, 2002.

[33] Embecosm, *Mibench*, https://github.com/embecosm/mibench.

[34] J. D. McCalpin, *Stream: Sustainable memory bandwidth in high performance computers*, https://www.cs.virginia.edu/stream/, Accessed: 28-05-2024.

[35] J. H. (jeffhammond), *Stream*, https://github.com/jeffhammond/STREAM.

[36] *Bzip2*, https://sourceware.org/bzip2/, Accessed: 28-05-2024.

[37] *Parallel bzip2 compression benchmarks — openbenchmarking.org*, https://openbenchmarking.org/test/pts/compress-pbzip2, Accessed: 28-05-2024.

[38] *Ffmpeg*, https://ffmpeg.org/, Accessed: 28-05-2024.

[39] *Big buck bunny*, https://peach.blender.org/, Accessed: 28-05-2024.

[40] *Benchmark: Big buck bunny trailer*, https://dcpomatic.com/benchmarks/input.php?id=2, Accessed: 28-05-2024.

[41] *Ffmpeg rabbit benchmarks — openbenchmarking.org*, https://openbenchmarking.org/result/2311122-NE-FFMPEGRAB69, Accessed: 28-05-2024.

[42] *525.x264_r*, https://www.spec.org/cpu2017/Docs/benchmarks/525.x264_r.html, Accessed: 28-05-2024.

[43] P. M. Lerman, “Fitting segmented regression models by grid search,” *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 29, no. 1, pp. 77–84, Dec. 2018. DOI: 10.2307/2346413. eprint: https://academic.oup.com/jrsssc/article-pdf/29/1/77/48620247/jrsssc_29_1_77.pdf. [Online]. Available: https://doi.org/10.2307/2346413.

[44] *Gridsearch documentation*, https://scikit-earn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV, Accessed: 13-05-2025.

[45] *Re: Amd zen2* $l3_missesevent$, https://www.spinics.net/lists/linux-perf-users/msg17608.html, Accessed: 14-07-2024.

Queryable Microarchitecture Knowledge Base using Retrieval-Augmented Generation

Vignesh Manjunath
Graz University of Technology
Graz, Austria
vignesh.manjunath@student.tugraz.at

Jesus Pestana
Pro2Future GmbH
Graz, Austria
jesus.pestana@pro2future.at

Tobias Scheipel, Marcel Baunach
Graz University of Technology
Graz, Austria
{tobias.scheipel, baunach}@tugraz.at

Abstract—**Microarchitecture documentation, such as datasheets and user manuals, is indispensable for embedded software development. However, the extensive volume and complexity of these documents render information retrieval a time- and effort-intensive task. To address this challenge, we propose a framework for constructing a queryable knowledge base on microarchitecture documentation, leveraging Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs). As a proof of concept, we implement a knowledge base on AURIX TriCore TC27x documentation and evaluate this knowledge base by querying it with a curated set of questions. The generated responses are evaluated by measuring their semantic similarity to reference answers. In our evaluation, we assess the performance of six LLMs with different model architectures and sizes. The results show that the smaller models (with 8 billion and 3 billion parameters) achieve similarity scores comparable to those of the larger model (with 72 billion parameters). These initial findings demonstrate the robustness of our framework for creating queryable knowledge bases and the potential of smaller LLMs for efficient information retrieval in this context.**

Keywords-Embedded systems, information extraction, retrieval-augmented generation

I. Introduction

Embedded software development relies on microarchitecture documentation, including datasheets and user manuals, to implement device drivers and various software functionalities. These documents contain information on, e.g., peripheral configuration, memory management, and internal microcontroller behavior. However, finding relevant information is a time-consuming and effort-intensive task since these documents are often hundreds or even thousands of pages long. Moreover, the required information may be dispersed across various sections within a single document or distributed across multiple documents, making it challenging to obtain comprehensive information efficiently. For instance, peripheral configuration information is often fragmented across the datasheet, application notes, and errata documents.

To retrieve information quickly and efficiently, we propose a framework to build a queryable knowledge base on microarchitecture documentation. The main idea is to transform the target microarchitecture documentation into a structured knowledge base, which is subsequently integrated with an information retrieval process involving Retrieval-Augmented Generation (RAG) [1] and a Large Language Model (LLM) [2]. By integrating the knowledge base with the information retrieval process, the framework facilitates querying for Open-Domain Question Answering (ODQA) tasks. In addition, we implement a filtering concept to support document-specific information retrieval.

As a proof of concept and demonstration of our framework, we build a queryable knowledge base on AURIX TriCore TC27x [3] documentation. We evaluate the knowledge base using a set of questions and reference answers. First, we query the knowledge base with the questions and record the generated responses. Next, we compute the semantic similarity between generated responses and their corresponding reference answers. This similarity score reflects the quality of the responses in terms of their relevance and alignment with the reference answers.

The primary focus of this paper is on the development of the proposed framework and a preliminary evaluation to assess the performance of the framework. The framework currently uses a simple naive RAG pipeline with a single retriever to retrieve information from the knowledge base. Further refinement of the RAG pipeline and extensive evaluation of the approach are currently a work-in-progress and are out of scope for this paper.

The rest of the paper is organized as follows: Section II describes the methodology of the proposed framework. Section III presents the current evaluation approach and the preliminary results. Section IV discusses the related work, and Section V concludes the paper with a brief outlook on future work.

II. Methodology

The naive RAG pipeline in our framework consists of two primary components: a retriever, which is responsible for identifying and extracting the most relevant information from the knowledge base, and a generator (an LLM), which formulates a coherent and contextually appropriate response to a given user query based on the extracted information. In this section, we first explain the process of creating a structured knowledge base using the target microarchitecture documentation, followed by the process of information retrieval and response generation.

Manuscript received May 20, 2025; revised July 31, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i2.95

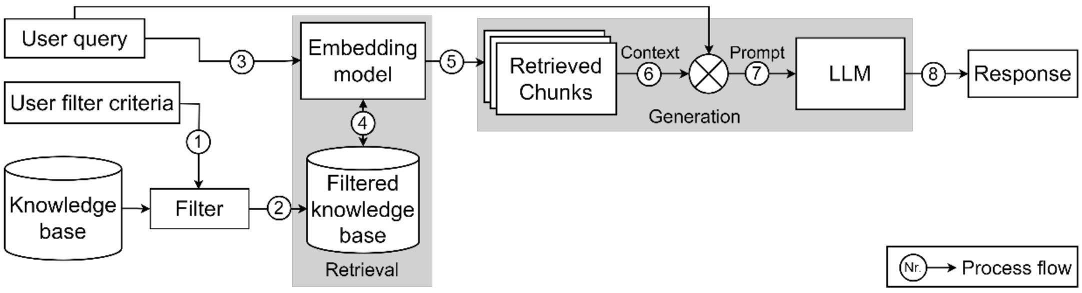


Figure 1: Information retrieval and response generation process.

A. *Knowledge Base Creation*

Microarchitecture documents are typically PDF files from hardware vendors. In general, these PDFs have complex table structures, images, and non-pertinent information, such as headers and footers. To make the information in the PDFs suitable for processing by an LLM, we first convert these PDFs into Markdown format using the PyPDF2 [4] Python library. Next, we remove non-pertinent information, translate figures into corresponding textual descriptions, and format complex table structures. Subsequently, we add metadata to every document, including details such as titles, versions, and tags.

Depending on the input PDFs, these Markdown files can be lengthy and may exceed the *context length* (i.e., the amount of text, in tokens, the model can process) of an LLM. Hence, we split the Markdown files into equally sized chunks based on word count. Next, we link each chunk with its corresponding document metadata and encode these chunks into dense vector representations (referred to as 'embeddings') using an embedding model (e.g., all-MiniLM-L6-v2 [5]). Lastly, we build and associate indexes for these embeddings using the FAISS library [6] to facilitate faster retrieval of document chunks relevant to a user query. The indexing step completes the creation of the knowledge base.

B. *Information Retrieval and Response Generation*

The information retrieval and response generation process begins with a user query and optional filter criteria and involves the sequence of steps (denoted by Ⓝ) illustrated in Figure 1.

In steps ① and ②, we filter the knowledge base and extract the embeddings corresponding to the document tag(s) specified by the user filter criteria. The resulting filtered knowledge base is then used to retrieve information relevant to the user query. If no filter criteria are specified, then the information is retrieved from the entire knowledge base.

In step ③, we encode the user query using the embedding model and then use the FAISS library to perform a similarity search on the knowledge base in step ④. The similarity search retrieves indexes of the most similar embeddings from the knowledge base, and these retrieved indexes are used to obtain the corresponding document chunks in step ⑤. Steps ③ through ⑤ represent the information retrieval process.

In step ⑥, the retrieved document chunks are concatenated as *context*. The *context* is then integrated with the user query and the rules for generation as a *prompt* in step ⑦. The rules instruct the LLM to generate a response based only on the provided *context*. The LLM uses the information contained in the *prompt* to generate the final response to the user query in step ⑧. Steps ⑥ through ⑧ correspond to the response generation process.

III. Proof of Concept and Evaluation

A. *Evaluation Setup*

To demonstrate and evaluate our framework, we implement a queryable knowledge base on a set of documents specific to the AURIX TriCore TC27x architecture. These documents include the core architecture user manuals, Instruction Set Architecture (ISA) description, and errata. We convert these documents into Markdown format and split them into chunks of 100 words each. Next, we encode these chunks into embeddings and then build and associate indexes with these embeddings. The resulting knowledge base is evaluated through semantic similarity analysis.

In our evaluation, we use Copilot and Nemotron [7] LLMs to generate a test dataset using the TC27x documents. The generated test dataset comprises 326 question-answer pairs, and we reviewed 25% of them to check their factual correctness. The answers in the test dataset serve as reference answers for evaluating the quality of the generated responses. Next, we integrate the TC27x knowledge base with the RAG pipeline and use the test dataset to benchmark six LLMs with different model architectures and sizes. Table 1 lists the LLMs under evaluation, and their short names represent the LLM family and the number of model parameters.

We conduct our evaluation by querying the LLM with the test questions and recording the generated responses. This evaluation is systematically repeated for all the LLMs under evaluation, and their responses are recorded. The evaluation is performed on a system equipped with three NVIDIA A100 80GB GPUs [10].

In addition to the response quality, we also measure the mean inference time for each LLM to assess its computational efficiency. As shown in Table 1, larger models exhibit higher inference times (e.g., Nemotron-70B at 28.25 s), while smaller models respond significantly faster (e.g., Llama3.2-1B at 1.70 s), illustrating the trade-off between model size and computational cost. In contrast, R1_DQwen_7B, although smaller than Nemotron_70B, exhibits a comparable inference time (27.34 s). This extended processing time is likely attributed to its chain-of-thought reasoning approach, which requires longer reasoning chains and tracking multiple logical branches, thereby increasing the computational effort required.

TABLE 1: LLMs UNDER EVALUATION.

LLM short name	**Number of model parameters**	**Model size (GiB)**	**Mean inference time (seconds)**
Llama3.2_1B [9]	1.23 billion	2.31	1.70
Llama3.2_3B [9]	3.21 billion	5.98	7.42
Qwen2.5_3B [8]	3.09 billion	5.76	5.66
R1_DQwen_7B [13]	7.62 billion	14.19	27.34
Llama3.1_8B [9]	8.03 billion	14.96	6.24
Nemotron_70B [7]	70.60 billion	131.5	28.25

B. Semantic Similarity Score Computation

In Natural Language Processing (NLP), semantic similarity scores are used to measure how closely two texts are aligned in meaning and context. In our work, we use an ensemble approach to compute the similarity score between the responses generated by different LLMs and their respective reference answers. The ensemble approach leverages two popular NLP metrics: BERTScore-F1 [11] and SBERT similarity score [12].

BERTScore-F1 measures how similar individual tokens are between two sentences by considering their meaning and context, while the SBERT similarity score compares the overall meaning of two sentences by transforming them into vector representations and measuring their closeness. Both scores range from -1 to +1, with values closer to +1 indicating a higher degree of similarity.

For each response generated by the LLMs under evaluation, we calculate the corresponding similarity scores, compute their means, and present the results in Table 2. The results indicate that the mean similarity scores remain consistent across both evaluation metrics. The model R1_DQwen_7B achieves the lowest mean similarity scores of all the LLMs under evaluation. This lower performance can be primarily due to two factors: (1) the inclusion of chain-of-thought reasoning in its responses, which introduces additional content, and (2) deviations in final answers, thereby reducing alignment with the expected outputs.

In contrast, most of the other smaller models achieve similarity scores closely aligned with those of the larger Nemotron_70B model. In particular, the smaller Llama3.1_8B model slightly outperforms the larger Nemotron_70B model, achieving the highest similarity score of 0.67 (highlighted using bold text in Table 2). This finding demonstrates the potential of smaller LLMs for effective information retrieval.

TABLE 2: MEAN SIMILARITY SCORE.

LLM short name	**Mean BERT score-F1**	**Mean SBERT similarity score**
Llama3.2_1B	0.57	0.61
Llama3.2_3B	0.65	0.65
Qwen2.5_3B	0.64	0.66
R1_DQwen_7B	0.50	0.49
Llama3.1_8B	**0.67**	**0.67**
Nemotron_70B	0.63	0.65

The similarity scores across models remain moderately close to +1, indicating a relatively high degree of similarity between the generated responses and their corresponding reference answers. This consistency highlights the robustness of our knowledge base framework and the computational efficiency of some smaller models, which are capable of generating contextually relevant outputs while significantly reducing GPU memory consumption and computational overhead compared to the larger Nemotron_70B model.

IV. RELATED WORK

In recent years, several approaches have leveraged various RAG architectures to address a broad range of tasks. Surveys such as [14-16] provide comprehensive overviews of RAG-based methods across multiple domains and applications, including domain-specific information retrieval, software safety analysis, and code generation. This section focuses specifically on existing approaches that employ RAG for domain-specific information retrieval.

Similar to our work, AeroQuery [17] and IDAS [18] use a naive RAG pipeline with vector similarity search to extract information from aerospace standards (e.g., DO-178C) and vehicle user manuals, respectively. In contrast, Kieu et al. [19] employ a hybrid retrieval approach that combines keyword-based and vector-based search results to enhance the explainability of AUTOSAR specifications. However, these approaches are evaluated on relatively small-scale datasets, typically involving only around 20 queries, which limits the generalizability and robustness of their findings.

Simoni et al. [20] introduce a multi-retriever RAG system that retrieves both textual information and code to answer cybersecurity-related queries. Similarly, Balu et al. [21] use multiple retrievers (one per document) to extract information from automotive standards. While both approaches reduce redundancy and summarize outputs from individual retrievers, the aggregated information can exceed the LLM's context length, potentially hindering response quality.

Some approaches [22–25] involve Graph-RAG, which retrieves relevant information from graph structures rather than isolated textual chunks. CyKG-RAG [22] applies this to cybersecurity by leveraging domain-specific knowledge graphs for multi-hop Q&A tasks. HSG-RAG [23] constructs hierarchical semantic knowledge graphs to improve retrieval from embedded systems documentation (such as API reference manuals). Liu et al. [24] use Graph-RAG to retrieve information from automotive software specifications, and Ojima et al. [25] extract information from event graphs representing automotive failure incidents. Although these methods demonstrate improved contextual retrieval, they often encounter challenges related to traceability and the limited context length of LLMs, particularly when aggregating information from numerous graph nodes or documents.

In contrast, our approach adopts a naive RAG pipeline augmented with a pre-retrieval filtering mechanism, which helps mitigate the context length limitations commonly encountered in graph-based or multi-retriever RAG systems. This filtering strategy enhances retrieval quality by selecting document chunks based on the user filter criteria, thereby improving the relevance of the retrieved information with respect to the user query. Furthermore, our preliminary evaluation results indicate that smaller LLMs can achieve performance levels comparable to their larger counterparts, thereby highlighting the feasibility of resource-efficient deployments without significant loss in retrieval quality.

V. Conclusion and Future Work

In this work, we presented a framework for building a queryable knowledge base on microarchitecture documentation using RAG with an LLM. Our proof-of-concept based on TC27x documentation demonstrates the feasibility of this approach for quick and efficient information retrieval in embedded software development. As a preliminary evaluation, we used semantic similarity metrics to assess the performance of six LLMs with different model architectures and sizes. The results show that smaller models, including those with 8 billion and 3 billion parameters, can achieve similarity scores comparable to those of a significantly larger model with 72 billion parameters. These findings highlight the robustness of our framework and the potential of smaller LLMs as resource-efficient alternatives for domain-specific information retrieval tasks.

While the preliminary evaluation demonstrates the feasibility of our approach, further work is required to enhance both the evaluation methodology and the underlying system. As future work, we plan to evaluate the factual correctness of the generated responses. This will involve developing or integrating more rigorous evaluation metrics and possibly including human-in-the-loop assessments.

In addition, we intend to refine the current naive RAG pipeline to improve retrieval quality. This includes optimizing document chunking strategies, enhancing query formulation, and exploring more advanced search and ranking strategies. These improvements are expected to increase the precision and relevance of retrieved content, thereby improving the robustness of our knowledge base framework.

Another important step in our future work is the implementation of an explicit traceability mechanism. Although naive RAG inherently allows tracking of generated responses back to the retrieved chunks, we intend to formalize this process by extracting and verifying the relevance of each chunk with respect to the final answer. This will enable more explainable and reliable responses, thereby increasing user trust in the knowledge base framework.

Finally, we plan to fully automate the conversion of source PDFs into structured markdown files. This includes extracting key elements like text, headings, and tables to streamline content preparation. Automating this step will significantly reduce manual preprocessing effort and ensure consistency and scalability in building and updating knowledge bases.

Acknowledgment

This work has been supported by the BMIMI, BMWET, and FFG, Contract No. 911655: "Pro²Future II", Graz University of Technology (TU Graz), and Elektrobit Automotive GmbH.

References

[1] P. Lewis et al., ‘Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks’, in *Advances in Neural Information Processing Systems*, 2020, vol. 33.

[2] V. Ashish et al., ‘Attention is all you need’, Advances in neural information processing systems, vol. 30, p. I, 2017.

[3] ‘AURIX TC27x D-Step 32-bit Single-Chip Microcontroller User Manual v2.2’, Infineon Technologies AG.

[4] ‘PyPDF2’. [Online]. Available: https://pypdf2.readthedocs.io/en/3.x/

[5] ‘Sentence-Transformer Model: all-MiniLM-L6-v2’. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[6] M. Douze et al., ‘The FAISS library’, arXiv preprint arXiv:2401. 08281, 2024.

[7] W. Zhilin et al., ‘HelpSteer2-Preference: Complementing Ratings with Preferences’, arXiv [cs.LG]. 2024.

[8] Q. Team, ‘Qwen2.5: A Party of Foundation Models’. Sep-2024. Available: https://qwenlm.github.io/blog/qwen2.5/

[9] A. Grattafiori et al., ‘The llama 3 herd of models’, arXiv preprint arXiv:2407. 21783, 2024.

[10] Nvidia, ‘LLM Benchmarking: Fundamental Concepts’. [Online]. Available: https://developer.nvidia.com/blog/llm-benchmarking-fundamental-concepts/.

[11] T. Zhang and Others, ‘Bertscore: Evaluating text generation with bert’, arXiv preprint arXiv:1904. 09675, 2019.

[12] N. Reimers and I. Gurevych, ‘Sentence-bert: Sentence embeddings using siamese bert-networks’, arXiv preprint arXiv:1908. 10084, 2019.

[13] DeepSeek-AI, ‘DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning’, arXiv [cs.CL]. 2025.

[14] M. Arslan et al., ‘A Survey on RAG with LLMs’, Procedia Computer Science, vol. 246, pp. 3781–3790, 2024.

[15] Y. Gao et al., ‘Retrieval-augmented generation for Large Language Models: A survey’, arXiv preprint arXiv:2312. 10997, vol. 2, p. 1, 2023.

[16] R. Chen et al., ‘Retrieval-Augmented Generation with Knowledge Graphs: A Survey’, in Computer Science Undergradaute Conference 2025@ XJTU.

[17] S. Yadav, ‘AeroQuery RAG and LLM for Aerospace Query in Designs, Development, Standards, Certifications’, in 2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2024, pp. 1–6.

[18] B. Hernandez-Salinas et al., ‘IDAS: Intelligent Driving Assistance System using RAG’, IEEE Open Journal of Vehicular Technology, 2024.

[19] K. Kieu et al., ‘Empowering Automotive Software Development with LLM-RAG Integration’, 2024.

[20] M. Simoni et al., ‘Morse: Bridging the gap in cybersecurity expertise with retrieval augmented generation’, in Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing, 2025, pp. 1213–1222.

[21] B. V. Balu et al., ‘Towards Automated Safety Requirements Derivation Using Agent-based RAG’, arXiv preprint arXiv:2504. 11243, 2025.

[22] K. Kurniawan et al., ‘CyKG-RAG: Towards knowledge-graph enhanced retrieval augmented generation for cybersecurity’, 2024.

[23] Z. Lu et al., ‘HSG-RAG: Hierarchical Knowledge Base Construction for Embedded System Development’, ACM Transactions on Design Automation of Electronic Systems.

[24] F. Liu et al., ‘Enhancing Automotive PDF Chatbots: A Graph RAG Approach with Custom Function Calling for Locally Deployed Ollama Models’, in Proceedings of the 2024 International Conference on Artificial Intelligence, Digital Media Technology and Interaction Design, 2024, pp. 6–13.

[25] Y. Ojima et al., ‘Knowledge Management for Automobile Failure Analysis Using Graph RAG’, in 2024 IEEE International Conference on Big Data (BigData), 2024, pp. 6624–663

An Approach for Automotive ECU Diagnosis via Ethernet Snooping & Microcontroller Tracing

Zafer Attal
Chair of Integrated Systems
Technical University of Munich
Munich, Germany

Matthias Ernst
Infineon Technologies AG
Munich, Germany

Gasper Skvarc Bozic
Infineon Technologies AG
Munich, Germany

Ibai Irigoyen Ceberio
Infineon Technologies AG
Munich, Germany

Albrecht Mayer
Infineon Technologies AG
Munich, Germany

Thomas Wild
Chair of Integrated Systems
Technical University of Munich
Munich, Germany

Andreas Herkersdorf
Chair of Integrated Systems
Technical University of Munich
Munich, Germany

***Abstract*—The increasing software complexity in modern vehicles necessitates diagnostic capabilities beyond traditional systems. This paper presents a Diagnosis Unit (DU) that supports runtime detection and analysis of anomalies by correlating irregularities in Ethernet communication with ECU-internal processing behavior. The DU captures execution traces upon detecting anomalous communication and performs localized analysis to assist in uncovering potential root causes. Implemented on a ZCU102 platform and interfaced with Aurix ECUs, the prototype effectively detects both communication and processing anomalies with minimal impact on in-vehicle network bandwidth, supporting scalable, adaptive, and non-intrusive in-vehicle diagnostics.**

Keywords- Automotive, Diagnostics, Health Monitoring, Anomaly Detection, Trace Analysis

I. Introduction

Modern vehicles are evolving into software-defined systems, built on complex architectures with hundred of interconnected Electronic Control Units (ECUs) [1]. As vehicle functionality—from driver assistance to autonomous operation—relies heavily on software, the associated computational demands introduce significant challenges for software reliability and, consequently, for fault diagnosis. Traditional On-Board Diagnostics (OBD) systems [2], though effective for hardware faults, are not designed to detect transient, software-induced anomalies in real-world operation.

Recent diagnostic methods have begun addressing these limitations, yet they often fall short in correlating anomalies between network-level symptoms and ECU-internal behaviors. This gap is critical, as many communication irregularities may reflect deeper malfunctions within individual ECUs or their subsystem interactions.

We present a Diagnosis Unit (DU) that supports correlation-based fault analysis by monitoring in-vehicle Ethernet communication and retrieving execution traces from the responsible ECU. Integrated in a non-intrusive manner at a gateway or central service node (Fig. 1), the DU performs targeted, online trace-based analysis with minimal impact on system behavior, identifying irregularities as they manifest on the IVN.

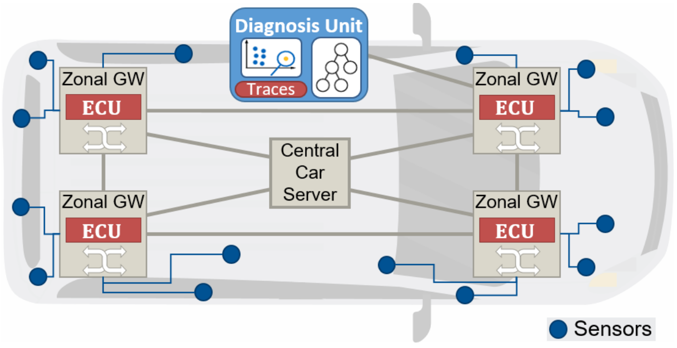


Figure 1. Diagnosis Unit deployment in a IVN.

Implemented on a ZCU102 platform and interfaced with Aurix ECUs, our prototype demonstrates the feasibility of detecting and correlating communication and ECU-internal processing anomalies.

The rest of this paper covers related work (II), the DU concept (III), system architecture and demonstration setup (IV), detection results and system performance (V), and conclusions with future directions (VI).

II. Related Work

Traditional diagnostic systems such as On-Board Diagnostics (OBD-II) are effective for hardware-level faults, but they are not designed to address dynamic, software-induced anomalies in modern vehicles [2]. As software complexity in in-vehicle systems increases, researchers have explored complementary diagnostic approaches.

Cloud-centric systems enhance diagnostic coverage by offloading data to backend processors for deeper analysis [3]. However, this approach incurs high bandwidth costs and cannot

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0457 (EMDRIVE).

Manuscript received April 30, 2025; revised July 23, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i2.96

provide timely responses within the vehicle. Similarly, automated trace preprocessing frameworks, as shown in [4], offer rich analysis capabilities but are typically designed for offline use during system validation and lack feasibility for deployment in online scenarios within operational vehicles.

Vehicle Health Monitoring Systems (VHMSs) incorporate predictive diagnostics through sensor analytics and machine learning [5][6]. However, they often focus on individual subsystems and struggle to correlate behavior across architectural domains. Similarly, advanced anomaly detection models [7][8] emphasize pattern recognition in communication or control flows but fall short in identifying causal relationships between network anomalies and ECU processing behavior.

In contrast, the proposed Diagnosis Unit (DU) operates locally and autonomously within the vehicle. It monitors Ethernet communication for anomalies and retrieves ECU execution traces to analyze them for potential correlations. Rather than replacing existing diagnostics, the DU complements them by delivering runtime insights that support the identification of potential root causes.

III. Design and Operating Principles

A. Diagnosis Unit Subsystems

The DU comprises three tightly integrated components:

- **Gateway Snooping:** Passively monitors mirrored Ethernet traffic to detect timing or behavioral anomalies without disrupting normal operation.
- **Trace Control System:** Upon detecting an anomaly, the DU identifies the affected ECU and initiates trace recording via its Tool Access Socket (TAS) server. This requires ECUs to support hardware tracing and tooling for remote trace configuration and retrieval.
- **Trace Analyzer:** Retrieved traces are analyzed during runtime to identify irregularities such as delayed functions, excessive execution time, or control-flow deviations potentially linked to the observed communication anomaly.

This modular structure supports localized, event-driven diagnostics without requiring continuous cloud connectivity. The current prototype operates autonomously, with all core logic implemented on a ZCU102 platform.

B. Timing Requirements for Effective Trace Capture

Effective diagnosis depends on capturing the relevant processing history in the trace buffer corresponding to the observed anomaly. This observable history depends on the size of the trace buffer and on the tracing granularity—i.e., how many trace events are recorded per time unit. To ensure the trace includes the necessary context, the Recording Window (T_{RW}) must satisfy:

$$T_{RW} \geq \Delta t_{AProp} + \Delta t_{DU} \quad (1)$$

Here, Δt_{AProp} represents the internal propagation delay before a processing anomaly manifests on the network. Δt_{DU} includes anomaly detection, identification of the source ECU, and the time to stop tracing and initiate trace retrieval. Given the limited size of the trace buffer and the risk of overwriting older entries, this constraint ensures the capture of causally relevant events.

Additionally, Δt_{TT} denotes the time required to transfer the trace from the ECU to the DU, though it does not impact the critical timing path for trace preservation. Fig. 2 illustrates the timing relationship between these components.

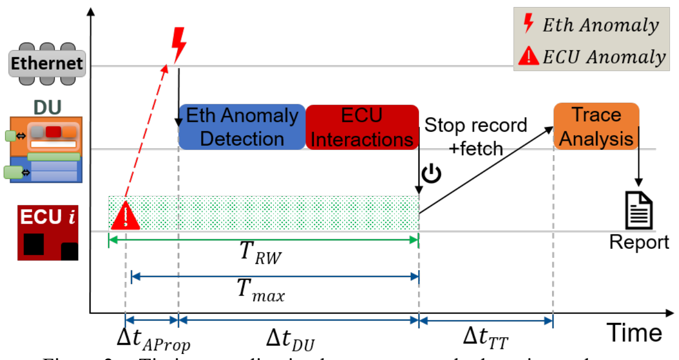


Figure 2. Timing coordination between anomaly detection and trace recording.

C. Diagnostic Advantages

The DU enables localized correlation between communication symptoms and processing anomalies during vehicle operation, eliminating the need for continuous cloud uploads or offline trace post-processing. It operates non-intrusively—without ECU software instrumentation—and integrated via gateway snooping and standard debug interfaces, assuming an existing ECU tracing subsystem. These features support future extensions such as cloud-assisted reconfiguration and adaptive anomaly classification for scalable, fleet-wide diagnosis.

IV. System Architecture & Demonstration Setup

A. Diagnosis Unit Implementation

The Diagnosis Unit (DU) is prototyped on a Xilinx ZCU102 board, which integrates a Zynq UltraScale+ MPSoC featuring programmable logic and a quad-core ARM Cortex-A53 processing system [9]. This heterogeneous architecture enables a clear separation between time-critical data-plane functions and flexible control-plane logic. The programmable logic (PL) hosts a custom hardware module for Ethernet traffic monitoring, anomaly detection, and timestamping with cycle-level precision. The processing system (PS) runs embedded Linux and hosts the DU Manager, which coordinates the Tool Access Socket (TAS) server [10], manages trace configurations and retrieval, and performs local analysis of ECU traces.

As shown in Fig. 3, the DU connects its monitoring port to a mirroring port on the in-vehicle Ethernet switch, ensuring non-intrusive monitoring of communication traffic. Upon detecting a communication anomaly, it identifies the affected ECU and configures trace capture via a Tool Access Socket (TAS) server. Retrieved traces are analyzed locally on the DU without relying on external computation resources during runtime. After analysis, the DU generates a compacted report summarizing the detected communication and processing anomalies.

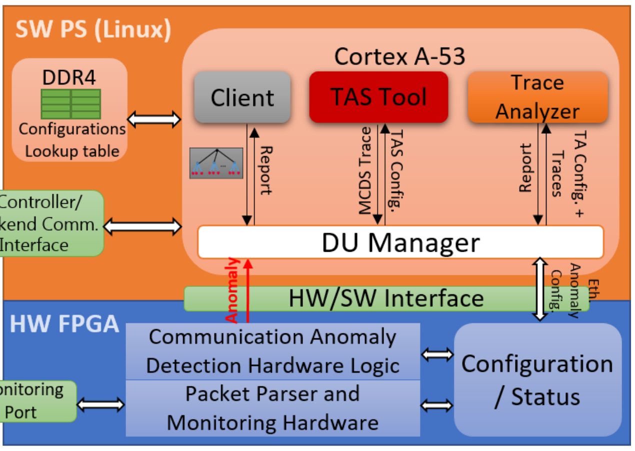


Figure 3. Modular Architecture of the Diagnosis Unit Prototype.

B. *Demonstration Setup*

To validate the DU's diagnostic capabilities, we developed a demonstration platform that combines real automotive hardware with simulation. The setup features three Infineon Aurix TC397 microcontroller boards, each equipped with a Multi-Core Debug Solution (MCDS) for trace recording [11]. These boards simulate different ECUs making up a distributed lane-keeping assistant application. The ECUs are connected to the CARLA simulator, which supplies real-time vehicle sensor inputs from a dynamic driving environment and receives steering commands.

The DU monitors Ethernet traffic for *communication anomalies*, including timing irregularities, missing messages, and burst structure deviations. When such anomalies are detected, the DU triggers trace collection to analyze *processing-level anomalies* such as delayed functions, task overruns, or atypical execution sequences. Fig. 4 shows the demonstration setup with integrated hardware components.

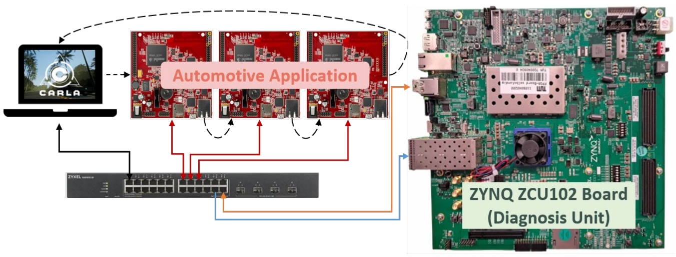


Figure 4. Demonstration Setup with Aurix ECUs and ZCU102-based Diagnosis Unit.

C. *Anomaly Detection Criteria*

To evaluate cross-domain detection capabilities, both communication and processing anomalies were deliberately introduced. Communication-level anomalies included timing deviations, altered periodicity, and packet drop patterns, as formally defined in Table I. Each anomaly type was detected based on predefined inequality-based thresholds for timing or packet count deviations.

TABLE I. COMM. ANOMALY TYPES MONITORED IN PROTOTYPE

Anomaly	Detection Rule
Timing Deviation Between Bursts	$T_B < TP_B - \Delta T_B$ *or* $TP_B + \Delta T_B < T_B$ [1]
Timing Deviation Between Packets	$T_P < TP_P - \Delta T_P$ *or* $TP_P + \Delta T_P < T_P$ [2]
Packet Count Deviation in Bursts	$P_{rec} < P_{exp} - \Delta P$ *or* $P_{exp} + \Delta P < P_{rec}$ [3]

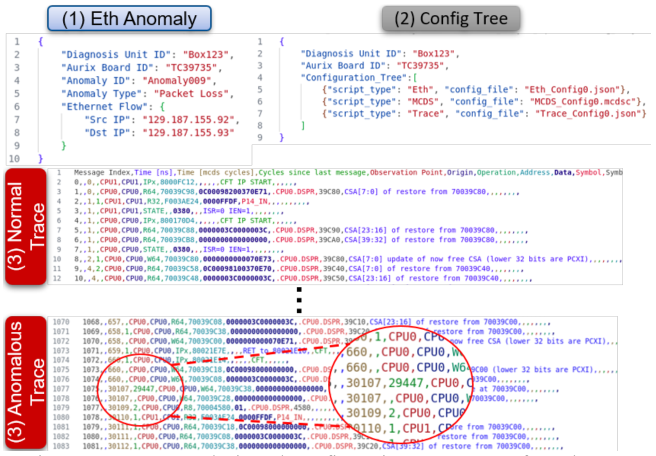
1. TP_B : Expected inter-burst interval, T_B : Observed inter-burst interval, ΔT_B : Burst corridor width threshold.
2. TP_P : Expected inter-packet interval, T_P : Observed inter-packet interval, ΔT_P : Packet corridor width threshold.
3. P_{exp} : Expected number of packets per burst, P_{rec} : Observed number of packets, ΔP: Burst size toleranc.

V. RESULTS AND OBSERVATIONS

A. *Demonstration of Cross-Domain Anomaly Localization*

Upon detecting a communication anomaly, the DU identified the source ECU and triggered trace retrieval via the TAS server. These traces enabled the analysis of related processing anomalies, such as prolonged execution delays, misordered instruction/function sequences, and irregular task load distribution. By linking anomalies in the communication domain with internal ECU behaviors, the DU demonstrated correlated, runtime insights across system domains.

Fig. 5 illustrates a trace excerpt highlighting instruction-level delays identified after a detected communication anomaly, confirming a processing deviation within the implicated ECU. Fig. 6 shows the physical prototype setup used for demonstration, featuring the ZCU102-based DU and connected Aurix ECUs with the Carla simulator as an environment for automotive application.



```
{
    "Diagnosis Unit ID": "Box123",
    "Aurix Board ID": "TC39735",
    "Anomaly ID": "Anomaly009",
    "Anomaly Type": "Packet Loss",
    "Ethernet Flow": {
        "Src IP": "129.187.155.92",
        "Dst IP": "129.187.155.93"
    }
}
```

```
{
    "Diagnosis Unit ID": "Box123",
    "Aurix Board ID": "TC39735",
    "Configuration_Tree":[
        {"script_type": "Eth", "config_file": "Eth_Config0.json"},
        {"script_type": "MCDS", "config_file": "MCDS_Config0.mcdsc"},
        {"script_type": "Trace", "config_file": "Trace_Config0.json"}
    ]
}
```

```
Message Index,Time [ns],Time [mcds cycles],Cycles since last message,Observation Point,Origin,Operation,Address,Data,Symbol,Symb
0,,0,,CPU1,CPU1,IPx,8000FC12,,,,,,CFT IP START,,,,,
1,,0,,CPU0,CPU0,R64,70039C98,0C00098200370E71,.CPU0.DSPR,39C80,CSA[7:0] of restore from 70039C80,,,,,,,
2,,1,1,CPU1,CPU1,R32,F003AE24,0000FFDF,P14_IN,,,,,,,,,
3,,1,,CPU1,CPU1,STATE,,0380,,,ISR=0 IEN=1,,,,,,,
4,,1,,CPU0,CPU0,IPx,800170D4,,,,,,CFT IP START,,,,,
5,,1,,CPU0,CPU0,R64,70039C88,0000003C0000003C,.CPU0.DSPR,39C90,CSA[23:16] of restore from 70039C80,,,,,,,
6,,1,,CPU0,CPU0,R64,70039CB8,0000000000000000,.CPU0.DSPR,39CA0,CSA[39:32] of restore from 70039C80,,,,,,,
7,,1,,CPU0,CPU0,STATE,,0380,,,ISR=0 IEN=1,,,,,,,
8,,2,1,CPU0,CPU0,W64,70039C80,0000000000070E73,.CPU0.DSPR,39C80,CSA[7:0] update of now free CSA (lower 32 bits are PCXI),,,,,,,
9,,4,2,CPU0,CPU0,R64,70039C58,0C00098100370E70,.CPU0.DSPR,39C40,CSA[7:0] of restore from 70039C40,,,,,,,
10,,4,,CPU0,CPU0,R64,70039C48,0000003C0000003C,.CPU0.DSPR,39C50,CSA[23:16] of restore from 70039C40,,,,,,,
```

```
1068,,657,,CPU0,CPU0,R64,70039C08,0000003C0000003C,.CPU0.DSPR,39C10,CSA[23:16] of restore from 70039C00,,,,,,,
1069,,658,1,CPU0,CPU0,R64,70039C38,0000000000000000,.CPU0.DSPR,39C2 [illegible] restore from 70039C00,,,,,,,
1070,,658,,CPU0,CPU0,W64,70039C00,0000000000070E71,.CPU0.DSPR,39 [illegible] now free CSA (lower 32 bits are PCXI),,
1071,,659,1,CPU0,CPU0,IPx,80021E7E,,,,RET to 80021E10,,CFT,,,
1072,,660,1,CPU0,CPU0,IPx,80021E14,,,,,,CFT,,,,,
1073,,660,,CPU0,CPU0,W64,70039C18,0C00098000000000,.CPU0.DS [illegible] 9C00 (lower 32 bits are PCXI),,,,,
1074,,660,,CPU0,CPU0,W64,70039C08,0000003C0000003C,.CPU0.D [illegible] 39C00,,,,,,,
1075,,30107,29447,CPU0,CPU0,W64,70039C38,0000000000000000, [illegible] at 70039C00,,,,,,,
1076,,30107,,CPU0,CPU0,W64,70039C28,0000000000000000,.CPU0. [illegible] 70039C00,,,,,,,
1077,,30109,2,CPU0,CPU0,R8,70004580,01,.CPU0.DSPR,4580,,,,,
1078,,30110,1,CPU1,CPU1,R32,F003AE24,0000FFDF,P14_IN,,,,,,,,,
1079,,30111,1,CPU0,CPU0,R64,70039C18,0C00098000000000,.CPU0.DSP [illegible] re from 70039C00,,,,,,,
1080,,30111,,CPU0,CPU0,R64,70039C08,0000003C0000003C,.CPU0.DSPR,39C [illegible] restore from 70039C00,,,,,,,
1081,,30112,1,CPU0,CPU0,R64,70039C38,0000000000000000,.CPU0.DSPR,39C20,CSA[39:32] of restore from 70039C00,,,,,,,
```

```
,660,,CPU0,CPU0,W
,660,,CPU0,CPU0,W64
,,30107,29447,CPU0,C
,,30107,,CPU0,CPU0,W
,30109,2,CPU0,CPU0
0110,1,CPU1,C
```

Figure 5. Trace Analysis and Configuration Tree Output from the DU Prototype.

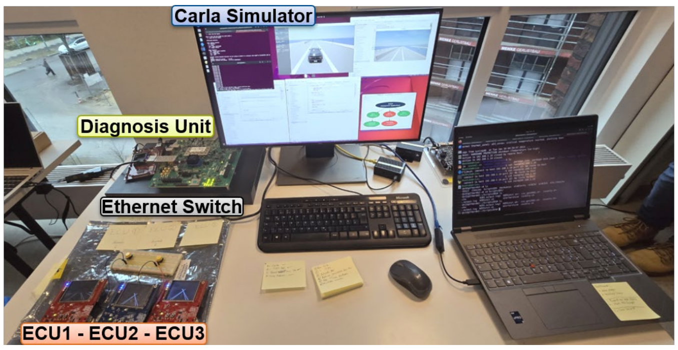


Figure 6. Physical demonstration of the Diagnosis Unit.

B. Detection Performance and Timing Behavior

To evaluate the responsiveness of the Diagnosis Unit, we measured its reaction time—defined here as the interval between the appearance of a communication anomaly on the Ethernet interface and the moment trace recording is halted. This time is critical to ensure that the trace buffer still retains the relevant processing history preceding the anomaly. While the propagation time from processing anomaly to their network manifestation is application-dependent, our measurement focuses on the DU's ability to respond quickly once a network-level deviation is observed.

Across multiple runs and under the configured trace buffer size, the DU required between 100 and 500 milliseconds to respond—depending on the recording granularity and the number of ECUs being traced concurrently. These values align with the buffer timing constraints outlined in Section III-B, ensuring adequate preservation of pre-anomaly trace context.

Trace retrieval occurred at approximately 6 MBps via the debug interface per ECU, indicating low bandwidth requirements on the IVN—an important factor in maintaining system non-intrusiveness. The subsequent local analysis of the retrieved trace typically completed within 300 milliseconds on average per anomaly case, depending on the trace length and granularity of recorded events. These results suggest that the DU is capable of real-time diagnosis while introducing low processing or communication overhead, supporting scalable in-vehicle deployment.

C. System Constraints and Prototype Limitations

The current prototype implementation presents practical constraints affecting diagnostic coverage and flexibility. A primary limitation is the 2~MB trace buffer per ECU, which restricts the retained processing history—particularly under fine-grained trace configurations where verbose logging can saturate the buffer. This bounds the diagnostic window and necessitates precise coordination between anomaly detection and trace retrieval.

While the DU performs local analysis independently of backend connectivity, its current evaluation logic is limited to predefined rule-based models. Planned backend integration—for dynamic rule updates and multi-vehicle correlation—was not included in the evaluated prototype. Similarly, advanced diagnostic methods, such as statistical learning or adaptive behavioral profiling, remain future work.

VI. Conclusion and Outlook

This work presents a cross-domain diagnostic approach—embodied in our Diagnosis Unit (DU)—that enables runtime anomaly detection in automotive systems by correlating communication anomalies with internal ECU processing behavior. Implemented on a ZCU102 platform and validated through a distributed lane-keeping assistant setup, the DU demonstrated its ability to localize anomalies efficiently and with low bandwidth overhead.

The DU is integrated via a mirrored switch port and debug interfaces, enabling non-intrusive deployment without requiring software modifications. Its modular design supports local, on-demand trace analysis, enhancing in-vehicle observability while minimizing reliance on backend infrastructure.

For fleet-scale deployment, distributed DUs autonomously detect and correlate anomalous events, forwarding concise reports to the backend. This low-bandwidth setup reduces network load, preserves privacy, and enables scalable diagnostics. Future versions will support cloud connectivity for remote control of diagnosis policies, result aggregation, and dynamic detection model updates.

Several enhancements are envisioned to improve the DU's precision and adaptability: (i) Semantic-level anomaly detection, such as recognizing out-of-range signals (e.g., sensor or actuator values); (ii) Learning-based classification to handle evolving or sporadic faults; (iii) Secure trace handling and integration with IVN security mechanisms to support encrypted communication monitoring.

Finally, for cost-effective deployment in production vehicles, we propose embedding DU functionalities directly into gateway Network Interface Controllers (NICs). Together, these improvements aim to deliver a scalable, resilient, and secure diagnostic infrastructure suited for the growing complexity of modern automotive systems.

Acknowledgment

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0457 (EMDRIVE). The responsibility for the content of this publication lies with the authors.

References

[1] SAE J3016 automated-driving graphic. Last Modified: 2020-05-15T14:03:15-04:00.

[2] Deepa Saibannavar, Mallikarjun M. Math, and Umakant P. Kulkarni. A survey on on-board diagnostic in vehicles. 2020.

[3] Malintha Amarasinghe, Sasikala Kottegoda, Asiri Liyana Arachchi, Shashika Ranga Muramudalige, Herath Mudiyanselage Nelanga Dilum Bandara, and Afkham Azeez. Cloud-based driver monitoring and vehicle diagnostic with obd2 telematics. 2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer), pages 243–249, 2015.

[4] Artur Mrowca, Thomas Pramsohler, Sebastian Steinhorst, and Uwe Baumgarten. Automated interpretation and reduction of in-vehicle network traces at a large scale. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pages 1–6, 2018.

[5] Md. Arafatur Rahman, Md. Abdur Rahim, Md. Mustafizur Rahman, Nour Moustafa, Imran Razzak, Tanvir Ahmad, and Mohammad N. Patwary. A secure and intelligent framework for vehicle health monitoring exploiting big-data analytics. IEEE Transactions on Intelligent Transportation Systems, 23:19727–19742, 2022.

[6] Uferah Shafi, Asad Ali Safi, Ahmad Raza Shahid, Sheikh Ziauddin, and Muhammad Qaiser Saleem. Vehicle remote health monitoring and prognostic maintenance system. Journal of Advanced Transportation, 2018:1–10, 2018.

[7] ¨Ovg¨u ¨Ozdemir, M. Tu˘gberk ˙Is¸yapar, Pınar Karag¨oz, Klaus Werner Schmidt, Demet Demir, and N. Alpay Karag¨oz. A survey of anomaly detection in in-vehicle networks. arXiv preprint arXiv:2409.07505, 2024.

[8] Paolo Dini, Sergio Saponara, Carlo Rosadini, Walter Nesci, and Stefano Chiarelli. Anomaly and intrusion detection algorithms for can-bus networking security in automotive applications. Automotive SPIN Italia Workshop, 2023.

[9] AMD. Zcu102 evaluation kit board user guide, 2024. Accessed: 2024-08-08.

[10] A new generation automotive tool access architecture for remote in-field diagnosis. SAE Technical Paper Series, 2023.

[11] Lauterbach GmbH. Multi-core debug solution user's guide, 2024. Accessed: 2024-08-08. G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

Works in Progress in Embedded Computing (WiPiEC) Journal, vol. 11, no. 1, 2025

Prompt-to-Metric: LLMs and Graph Algorithms for Platform Ecosystem Health Monitoring

Shady Hegazy, Muhammad Ammar, Christoph Elsner
Siemens Foundational Technologies
Siemens AG
Munich, Germany
Firstname.lastname@siemens.com

Jan Bosch
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden
Jan.bosch@chalmers.se

Helena Holmström-Olsson
Department of Computer Science and Media Technology
Malmö University
Malmö, Sweden
Helena.holmstrom.olsson@mau.se

Abstract—**Platform ecosystems are networks of interconnected actors co-creating value through a shared technological platform. Such socio-technical systems require unique key performance indicators and health evaluation metrics to address the unique characteristics and value-creation modes they entail. Several platform ecosystems health evaluation models have been suggested in literature, along with a plethora of metrics. This study presents Prompt-to-Metric, a system that allows users, mainly platform orchestrators and decision-makers, to monitor the health of a platform ecosystem through natural language queries. The system relies on a KPI network of approximately 400 health metrics classified across four levels of hierarchy according to a model developed through a systematic literature review on the topic. In addition, the pipeline uses graph algorithms to enhance the relevancy of the responses and uncover insights regarding metrics relatedness. The system was implemented as a prototype and is being evaluated for feasibility in real-world application scenarios using data from an operational platform ecosystem. Future work includes expanding the set of calculable metrics, improving response relevance, and further evaluation in real-world settings.**

Keywords-platform ecosystem; software ecosystem; performance evaluation; analytics; graph algorithms; large language models

I. Introduction

Platform ecosystems are socio-technical networks in which diverse actors such as developers, users, and organizations collaborate and co-create value around a shared technological platform [1]. Examples of such ecosystems include open-source software communities, cloud service marketplaces, and mobile app stores. Assessing and monitoring the health of platform ecosystems presents significant challenges. Unlike traditional software systems, platform ecosystems involve complex interdependencies among actors, diverse contribution patterns, and evolving value-creation models. These characteristics require specialized key performance indicators (KPIs) and health evaluation metrics that capture both technical and socio-economic dimensions [2]. Although several health evaluation models and metrics have been proposed in the literature, they remain fragmented, tool-specific, and often inaccessible to non-technical stakeholders due to lack of attention to ecosystems unique complexities in standard analytics and performance monitoring solutions [3]. As a result, platform orchestrators and decision-makers struggle to gain actionable insights into ecosystem health. This paper presents Prompt-to-Metric, a work-in-progress system designed to address these challenges by enabling natural language access to platform ecosystem health analytics. The system integrates a hierarchical KPI network of approximately 400 health metrics, derived from a systematic literature review, and organizes them across four abstraction levels. By leveraging graph algorithms, Prompt-to-Metric retrieves relevant metrics in response to user queries and uncovers relationships among KPIs to support deeper analytical insights.

An initial prototype has been developed and deployed in a real-world platform ecosystem to evaluate its feasibility. Preliminary findings suggest that Prompt-to-Metric can bridge the gap between complex platform ecosystem health analytics and brevity and accessibility requirements of non-technical stakeholders. The contributions of this study are three-fold. First, it presents a four-tier network-based data model for ecosystem health evaluation metrics. Second, it presents a pipeline for natural language-based platform ecosystem health monitoring. Third, it presents preliminary findings from evaluations in real-world settings. The remainder of this paper is structured as follows. Section II presents details on the KPI network used within the system. Section III presents details on the Prompt-to-Metric pipeline. Section IV presents discussion of the findings from the preliminary evaluation of the system.

II. KPI Network for Platform Ecosystem Health evaluation

A. Health Metrics Elicitation

To elicit the health metrics to be integrated in the system, we conducted a systematic literature review with a focus on data-

Manuscript received July 16, 2025; revised July 26, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.97

driven quantitative metrics and performance indicators of platform ecosystems [4]. The search was executed on three major scientific databases: IEEE Xplore; ACM Digital Library; and Scopus. Application of the inclusion and exclusion criteria, followed by a thorough quality appraisal, resulted in the inclusion of 52 primary studies in the review process after full-text screening and analysis. The reviewed studies presented a mix of empirical analyses, metric proposals, and framework developments targeting ecosystem robustness, resilience, productivity, niche-creation, and evolution. Our review distilled 416 health metrics ranging from low-level activity indicators to abstract ecosystem-level constructs, and varying significantly in abstraction, measurability, and scope. In addition to cataloguing these metrics, we inferred, from the overall approach of the reviewed studies, a structuring of the extracted metrics into a formal, hierarchical graph, serving as both an ontology and a computable model for automated health assessment which was integrated and operationalized through the Prompt-to-Metric system. Figure 1 shows the distribution of extracted metrics across the inferred four levels of abstraction hierarchy elaborated below.

- Level 1: Metrics at this level represent broad strategic categories that constitute a major distinct domain of performance for the ecosystem and frames a different perspective on its health. Examples include technical health; productivity; niche-creation; and network health.
- Level 2: Metrics at this level represent conceptual characteristics and qualities that indicate performance regarding specific goals or capabilities within a domain of performance. While not directly measurable, they guide the formulation of composite indicators. Examples include visibility; scalability; and robustness.
- Level 3: Metrics at this level represent tangible qualities and quantifiable indicators which indicate specific modular performance aspects. These metrics are neither abstract nor directly quantifiable but can rather be estimated by aggregating lower-level directly quantifiable metrics. Examples include developer activeness; communities' growth; contribution satisfaction; and profit focus.
- Level 4: Metrics at this level represent directly measurable and quantifiable performance indicators. Examples include bug fix time; active projects count; cash flow; and network transitivity.

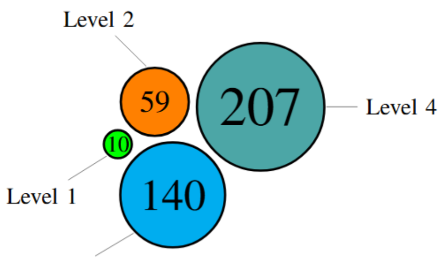


Figure 1. Distribution of health metrics per hierarchy level.

B. Metric Graph Representation

Each metric was modeled as a node in a directed property graph stored in a Neo4j graph database. Edges encode parent-child relationships that reflect conceptual aggregation, dependency, or influence between metrics as suggested by their origin studies. Nodes represent individual metrics, and store metric-specific attributes such as desired direction, quantifiability, measurement unit, among others. Figure 2 illustrates a subsection of this graph with each color representing a different abstraction level. The hierarchical layout allows traversing from abstract ecosystem performance aspects the tangible metrics that quantify them. The graph structure serves both analytical and operational purposes. Within the Prompt-to-Metric pipeline, this graph also acts as the lookup structure for matching user prompts to valid health metrics and their associated computation logic. For example, it enables queries such as "find all measurable indicators that contribute to developer engagement" or "identify all financial metrics associated with ecosystem maturity."

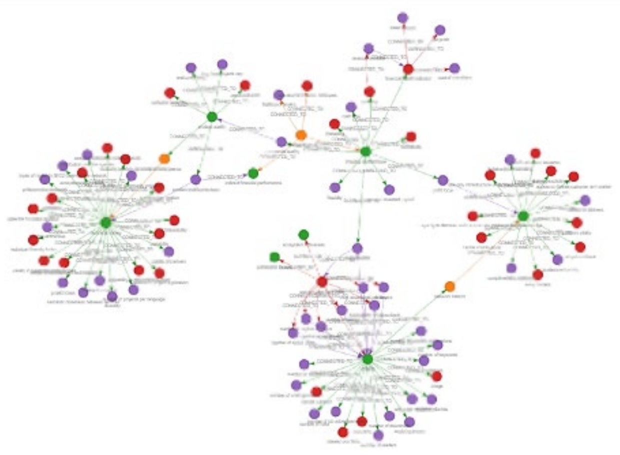


Figure 2. Visualization of a sub-graph of the KPI network.

C. Metric Operationalization and Data Mapping

Prompt2Metric draws on three categories of complementary data sources. Each category is ingested or queried in a way that supports real-time calculation of its associated metrics. Each metric is associated with a calculation script that executed on different data sources according to the metric genre.

- Network health metrics: Data from the platform's services and interfaces are periodically collected, transformed and loaded into a Neo4j graph database according to an integrative schema. This enables the execution of graph algorithms necessary for computing network metrics such as degree centrality, eigenvector centrality, community evolution, among others.
- Technical health metrics: Data from DevOps, version control, bug and issue tracking, and collaborative coding systems of the platform are fetched on-demand

to estimate metrics such as developer activeness, bug-fix time, and similar metrics. Additionally, analytics data of the user-facing interfaces of the platform are utilized for calculating relevant technical metrics such as crash-related metrics.

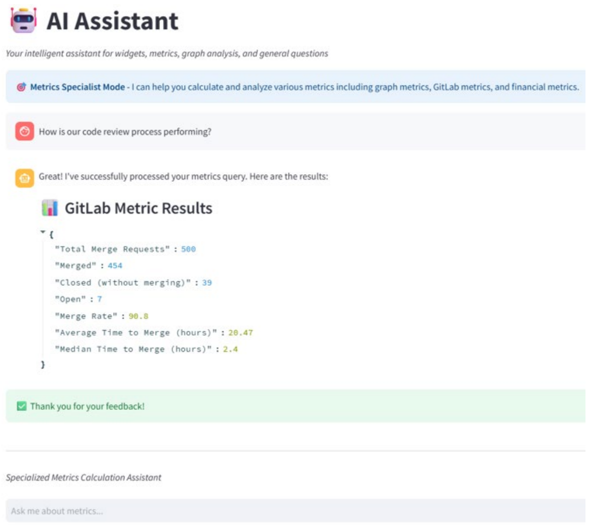
- Financial health metrics: Indicators related to financial health of the ecosystem are computed from figures extracted out of periodic financial statements.

III. Prompt2Metric System

A. System Architecture

The Prompt-to-Metric system comprises five main components:

1) Metrics graph database: As described in Section II.
2) Query-to-Metric mapping engine: A language model-based component that serves as the technical bridge between natural language user inquiries and ecosystem health metrics. This component achieves the following functions.
   - Receiving the user's query from the interface.
   - Accessing the comprehensive list of available metrics.
   - Semantically analyzing the query to identify intent and required metrics.
   - Selecting the most relevant metrics from the metrics database.
   - Executing node centrality and community detection analyses on the metrics database to identify strongly relevant or closely related metrics to the initial list of relevant metrics.
   - Generating the corresponding code for the selected metrics.
3) Metric execution pipeline: A service that achieves the following functions.
   - Generates the concrete data request for the chosen metric, either as a Cypher query (for network data) a GitLab REST call (for technical data), or for other data sources, according to the calculation scripts associated with the selected metrics.
   - Retrieves and executes that request against the corresponding data source.
   - Invokes Neo4j Graph Data Science algorithms when network metrics require centrality or community detection.
   - Post-processes raw results into user-friendly tables or charts and attaches a textual interpretation
   - Logs the prompt, query, runtime and output to our MLflow instance for traceability and evaluation.
   - Streams the formatted response to the Streamlit chat interface.
   - Optionally persists snapshots for longitudinal analysis.
4) User interface: Prompt-to-Metric is delivered through a single-page Streamlit chat application that runs entirely in the browser. All conversational context is kept inside the session state containing the alternating user-assistant message list, the identifier of the last metric served, and a small cache of recently generated Cypher and REST queries. Because this state object is scoped to the browser session, no external store is needed to preserve the context between prompts. UI components include data visualizations widgets, in addition to feedback elicitation buttons. Figure 3 presents a screenshot of the interface after returning metric statistics in response to a query.


AI Assistant

Your intelligent assistant for widgets, metrics, graph analysis, and general questions

Metrics Specialist Mode - I can help you calculate and analyze various metrics including graph metrics, GitLab metrics, and financial metrics.

How is our code review process performing?

Great! I've successfully processed your metrics query. Here are the results:

GitLab Metric Results

```
{
  "Total Merge Requests" : 500
  "Merged" : 454
  "Closed (without merging)" : 39
  "Open" : 7
  "Merge Rate" : 90.8
  "Average Time to Merge (hours)" : 20.47
  "Median Time to Merge (hours)" : 2.4
}
```

Thank you for your feedback!

Specialized Metrics Calculation Assistant

Ask me about metrics...

Figure 3. Prompt-to-Metric *Streamlit Chatbot UI* interface.

5) Evaluation and logging: Every user interaction is tracked by a two–stage feedback pipeline with two components.
   - Automatic run logging: As soon as a workflow is completed, a background thread generates a new MLflow run, which is hosted inside the team's GitLab instance, that records: the raw prompt and timestamp; the selected metric ID and definition; the generated calculation script; execution time; success flag; results row count; and the LLM model version that served the request.
   - Explicit user feedback: If the pipeline completes successfully, the interface displays thumbs-up and thumbs- down buttons. A thumbs-up is logged as feedback = 2, a thumbs-down as feedback = 1. If the pipeline fails before a result is shown, feedback = 0 is logged automatically. The feedback value is appended to the same MLflow run ID in a separate thread so that user experience remains unaffected. This three-point Likert-style scale provides a lightweight but actionable quality signal for longitudinal analysis, regression testing, and future fine-tuning of the LLM components.

B. Workflow

Figure 4 visualizes the end-to-end flow, which unfolds in six steps:

- User prompt: The user submits a natural-language inquiry through the Streamlit chat interface.
- Intent router: An LLM classifier decides whether the prompt requests a health metric or general conversation. Non-metric prompts are handled conversationally; metric prompts advance to Step 3.
- Metric mapper: A second LLM request compares the prompt with all metric descriptions in the unified graph and selects the best-matching metric node. The metric is then analyzed using graph algorithms to suggest closely related or strongly relevant metrics.
- Code generator: The selected metrics calculation scripts are retrieved and adapted to the prompt context. The generated code is executed on the targeted data sources, and the responses are fetched and passed forward for post-processing.
- Result delivery: Raw results are post-processed and displayed in the Streamlit UI and optionally persisted as CSV snapshots.
- MLflow logging: The prompt, chosen metric, generated query, execution metadata, output summary, and user feedback are logged to MLflow for traceability and future evaluation.

IV. Discussion

The preliminary evaluation of the Prompt-to-Metric prototype in a real-world platform ecosystem has provided valuable insights into its feasibility and potential impact. Early deployments with platform orchestrators and technical managers suggest that natural language access to ecosystem health metrics can significantly lower the barrier to understanding complex platform dynamics. Users were able to formulate high-level queries about ecosystem performance and receive actionable responses without requiring prior knowledge of underlying data structures or query languages. The hierarchical KPI network, comprising over 400 metrics organized across four abstraction levels, proved effective in structuring a wide range of health indicators. The integration of graph algorithms enabled the system to identify related metrics and suggest complementary indicators, which was perceived as particularly helpful for exploring unfamiliar dimensions of ecosystem health. However, the evaluation also surfaced key challenges. In particular, financial metrics were found to be difficult to estimate due to data clearance issues, which limited the system's ability to provide a complete view of ecosystem health in some scenarios. Furthermore, certain queries involving large-scale network computations introduced noticeable response latency, and ambiguous prompts sometimes led the language model to select metrics that were technically relevant but not fully aligned withthe user's intent. Despite these challenges, the evaluation

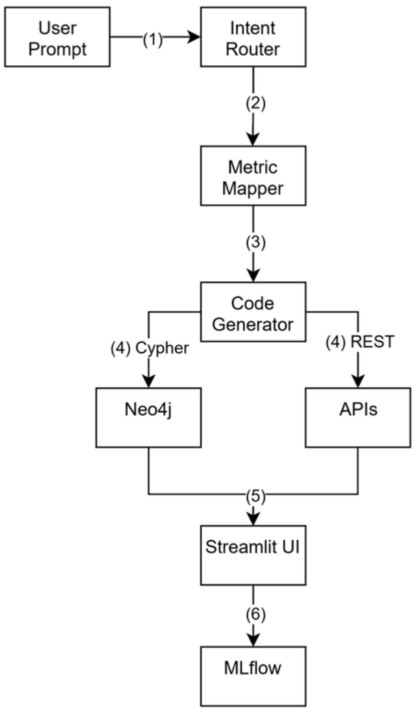


Figure 4. Overview of the Prompt-to-Metric system workflow.

confirmed the system's potential to bridge complex health analytics with the accessibility needs of non-technical stakeholders. Future work will involve expanding the evaluation to additional real-world application scenarios to assess the system's generalizability across diverse ecosystem types. We also plan to incorporate longitudinal analysis of metric time-series data by extending the system's storage to maintain historical values, enabling temporal trend analysis and proactive ecosystem governance. Additionally, an extended usefulness study [5] is planned to be carried out in collaboration with stakeholders and orchestrators of two different platform ecosystems.

References

[1] J. Bosch, "From software product lines to software ecosystems," in *Proceedings of the 13th International Software Product Line Conference*, in SPLC '09. USA: Carnegie Mellon University, Aug. 2009, pp. 111–119.

[2] C. Alves, J. Oliveira, and S. Jansen, *Understanding Governance Mechanisms and Health in Software Ecosystems: A Systematic Literature Review*. 2018. doi: 10.1007/978-3-319-93375-7_24.

[3] F. Fotrousi, S. A. Fricker, M. Fiedler, and F. Le-Gall, "KPIs for Software Ecosystems: A Systematic Mapping Study," in *Software Business. Towards Continuous Value Delivery*, C. Lassenius and K. Smolander, Eds., in Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2014, pp. 194–211. doi: 10.1007/978-3-319-08738-2_14.

[4] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and Durham University Joint Report / Keele University, EBSE 2007-001, Jul. 2007.

[5] F. Davis and F. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, vol. 13, p. 319, Sep. 1989, doi: 10.2307/249008.

CRC: Compressed Reservoir Computing on FPGA via Joint HSIC LASSO-based Pruning and Quantization

Atousa Jafari[1], Hassan Ghasemzadeh Mohammadi[2], and Marco Platzner[1]
[1]*Department of Computer Engineering,* Paderborn University, Germany
[2]*Reneo Group GmbH,* Hamburg, Germany
Email: [1]{atousa.jafari , platzner}@uni-paderborn.de,[2] ghasemzadeh@reno.de

***Abstract*—While reservoir computing (RC) networks offer advantages over traditional recurrent neural net- works in terms of training time and operational cost for time-series applications, deploying them on edge devices still presents significant challenges due to re- source constraints. Network compression, i.e., pruning and quantization, are thus of utmost importance. We propose a Compressed Reservoir Computing (CRC) framework that integrates advanced pruning and quantization techniques to optimize throughput, latency, energy efficiency, and resource utilization for FPGA-based RC accelerators.**

We describe the framework with a focus on HSIC LASSO as a novel pruning method that can capture non-linear dependencies between neurons. We validate our framework with time series classification and regression tasks, for which we generate FPGA accelerators. The accelerators achieve a very high throughput of up to 188 Megasamples/s with a latency of 5.32 ns, while reducing resource utilization by 12× and lowering the energy by 10× compared to a baseline hardware implementation, without compromising accuracy

***Keywords*—Dataflow accelerator, Echo state network, Pruning, Quantization, Time-series application**.

I. Introduction and Background

Reservoir computing (RC) has emerged as a promising alternative to traditional recurrent neural network (RNNs), offering a simpler and more efficient approach to time-series analysis. The most common variant of RC is the Echo State Network (ESN), which consists of an input layer, a reservoir layer, and an output layer as illustrated in Figure 1. The input layer is connected to the neurons in the reservoir layer through randomly generated synaptic connections with weights, modeled as a matrix $\boldsymbol{W_{in}}$. The reservoir layer contains neurons with randomly initialized sparse interconnections, represented by the matrix $\boldsymbol{W_r}$. The reservoir is the core of an ESN, where the feedback connections of neurons together with the non-linearity of their activation functions form a high- dimensional dynamical and non-linear system. The output layer is connected to the reservoir via weighted connections, denoted as $\boldsymbol{W_{out}}$. Unlike standard RNNs, where all layers are trained, an ESN simplifies the training process by randomly initializing and fixing the input and reservoir layers. Only the output layer is trained using basic regression techniques, significantly reducing the computational cost and complexity of training [1]. The reduced effort for training and the rather simple layer structure make RC approaches well suited for time-series tasks on edge devices, particularly for non-linear time series forecasting and time-series classification [2]. This increased network size results in significant computational effort and energy requirements during inference. Thus, effective network compression techniques are studied to reduce the network size and the computational load without compromising performance [3], [4].

For example, network pruning techniques to reduce the number of neurons and connections are discussed in [4]–[6], and quantization for RC models is studied in [2], [7], [8]. In our previous work [9], we presented an approach to map ESN to FPGA hardware as a fully unrolled and quantized dataflow streaming architecture that achieves ultra-low latency and extreme throughput.

Manuscript received May 19, 2025; revised July 30, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.99

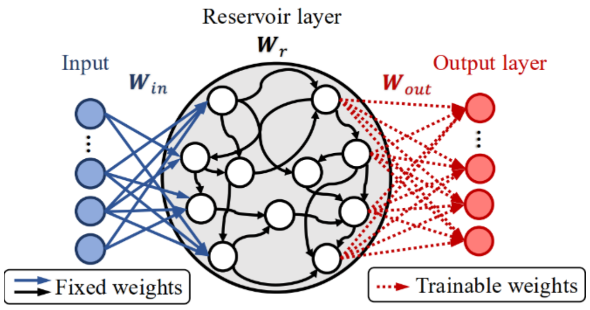


Fig. 1. Layer model of echo state networks (ESN).

In this work-in-progress paper, we propose the feature selection technique HSIC LASSO to be used as a novel pruning algorithm for RC models. HSIC LASSO identifies and removes less important neurons by considering nonlinear correlations within the network, which is a novelty over related work. We present a Compressed Reservoir Computing (CRC) framework for the efficient mapping of RC models to FPGAs, combining pruning and quantization as compression techniques. We experimentally study the effects of pruning and quantization and show that we can reduce the hardware resource requirements up to 12× and decrease the energy by 10× compared to a 32-bit fixed- point baseline hardware implementation.

II. Proposed CRC Framework

Figure 2 depicts the overall flow of our proposed framework for Compressed Reservoir Computing (CRC) on FPGAs. The flow includes four main steps. The first step is *Network Initialization*, where we construct and train an RC model for a given dataset. This step leverages the ReservoirPy framework [10] and includes hyperparameter optimization to achieve the required accuracy. The second step is *HSIC LASSO Pruning*, where we eliminate less significant neurons from the model by considering their non-linear correlations. The third step is *Quantization and Streamlining*, which uses a hardware-friendly streamlining approach to quantize all layers of the RC using the Brevitas framework [11]. The final step is *Direct Logic Implementation*, where we convert the compressed RC model into an FPGA design by mapping all RC layers onto LUT-based structures and creating an RTL (Register Transfer Level) description for the overall design. Subsequently, we synthesize the design into hardware using Xilinx Vivado and evaluate parameters such as the finally achieved accuracy, hardware resource usage, throughput, latency, and power. The remainder of this section details the novel HSIC LASSO-based pruning for RC, followed by an

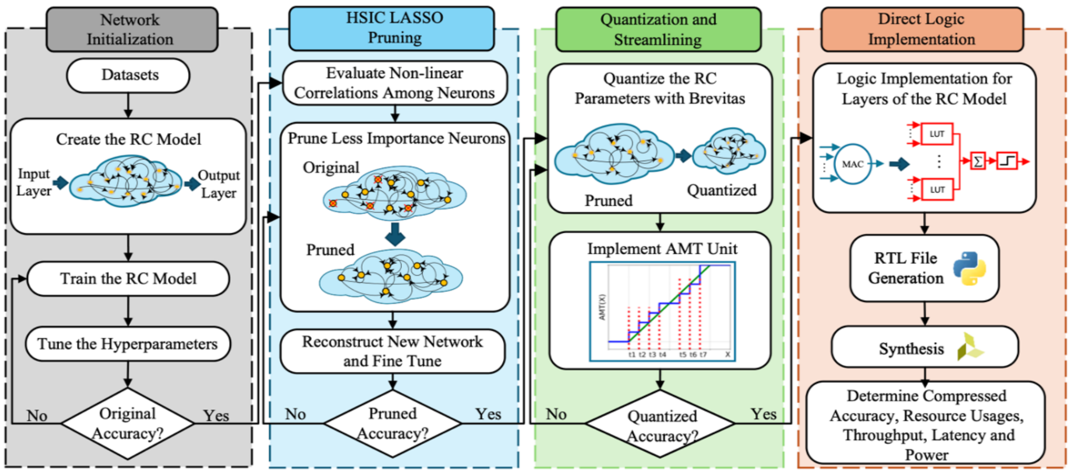

Network Initialization
Datasets
Create the RC Model
Input Layer
Output Layer
Train the RC Model
Tune the Hyperparameters
No
Original Accuracy?
Yes
HSIC LASSO Pruning
Evaluate Non-linear Correlations Among Neurons
Prune Less Importance Neurons
Original
Pruned
Reconstruct New Network and Fine Tune
No
Pruned Accuracy?
Yes
Quantization and Streamlining
Quantize the RC Parameters with Brevitas
Pruned
Quantized
Implement AMT Unit
No
Quantized Accuracy?
Yes
Direct Logic Implementation
Logic Implementation for Layers of the RC Model
MAC
LUT
RTL File Generation
Synthesis
Determine Compressed Accuracy, Resource Usages, Throughput, Latency and Power

Fig. 2. Overview of our proposed CRC framework for FPGA-based implementation.

overview of the streamlining quantization approach. The last two steps of our flow were elaborated in more detail in [9].

A. *Pruning Echo State Networks via HSIC LASSO*

Traditional methods to identify and prune less contributing neurons in an ESN include Spearman [12], PCA [13] and LASSO [13]. Spearman directly assesses a neuron's contribution to accuracy by measuring how well its activity predicts the final output. Keeping a neuron with high correlation helps minimize the prediction error. PCA work indirectly as it ranks neurons based on their contribution to the reservoir's internal dynamic "richness," not to the final output. Hence, PCA provides a superior feature space for the output layer to learn from, thereby reducing the error. LASSO directly ranks neurons by their importance in minimizing prediction errors within a simplified linear model. By forcing the weights of nonessential neurons to zero, it explicitly identifies and removes the neurons that can be ignored with the least impact on accuracy. All this methods are iterative removing neuron by neuron and retraining the network as long as the desired accuracy is retained.

None of the traditional methods, however, captures the nonlinear correlation that exist between the neurons of the reservoir and the reservoir and the neurons of the output layer. HSIC LASSO leverages the Hilbert-Schmidt Independence Criterion (HSIC) to measure nonlinear dependencies, making it particularly suitable for pruning ESNs. In contrast to traditional methods, it provides an efficient alternative by enabling the selection and removal of redundant neurons in a single step, once the hyperparameters are selected. After removal of redundant neurons, the network is retrained.

HSIC LASSO [14] is an extension of the traditional LASSO method, which replaces the mean squared error component of the objective function with a nonlinear dependency measure. The objective function of HSIC LASSO is formulated as:

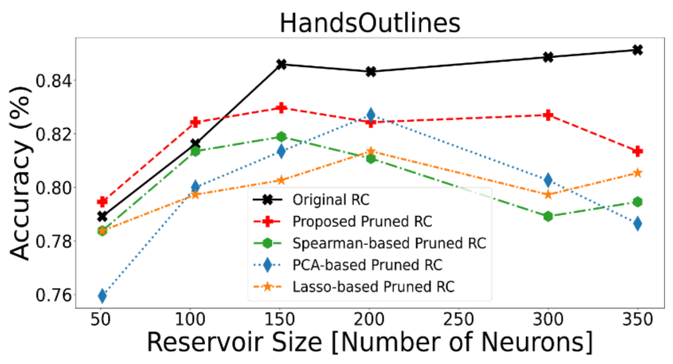
$$argmin\,(\frac{1}{2}\ \ ||R_C - \sum_{i=1}^{n} W_i U_c^{(i)}\,||_F^2 A = \pi r^2 + \lambda||w||_1)$$

$$subject\ to\ w_i\ \geq 0\ \forall i \qquad (1)$$

Where $|.|_F$ denotes the Frobenius norm, and $w\ \in R^n$ in is the weight vector that determines the contribution of each neuron. The matrices $R_c \in R^{d\times d}$ and $U_c^{(i)} \in R^{d\times d}$ are centered Gram matrices of $R_{j,k} = R(y_{i,j}, y_{i,k})$ and $U_{j,k}^{(i)} = U(x_{i,j}, x_{i,k})$, respectively. In this context, the Gram matrix R captures the pairwise similarities between the states of neurons in the reservoir layer. Each element $U_{j,k}^{(i)}$ is computed using a kernel function $U(.,.)$, such as the Gaussian kernel, which measures the similarity between the states of the j-th and k-th neurons at time step i. Similarly, the Gram matrix R captures the pairwise similarities between the network outputs.

B. *Quantization Based on Streamlining Approach.*

We introduce a hardware-friendly quantization approach with the so-called streamline deployment for quantized RC networks. In this method, floating point (FP) operations (e.g., scale and bias parameters) extracted from quantization are absorbed into the activation function for an efficient hardware implementation according to streamline algorithm described in [9], [15]. The quantized activation layer ($HardTanh$ ($Q_{HardTanh}$) is converted into a successive multi-threshold (MT) layer by dividing the range of the activation function into $2^K - 1$ discrete levels, where K is the bit-width of the quantized activation function. The difference between these levels, referred to as the $step$ corresponds to the $Scale$ of the activation function. The input is then compared to these threshold values, and the closest threshold index is selected as the nearest integer. However, the floating-point $Scale$ persists in the process. To eliminate this as well, we divide each threshold by the $Scale$, round it to the nearest integer, and then multiply by the $step$. This method, called absorbing multi-threshold (AMT), absorbs all FP calculations into the successive multi-threshold process.



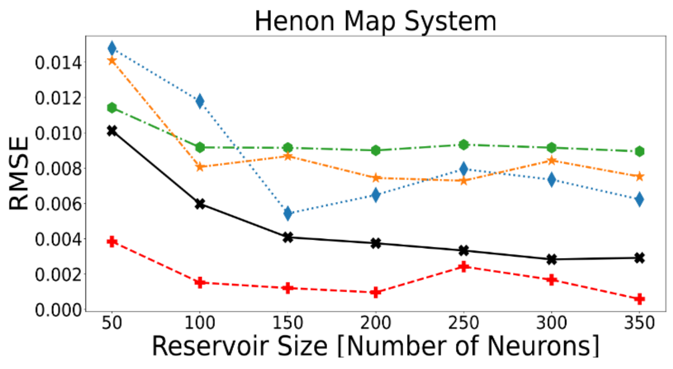


Fig.3. Performance (accuracy, RSME) of the original and pruned networks across reservoir sizes.

To implement the streamlined approach in our framework, we replace the original state update and output equations presented in literature [1] with the updated equations given in Eq. 1 and Eq. 2. In these modified equations, $Q_{u(t)}$, Q_{Win}, $Q_{x(t)}$, Q_{Wr}, and Q_{Wout} denote the integer forms of quantized input, input weight, state and reservoir weight, and output weight, respectively.

$$x(t) = \left(AMT_{Input}\left(Q_{Win} \times Q_{u(t)}\right) + AMT_{Reservoir}\left(Q_{x(t-1)} \times Q_{Wr}\right)\right) \times step \quad (1)$$

$$y(t) = Q_{Wout} \times Q_{x(t)} \times S_{final)} \quad (2)$$

TABEL I: Hardware results for the CRC framework on FPGA (worst-case regression/classification, after pruning).

Accelerator	Network Size	Bit-width (K)	LUTs	FFs	Throughput [Msps]	Latency [ns]	PDP [µWs]
Baseline RC	N_{RC}= 200	32-bit fixed-point	1,629.2K	923.7K	120	8.34	3.31
Quantized RC	N_{RC} = 200	8-bit quantized	21.9K	14.0K	185	5.40	0.65
Pruned RC	N'_{RC} = 149	32-bit fixed-point	228.8K	134.5K	125	7.97	2.34
Compressed RC	N'_{RC} = 149	8-bit quantized	4.6K	2.7K	188	5.32	0.31

TABEL II: PERFORMANCE OF ORIGINAL VS. COMPRESSED 200-NEURON RC MODELS WITH COMPRESSION RATIO

HandsOutlines: Original Accuracy = 83.78%, HSIC LASSO-based Pruned Accuracy = 81.08%, Compression Ratio = **1.8**×
Henon Map System: Original RMSE = 0.003, HSIC LASSO-based Pruned RMSE = 0.002, Compression Ratio = **3.8**×

III. EXPERIMENTAL EVALUATION

A. *EVALUATION OF CRC FRAMEWORKE*

We evaluate our framework on two widely used RC benchmarks: The HandsOutlines dataset as a time-series classification task, and the Henon Map dataset as a regression task for time-series forecasting. Figure3 presents a comprehensive comparative analysis of the ESN model performance, measured as accuracy for HandsOutlines and RMSE (root-mean square error) for Henon Map for the original un-pruned model, models pruned with Spearman, PCA, LASSO, and with our proposed HSIC LASSO-based technique for varying reservoir sizes.

To get a meaningful and fair comparison, we have varied the reservoir size from 50 to 350 and determined the pruned network sizes using the HSIC LASSO technique. Then, we have used the related pruning techniques to reduce the networks to the same sizes. Hence, we compare the pruning techniques at the same compression ratios. The compression ratio is defined in Eq.3, where N_{RC} and N'_{RC} depict the reservoir size in the numbers of neurons for the original and pruned networks.

$$Compression\ Ratio = \frac{N_{RC}}{N'_{RC}} \quad (3)$$

Figure [3] shows that HSIC LASSO performs better than related pruning methods for both classification and regression tasks, the exception being PCA which achieves a slightly better accuracy for HandOutlines with reservoir size 200. For the regression task, our proposed pruning method even achieves lower error than the original un-pruned model, which points to overfitting. As depicted in Table II, HSIC-LASSO can reduce the number of neurons by almost 1.8X and 3.8X for the reservoir layer in the datasets HandsOutlines and Henon Map System with negligible performance losses.

B. *Hardware Implementation for Compressed RC*

Our CRC framework maps the pruned and quantized ESN models to FPGA in the form of a direct logic implementation. All computations are fully unrolled, and weights are hardwired into look-up tables (LUTs) to eliminate memory accesses [16], [17]. Such an approach promises ultra-low latency and extreme- throughput. Since the available logic resources limit the size of the ESN that can be mapped to an FPGA, the approach is targeted towards small and medium sized ESN typically found in edge applications. Further, compression techniques as discussed in this paper are not only vital to achieve efficient hardware resource usage and reduced energy consumption, but also to improve scalability. Our framework generates ESN designs in Verilog, which are then synthesized with Xilinx Vivado 2022.2 to the Virtex Ultra-Scale xcvu19p-fsvb3824-1-e device.

Table I compares the impact of quantization, pruning, and the combination of them (compressed RC) relative to a baseline RC on the metrics hardware utilization, throughput, latency, and Power Delay Product (PDP). The baseline RC is without any pruning, but also a fully unrolled streamlined design with 32-bit fixed-point quantization, and piecewise linear approximation for the activation function. The other quantized designs use 8- bit quantization and the successive multi-threshold approach to realize the activation function. The reported results for latency and PDP are for running a regression/classification on one input vector.

Table I shows that the baseline RC achieves a high throughput of 120 Msps and a latency of 8.34 ns, at rather high hardware costs. By applying the proposed streamline quantization for the same size network, the resource usage drops significantly. By leveraging quantization and pruning (compressed model), it is possible to further reduce hardware costs down to 12× and 8× in LUTs and FFs, respectively, with maximum throughput and minimum latency. The compressed model also achieves the lowest PDP, with 10× improvement over the baseline RC.

IV. CONCLUSION

We have presented a CRC framework that leverages HSIC LASSO-based pruning and hardware-friendly quantization to compress an RC model for efficient FPGA implementation. The compressed direct logic implementation achieves high throughput and ultra-low latency, up to 188 Megasamples/s and 5.32 ns, respectively, and reduce resource utilization by 12× and energy by 10× compared to a baseline hardware implementation.

ACKNOWLEDGMENT

This work is supported by the German Federal Ministry for the Environment, Nature Conservation, Nuclear Safety, and Consumer Protection under grant no. 67KI32004A.

REFERENCES

[1] G. Tanaka et al., “Recent advances in physical reservoir computing: A review,” Neural Networks, vol. 115, pp. 100–123, 2019.

[2] C. Lin et al., “Fpga-based reservoir computing with optimized reservoir node architecture,” in 2022 23rd International Sympo- sium on Quality Electronic Design (ISQED). IEEE, 2022, pp. 1–6.

[3] X. Zhang et al., “Appq-cnn: An adaptive cnns inference accel- erator for synergistically exploiting pruning and quantization based on fpga,” IEEE Transactions on Sustainable Computing, 2024.

[4] H. Wang et al., “Optimizing the echo state network based on mutual information for modeling fed-batch bioprocesses,” Neurocomputing, vol. 225, pp. 111–118, 2017.

[5] D. Li et al., “Structure optimization for echo state network based on contribution,” Tsinghua Science and Technology, vol. 24, no. 1, pp. 97–105, 2018.

[6] J. Huang et al., “Semi-supervised echo state network with partial correlation pruning for time-series variables prediction in industrial processes,” Measurement Science and Technology, vol. 34, no. 9, p. 095106, 2023.

[7] S. Liu et al., “Quantized reservoir computing on edge devices for communication applications,” in 2020 IEEE/ACM Symposium on Edge Computing (SEC), 2020, pp. 445–449.

[8] Y. Abe et al., “Spctre: sparsity-constrained fully-digital reser- voir computing architecture on fpga,” International Journal of Parallel, Emergent and Distributed Systems, vol. 39, no. 2, pp. 197–213, 2024.

[9] A. Jafari et al., “Ultra-low latency and extreme-throughput echo state neural networks on fpga,” in Applied Reconfigurable Computing. Architectures, Tools, and Applications. Springer Nature Switzerland, 2025, pp. 179–195.

[10] N. Trouvain et al., “Reservoirpy: an efficient and user-friendly library to design echo state networks,” in International Confer- ence on Artificial Neural Networks. Springer, 2020, pp. 494– 505.

[11] A. Pappalardo, “Xilinx/brevitas,” 2023. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[12] Z. Huang et al., “Rethinking the pruning criteria for con- volutional neural network,” Advances in Neural Information Processing Systems, vol. 34, pp. 16 305–16 318, 2021.

[13] H. Ghasemzadeh Mohammadi et al., “Efficient statistical pa- rameter selection for nonlinear modeling of process/performance variation,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 12, pp. 1995–2007, 2016.

[14] M. Yamada et al., “High-dimensional feature selection by feature-wise kernelized lasso,” Neural computation, vol. 26, no. 1, pp. 185–207, 2014.

[15] Y. Umuroglu et al., “Streamlined deployment for quantized neural networks,” https://arxiv.org/abs/1709.04060, 2018.

[16] Logicnets: Co-designed neural networks and circuits for extreme-throughput applications,” in *2020 30th Interna- tional Conference on Field-Programmable Logic and Applica- tions (FPL)*, 2020, pp. 291–297.

[17] A. H. Hadipour *et al.*, “A two-stage approximation methodology for efficient dnn hardware implementation,” in *2025 IEEE 28th International Symposium on Design and Diagnostics of Elec- tronic Circuits and Systems (DDECS)*, 2025, pp. 119–122

Towards a Distributed Quantized Machine Learning Inference with Commodity SoC-FPGAs Using FINN

Mathieu Hannoun[1,2]
[1]Laboratoire ETIS, UMR8051, CY Cergy Paris Universités, ENSEA, CNRS, F-95000 Cergy, France
[2]Madicob, 14 Rue du Petit Albi, 95520 Osny, France

Stéphane Zuckerman, Olivier Romain[1]
[1]Laboratoire ETIS, UMR8051, CY Cergy Paris Universités, ENSEA, CNRS, F-95000 Cergy, France

Abstract—**Deep Neural Networks (DNNs) have experienced significant growth over the years, accompanied by a corresponding rise in energy consumption due to their escalating demand for computational resources. To mitigate the environmental impact of AI, and address growing concerns over data privacy, a growing trend is to process data locally at the edge rather than relying on large-scale data centers. FPGA-based systems are particularly suited for this kind of applications, with their low power consumption to high parallel computation ratio. The main drawback of commodity FPGAs is their limited hardware resources, constraining the size of the DNNs which can run efficiently on such targets. This paper presents a methodology for distributed DNNs on multiple commodity FPGAs to support models that are usually only suited for larger FPGAs. We are able to support the inference for a MobileNetV1 on six Zedboards with a peak throughput of 118.3 inferences per second for an estimated power consumption of 16.176 Watts.**

Keywords-*Machine Learning, Deep Learning, CNN, FPGA, Edge Computing*

I. Introduction

The inference of deep neural networks (DNNs) and particularly convolutional neural networks (CNNs) at the edge (through edge and fog nodes) has gained in popularity for its energy efficiency, data locality, and data privacy – all growing concerns [1]. This computation can be done by a variety of hardware, *e.g.*: smartphones [2], [3], microcontrollers [4] and FPGAs, leading to the rise Tiny Machine Learning (TinyML) [5], *i.e.*, the deployment of machine learning models on ultra low-power, resource-constrained devices. Microcontrollers are a possible target, but while they fulfill the need for very low-power consumption, they lack the efficient parallel computation capability of FPGAs, while suffering the same drawbacks of having too few resources to support larger models. Performing the inference of large scale DNNs and CNNs require a massive amount of floating-point operations.

As data privacy is an important factor, implementing a system closer to the data source may help avoid resorting to cloud-based solutions. Yet, high-end FPGA are expensive and have a high absolute power consumption (can be over hundreds of watts). In the context of smart buildings, where heterogeneous low-power devices are abundant, commodity FPGAs offer a very promising trade-off: they yield a low power footprint while being able to take advantage of parallelism in a pre-trained neural network model, thus being able to deliver great performances in the context of edge computing. However, commodity FPGAs offer limited hardware resources, making the implementation of DNNs challenging for models larger than a few binarized layers.

Multiple techniques have been developed in recent years to provide a flexible way to implement DNNs on FPGA, such as FINN [6], from weight compression using quantization, up to using only one or two bits for weights and bias [7]. Operations conversion has also been developed to leverage this level of compression [8], [9]. Few works have tackled distributed inference on FPGAs to support larger networks or to speed up computation. Alonso et al. [10] focused on resource partition and optimization for splitting a MobileNetV1 and a Resnet50 across two high-end FPGA boards. It is based on direct FPGA to FPGA communication with 100 Gbps Ethernet links, using VNx IP cores. While this setup yields very high throughput, it is not suited for low-power IoT devices. Some works have explored such hardware. Notably, Fiscaletti et al. [11] used FINN to split a network of binarized CNN on three Pynq boards, but their model splitting was done manually. Jiang et al. [12] have tackled the implementation of a framework for DNN distributed inference on multiple FPGAs. They use it on two boards to speed up DNN inference, but it does not implement the hardware optimizations used by FINN related to quantized neural networks.

In this paper, we address the problem of how to fit DNNs onto several embedded devices on the same network while leveraging FPGAs suitability for hardware acceleration and their low power consumption. In addition, we are interested in the possibility to substitute a dedicated high-end FPGA for a multiple of networked low-power commodity FPGAs. We propose a way to distribute DNN models inference over several FPGAs when a single board is not sufficient to hold the whole DNN. Communications are handled by the CPU, while the actual inference is done by the FPGA.

II. Methodology

Our approach enables the deployment of DNNs, typically suited for high-end FPGAs (*e.g.*, Alveo U250) due to their size, onto commodity FPGAs by partitioning the model into multiple sub-models. These sub-models are distributed across a network of SoC-FPGAs.

Our methodology provides a flexible and scalable approach to FPGA-based inference, enabling larger models to run on low-

Manuscript received April 30, 2025; revised July 24, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.76

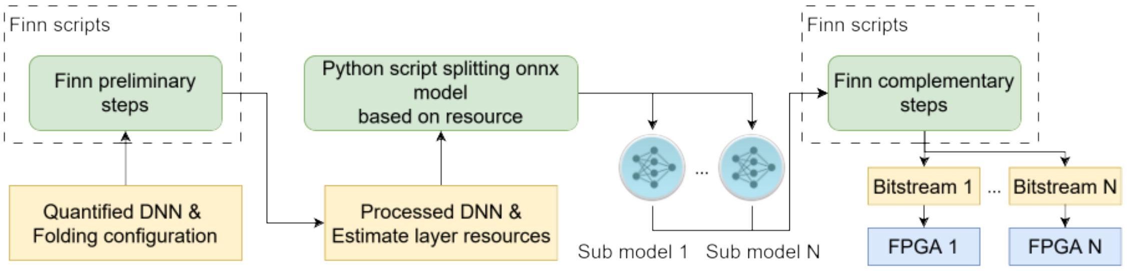

Finn scripts
Finn preliminary steps
Python script splitting onnx model based on resource
Finn scripts
Finn complementary steps
Quantified DNN & Folding configuration
Processed DNN & Estimate layer resources
Sub model 1
Sub model N
Bitstream 1
Bitstream N
FPGA 1
FPGA N

Fig. 1: DNN splitting process. Complete pipeline from quantified DNN to FPGA implementation.

cost hardware. While it does not yet match the performance of high-end FPGAs, it represents a step toward more accessible and efficient distributed inference solutions. Our work uses FINN as a basis for hardware implementation to run inferences. FINN can convert a DNN model to target an FPGA. It will take an ONNX model representing the DNN architecture and its weights, convert it to multiple intermediate representations and output a bitstream file, including the model and custom Direct Memory Access (DMA) engines. However, the intended use is to take a complete model to produce a configuration that will fit in a single FPGA. Our goal is to partition the model into smaller sub-models that can each fit into a commodity FPGA and communicate with each other to complete the overall model. To achieve this, we split the ONNX model representing the DNN, use FINN to generate a bitstream from each sub-model, use those bitstreams to configure each board, and establish communication with every board to run inferences.

Network Splitting Strategy

To allow medium size DNN models to fit on low-power FPGAs, it is necessary to split them to fit resource constraints. FINN [9] is used to automatically generate a preprocessed model and a per-layer resource estimation. The resulting split is based on those models. The objective of our automated splitter is to maximize resource occupancy to reduce the number of bitstreams. Fig. 1 provides an overview of our splitting process. Currently, only the network splitting is automatic. FINN's preliminary and complementary steps are still launched manually.

To find a suitable cut, the ONNX graph is traversed, starting from the first layer. For each node, the required resources estimate (*i.e.*, LUTs, BRAMs, DSPs) are added, until they exceed available resources for a given board. For now, the only objective of our automated splitter is to maximize resource occupancy to reduce the number of bitstream files (in the future, other metrics and objectives may be targeted, such as the amount of data transmitted between layers to reduce network traffic). The original ONNX model is then split into several ONNX submodels. The resulting models are then fed to FINN to generate FPGA design as well as a bitstream for each sub-model. We modified FINN to support the Zedboard for bitstream generation (since we do not use the Pynq OS, we have no need for the Pynq overlay). For now we target identical boards, but in the future we will have different types of SoC-FPGA systems, and some submodel may only fit on specific boards, or alternatively, a given sub-model may be allocated differently resource-wise (to favor resource utilization, limit power consumption, *etc.*) depending on the target hardware for a given submodel. A more complex automatic solver will be required then.

III. Experimental Results

A. Experimental Testbed

The overall system is architected as a pipeline, with each SoC-FPGA board running its own client/server software. The client part sends data to the next board while the server part waits and processes data from the previous board (see Fig. 2). Submodels are generated beforehand, as explained in Section II. Each board embeds all submodels (including Initial weights), stored locally as bitstreams. A client can distribute a DNN on-the-fly by ordering a selected board to reconfigure its Programmable Logic (PL) into the desired submodel.

Each board can be queried independently by the orchestrator. Communications are thus handled by a TCP server written in C++, and running on the Processing System (PS) side, as shown in Fig. 2. The server directly interacts with the PL part to trigger network inferences through the CPU.

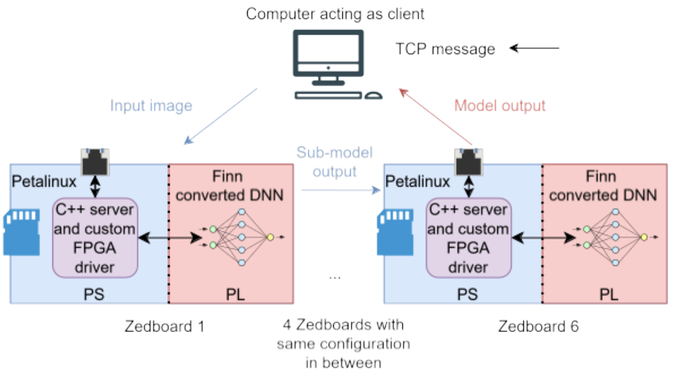

Computer acting as client
TCP message
Input image
Model output
Sub-model output
Petalinux
C++ server and custom FPGA driver
PS
Finn converted DNN
PL
Petalinux
C++ server and custom FPGA driver
PS
Finn converted DNN
PL
Zedboard 1
4 Zedboards with same configuration in between
Zedboard 6

Fig. 2: Example of a full communication for a DNN divided into six bitstreams, from an image input to the model classification (Sub model 6 output), from right to left in a sequential manner.

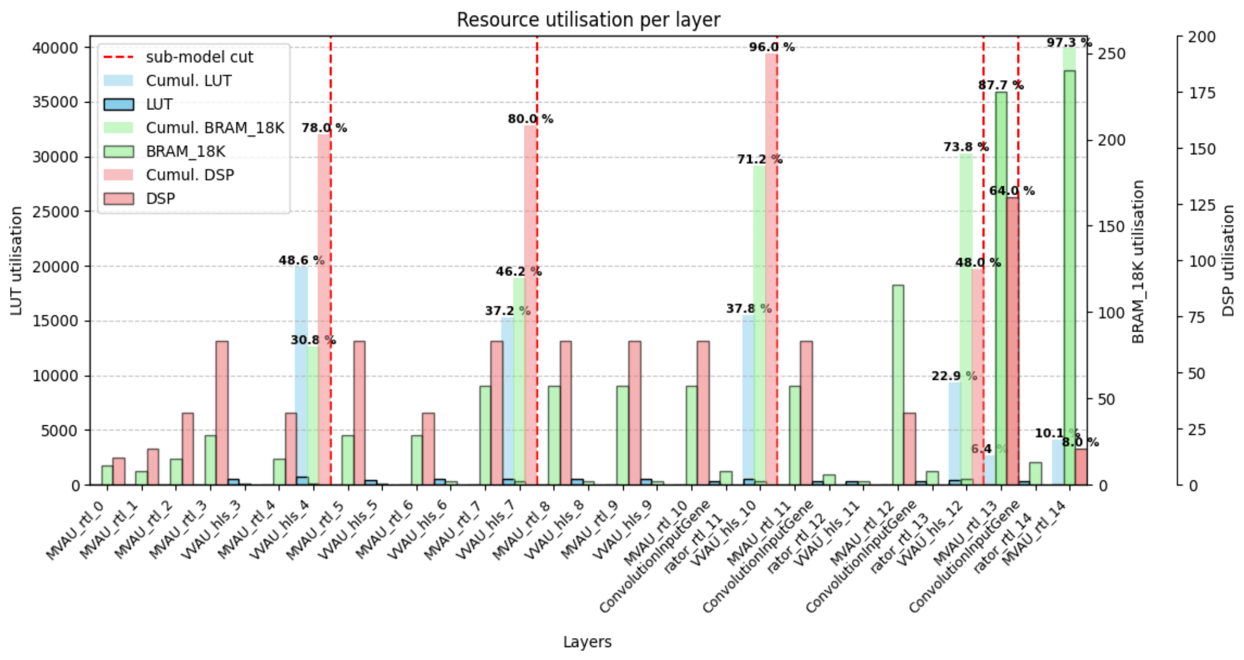


Fig. requ case. rs not in this

DNNs are fully executed on the FPGA fabric, with no contribution from the CPU: the ARM processor is only used for data transfers: network communications, input data buffering, and moving data to and from the PL. It currently runs on two threads, one listening for incoming data and storing them in a queue, while the second pops data items from the queue, passes them to the PL and sends the output to the next board. For this study, our system is comprised of 6 Zedboards (using Zynq XC7Z020) connected to a 1 Gbps Ethernet switch. The Zynq is a heterogeneous system with a dual-core processor (ARM Cortex-A9) coupled with an FPGA (XC7Z020-CLG484-1). Each board is set up with an identical version of Petalinux v2024.1. A laptop acts as an orchestrator for initial configuration and as a client sending data to the first board and receiving model inference output from the last (see Fig. 2). Time measurement has been taken client-side to account for all network transit. Power estimations are given by Vivado after the bitstream synthesis stage. In our case, we take the sum of each PS and PL estimates for each board.

B. *MobileNetV1*

MobileNetV1 [13] is a CNN with 3.22 million parameters. The DNN is quantized to 4 bits for weights and activations. We are using the version made available by Xilinx on their FINN example git repository. It has been trained on the ImageNet dataset [14] using input images with a resolution of 224×224×3 pixels. Using our splitting automation script, it resulted in 6 bitstreams which can each fit in a single XC7Z020. In FINN, the folding configuration is the per-layer selection of processing-element (PE) and SIMD-lane counts that affect the level of parallelization of each layer, trading resource use against throughput. Default folding configuration was used, except for the first and last layer of every submodel where, respectively, SIMD and PE were set to two. This is due to the 4 bits quantization of the MobileNetV1. This allows to pack multiple data words into a single byte at DMA level to avoid having half empty bytes at the sub-model output. This halves the network communication overhead. The number of layers in those six submodels is very disparate (31 layers for the first submodel, only 5 for the last submodel). This is due to the high variability in resource consumption between each layer, as shown in Fig. 3.

TABLE 1 COMPARISON TO EXISTING WORKS ON TINY ML ACCELERATION

Work	Hardware	Model	Number of boards	Peak FPS	Total estimate power (W)	FPS/W
Ours	Zedboard	MobileNetV1	6	118.3	16.176	7.31
[15]	Zynq ZCU104	MobileNetV1 + SSD	1	17.85	N/A	N/A
[10]	Alveo U280	MobileNetV1	1	3741	N/A	N/A
[10]	Alveo U280	MobileNetV1	2	5923	152.3	38.82

The disparity is also present in the type of resources used: For this model, using the default folding configuration, targeting a single Zedboard, the total estimated resources yield 126% of its LUTs, 340% of its DSPs and 378% of its BRAMs.

Fig. 4 breaks down the computation and network cost in time. There we can observe that transfer times vary widely. It is due to the difference in input/output sizes of the submodels (up to 100 kb per inference between submodel 1 and 2). Submodels inferences time also widely vary from 6754 µs for submodel 1, to 3422 µs for submodel 6. This could be leveraged with submodels duplication and parallel processing using more boards.

Using our pipelined architecture we are able to achieve 118.3 peak FPS for a batch size of 1 using our neural network splitting technique. Hence, we achieve a throughput sufficient for real-time video classification while maintaining reasonable power consumption.

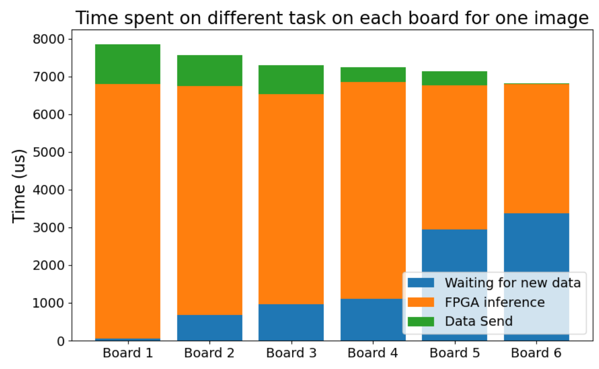


Fig. 4: Average time per task for one inference for each board, waiting/receiving new data, FPGA inference and sending data. In blue, waiting for new data from previous board/client, in orange, data transfer from PS to PL and submodel inference on the FPGA fabric, in green network transfer from the board to the next board/client

IV. Conclusion

In this paper, we have presented a methodology to partition neural networks across several commodity SoC-FPGA systems. This solution supports an arbitrary number of bitstreams and boards. Our first results are promising with a peak throughput of 118.3 inferences per second for MobileNetV1. We stand at a compromise between throughput and power requirements.

It is difficult to compare pure performances between works since most papers use different hardware, slightly different models, may use an object-detection head (SSD) in conjunction, and/or an altogether reworked MobileNet architecture. Our solution is suited to local processing of data with its fairly small power consumption, with a throughput allowing for real-time image processing. We believe it is suitable for environments seeking data privacy and that do not want to rely on cloud services.

Future work includes partitioning larger neural networks to map onto FPGA chips, forming a heterogeneous system of FPGAs to accommodate particularly large layers. This will imply exploring the various tradeoffs to partition such networks, *e.g.*, maximal resource usage per board, the type of resources used (*e.g.*, BRAM vs. LUT-RAM, *etc.*), as well as automatic generation of folding configurations.

References

[1] H. F. Atlam, R. J. Walters, *et al.*, Fog Computing and the Internet of Things: A Review.Big Data and Cognitive Computing. vol. 2, p. 10, June 2018. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[2] H. Li, G. Shou, Y. Hu, *et al.*, “Mobile Edge Computing: Progress and Challenges,” in 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), pp. 83–84, Mar. 2016.

[3] T. Zebin, P. J. Scully, *et al.*, “Design and Implementation of a Convolutional Neural Network on an Edge Computing Smartphone for Human Activity Recognition,” IEEE Access, vol. 7, pp. 133509–133520, 2019.

[4] M. Merenda, C. Porcaro, *et al.*, “Edge Machine Learning for AI-Enabled IoT Devices: A Review,” Sensors, vol. 20, p. 2533, Jan. 2020. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[5] N. N. Alajlan and D. M. Ibrahim, “TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications,” Micromachines, vol. 13, p. 851, June 2022. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[6] Y. Umuroglu, N. J. Fraser, *et al.*, “FINN: A Framework for Fast, Scalable Binarized Neural Network Inference,” in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA ’17, (New York, NY, USA), p. 65–74, Association for Computing Machinery, 2017.

[7] M. Courbariaux, I. Hubara, *et al.*, “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1,” Mar. 2016. arXiv:1602.02830.

[8] M. Rastegari, V. Ordonez, *et al.*, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks,” in Computer Vision – ECCV 2016 (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 525–542, Springer International Publishing, 2016.

[9] M. Blott, T. B. Preußer, *et al.*, “FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks,”ACM Transactions on Reconfigurable Technology and Systems (TRETS),vol. 11, no. 3, pp. 1–23, 2018.

[10] T. Alonso, L. Petrica, *et al.*, “Elastic-df: Scaling performance of dnn inference in fpga clouds through automatic partitioning,” ACM Trans. Reconfigurable Technol. Syst., vol. 15, Dec. 2021.

[11] G. Fiscaletti, M. Speziali, *et al.*, “BNNsplit: Binarized neural networks for embedded distributed FPGA-based computing systems,” in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 975–978, 2020.

[12] W. Jiang, E. H.-M. Sha, *et al.*, “Achieving Super-Linear Speedup across Multi-FPGA for Real-Time DNN Inference,” ACM Trans. Embed. Comput. Syst., vol. 18, pp. 67:167:23, Oct. 2019.

[13] A. G. Howard, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.

[14] J. Deng, W. Dong, *et al.*, “Imagenet: A large-scale hierarchical image database,” in 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255, Ieee, 2009.

[15] A. Sharma, V. Singh, *et al.*, “Implementation of CNN on Zynq based FPGA for Real-time Object Detection,” IEEE Internet of Things Journal, pp. 1–7, July 2019.

Patterns in Design of Microservices Architecture: IT Practitioners' Perspective

Vadim Peczyński, Joanna Szłapczyńska
Faculty of Electronics, Telecommunication and Informatics
Gdansk University of Technology
Gdansk, Poland
vadim.peczynski@pg.edu.pl,
joanna.szlapczynska@pg.edu.pl

Anna Szopińska
Competency Center Digital
Sii Poland
Gdansk, Poland
aszopinska@sii.pl

Abstract—**The literature on Microservices Architecture (MSA) outlines a range of design blueprints as well as certain detrimental practices, reflecting the diverse architectural considerations inherent in MSA design. However, it remains unclear whether and to what extent the practitioners actually adopt the good practices. The study aimed to explore how MSA practitioners apply established patterns and how they address various architectural drivers. The advantages and disadvantages of these approaches were also examined. To achieve this, we conducted a survey on patterns in microservice design among a group of 77 MSA practitioners from IT companies worldwide. The survey shows a need for more accessible and standardised the MSA solutions supporting MSA design phase.**

Keywords-Microservices architecture, MSA, Survey, Patterns, MSA design

I. Introduction

The ideal starting point for a project is to build it with a monolithic architecture, utilising a single database and a single executable that can be easily run on a developer machine [1]. This type of architecture is structured with three primary layers: the client-side user interface, the server-side application, and a database. As the system grows, the maintenance of its architecture is becoming a challenge for developers and architects - all requests must be handled by a single process, and even a minor change triggers the deployment process for the entire application [2]. To overcome these disadvantages, a new type of architecture was introduced - Service-Oriented Architecture (SOA) [3]. SOA is an architecture designed with multiple services that collaborate with each other to provide the final set of functionalities. Each service is using a separate system process and promotes the re-usability of the software. This architecture also gives the possibility of replacing a service with another implementation as long as it keeps the same set of functionalities and communication interface. SOA usually still relies on a single database for the entire system, which ultimately results in the deployment of the entire application, and often uses the SOAP protocol for communication [3].

The Microservices Architecture (MSA) is an evolution of the SOA concept, offering greater independence through loosely coupled, small services that communicate via lightweight mechanisms such as: RESTful API or stream-based communication [4]. Microservices are designed for deployment in cloud environments, where their advantages simplify maintenance, enable autonomous scalability, and support independent deployment [5].

During the design phase of a MSA application, several challenges can arise, which require careful attention to ensure successful implementation. A primary challenge lies in determining the appropriate set of patterns to be employed during the implementation phase. Wrong architecture can lead to tightly coupled services, unnecessary fragmentation, or raising of technical debt [6], [7]. One possible approach involves supporting, balancing, and optimizing the Microservices architecture through the application of an appropriate set of design patterns. The goal of using design patterns in microservices design is to create a solution that satisfies the business's diverse needs while considering the various technical, operational, and financial factors at play.

The scientific literature provides a wealth of analyses and proposals for MSA design. However, it remains unclear to what extent these concepts are actually implemented by MSA practitioners in real world settings. This paper aims to address this gap by exploring the practices of a diverse group of MSA practitioners, primarily IT architects and software developers. We sought to understand the techniques they use for MSA design, the patterns and anti-patterns they apply. To achieve this, we designed and conducted a survey on MSA, involving 77 relevant participants in total from MSA professionals from IT companies around the world.

The rest of the paper is structured as follows. Section Background briefly outlines related literature and surveys on patterns and anti-patterns. Section Method introduces the MSA survey discussed in this publication, presenting the research questions, assumptions, detailing the groups of survey participants and examines potential threats to the validity of the survey. The following section presents results of the survey, categorised into areas: API Gateway, Circuit breaker, discovery mechanisms, transactional messaging, maintaining data consistency, querying and service observability. Section

Manuscript received July 14, 2025; revised September 2, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Regular
DOI: doi.org/10.64552/wipiec.v11i1.101

Discussion interprets and discusses these findings and finally section Conclusions concludes the paper.

II. Related Works

Distributed systems, such as microservices, require a new set of technologies that must be integrated alongside the architecture. To manage initial setup costs, the use of new libraries and design patterns should be kept to a minimum [8]. In [9] the author analyses and describes diversified MSA design patterns applied to different levels of architecture such as communication, database, decomposition, discovery, deployment testing, and observability. In [3] the author extends the previous set of patterns with categories: reliability, scalability and security. Also, [3] proposes patterns focused more on human interaction and UI architecture (e.g. Micro Frontends, Central Aggregating Gateway, Backend for Frontend - BFF). Those patterns promote flexibility and loose coupling to enhance the development of large-scale systems.

Choosing the right set of patterns can be challenging and publications that address this topic can be found in [10], [11]. Also, the research community is increasing its attention around quality attributes (e.g. performance, scalability, security) in Microservices Architecture [4], [12]-[15] and the dependencies between microservices [16].

In [17], the authors also collect information about the usage of design patterns in MSA. They used the Likert scale to describe the use of patterns, and the comparison is discussed in the Discussion section. In [18], the authors propose queueing networks to obtain quantitative insights about seven performance-oriented patterns. Also in [11] the authors analyse the set of 14 design patterns on seven quality attributes during 9 semi-structured interviews. The set of patterns was chosen from the Azure Architecture Center [19].

The industry uses the patterns and strategies to improve the process of implementing the MSA, but many practitioners tend to overlook a critical aspect, the existence of anti-patterns and how they may evolve throughout the various phases of the transition. In [20] the authors describe eight anti-patterns and divide them into two categories: design and implementation. In [21] 19 anti-patterns are described and the research is also extended by adding visualization of these anti-patterns. In [22] the quality model based on 11 anti-patterns is proposed. It shows the need for solving this urgent issue in the form of a decision model lowering the impact of anti-patterns on overall MSA design.

III. Methodology

To guide the study, the research questions were formulated as follows.

- RQ1 - What are the most commonly used design patterns in MSA?
- RQ2 - What patterns are rarely used by practitioners?
- RQ3 - Is data consistency across multiple microservices maintained by design patterns?

To address these questions, we formulated a survey that was conducted among 77 participants from seven countries on three continents: Europe (Poland, Great Britain, Germany, Austria), North America (United States), and Asia (India, Afghanistan). The majority (82%) of the respondents work for companies with more than 1.000 employees. The participants work on the applications from sectors: IT (24%) followed by e-commerce (19%), finance (16%), engineering (11%) and others (30%).

In the survey, 92% of the respondents declared a programming role - out of which 29% are architects, 9% technical leaders and 54% software developers. The other 8% of the respondents are consultants, delivery manager, team leader, engineering manager, software quality (tester), director and chief procurement officer (CPO). The seniority of the participants is as follows: 83% of the respondents declared the level of senior knowledge, 14% declared the regular level of knowledge. Only 3% said they are at the beginning of their professional path (junior).

In the survey we focused on the design patterns commonly used side by side with MSA which can be found in the literature (Tab. 1). Patterns were divided into five groups related with their purpose:

- Communication and reliability,
- Discovery mechanism of Microservices,
- Transactional messaging,
- Maintaining data consistency,
- Observability and monitoring.

The first part of the questions focused on reliability (Circuit Breaker), external API patterns (API gateway) and querying techniques (CQRS, API Composition). These patterns are configured to establish reliable and secure communication with a distributed architecture.

The next part of the survey focused on the discovery mechanism of microservices. This mechanism is the most crucial topic for fault tolerance scenarios [23]. The services that are not working properly must be replaced by new instances and it is a typical action that improves the system's reliability. We asked our respondents if they are using the discovery mechanism in their applications, where the discovery of the services is placed (client-side or server-side), and if they use self- or third-party registration systems.

Transactional messaging was the subject of the next part. Each microservice maintains its own state and has its own database, if needed [3]. Several design patterns were introduced to overcome the problems with data consistency, distributed transactions, and eventual consistency. Transactional outbox (outbox pattern), message relay, and polling publisher are patterns that are responsible for establishing reliable communication between Microservices. Patterns were also added on the database layer where the transaction log miner uses the transaction log (transaction journal) and publishes each change as a message in message broker. We asked about usage of those patterns and which of them are used in the participants'

TABLE I. SYSTEMS. PATTERNS REFERENCE IN LITERATURE

Pattern	Referenced works		
	Richardson[9]	*Newman[3]*	*Newman[8]*
API Gateway	✓	✓	✓
Circuit breaker	✓	✓	
CQRS	✓	✓	
API Composition	✓		
Server-side discovery	✓	✓	
Client-side discovery	✓	✓	
3rd party registration	✓	✓	
Self registration	✓	✓	
Transactional outbox	✓		
Polling publisher	✓		
Transaction log tailing	✓		
Domain event	✓	✓	✓
Aggregate (DDD)	✓	✓	✓
Event sourcing	✓	✓	
Saga	✓	✓	✓
Log aggregation	✓	✓	✓
Application metrics	✓	✓	✓
Audit logging	✓		
Distributed tracing	✓	✓	✓
Exception tracing	✓	✓	
Health checks API	✓		
Log deployments and changes	✓	✓	
Correlation ID	✓	✓	✓

Data consistency in the distributed system is one of the most complex topics [24]. Maintaining distributed transactions when objects are constantly changing must be secure and consistent through all the databases involved in the process. We asked practitioners if they used any patterns to achieve this goal and which of these patterns are implemented in their applications.

The last part of the design patterns section of the survey was focused on observability and monitoring. In huge systems with MSA we need to rely on automation that will bring back all components in the case of any unexpected or faulty behaviour [23]. On the other hand, logs and monitoring services should provide us with documentation of such troublesome behaviour to improve the reliability of the system in the future. Detailed results of the survey are described in the next chapters.

We acknowledge the possible threats to validity related to the research method, the findings, and the strategies that were used to mitigate these threats. They are as follows:

- responses collected can limit their findings - 77 responses were received, the number might be increased if we redo the survey in future works;
- respondents may have different interpretations and understandings of MSA and the design patterns - graphics describing patterns were provided and open option was added in most of the questions to give space also for other answers;
- lack of clarity of the questions - four pilot surveys were conducted with system architects with extensive experience in MSA, language of some of the questions was improved;
- responses from those who were not involved in designing the Microservices systems - by using branching, we closed some of the questions to those respondents without experience in MSA;
- some of the design patterns might have not been mentioned in the survey - open answer was added for any other pattern that was not mentioned.

IV. RESULTS OF THE SURVEY ON MSA PATTERNS AND ANTI-PATTERNS

The study focuses on analysing design patterns commonly used in MSA. The main problems which can be encountered during work with Microservices are: distributed transaction, discovery and reconnection mechanisms, data consistency, and querying.

Communication and reliability. In MSA large monolithic applications are divided into smaller modules (microservices). In this approach the potential points for a cyberattack is bigger, because each Microservice has its own interface for the communication. To mitigate some of the potential risks, the API Gateway pattern was introduced [15]. In addition, it solves problems with cross-platform compatibility and inconsistent issues with microservices call standards [25]. This pattern can also be extended into a Backend for Frontends approach (using multiple API Gateways), which further enhances its versatility. The API Gateway pattern is commonly used in the projects of the respondents (73%). More than a quarter (26%) is not using it in their projects. One participant decided to not answer this question. This pattern's popularity is also evident when examining its usage broken down by project role (Fig. 1).

Circuit breaker is used to improve the resiliency of the MSA. During communication between Microservices Circuit Breaker detects faults and protects the system from cascading failures [26]. It works like a fuse, and when failures consecutively cross the threshold, a circuit breaker will stop the downstream request (open state) for a certain period. After that period, the circuit breaker allows part of the test calls (half-open state) and resumes normal operation (close state) until these calls succeed [4]. The Circuit Breaker is not as commonly used in participant projects in comparison to the API Gateway even though its complexity is compensated by already existing implementations within libraries (e.g. Polly, Resilence4j). It is used in 36% of the projects, 61% declares that they are not using it, and 3% (two participants) did not answer the question.

In MSA the information is scattered between different databases belonging sometimes to hundreds of microservices. In the monolithic architecture, a single database can provide the dedicated views serving data that the user is looking for. One of the challenges in MSA is to handle querying the data across the whole system. One of the potential solutions to this problem is by using API Composition pattern. The pattern provides a simple method to query the data in MSA [17]. The API Composer is a central point of the querying system which knows which microservice endpoint should be called to get the data. Potentially a front-end client could be an API Composer, but due to firewall restrictions and network limitations, it is better to use API Gateway as an API Composer (API Gateway is an internal part of the server solution). This pattern is quite simple and intuitive for querying in MSA. However, it also has its drawbacks such as higher costs of the infrastructure (calling of multiple services each time when data is needed), risk of lower availability (API Composer and all involved microservices need to be available for a query), and potential inconsistencies in transactional data. A more detailed description of this pattern can be found in [9]. The second approach can be CQRS - a pattern that separates read from write operations by querying different databases and keeping them in sync using a dedicated strategy (e.g. Event Sourcing or Relational Database Management System trigger with a special flag to mark data as 'dirty') [10]. This pattern can also be implemented as a single centralised service with dedicated views updated by changes in other databases. The advantages of using CQRS are as follows:

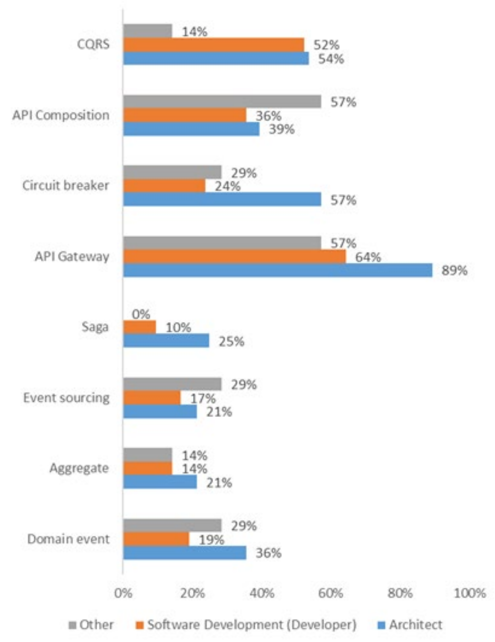


Figure 1. Resilience, communication and data maintenance patterns divided by role in the team among all particpants

- efficient implementation of querying in MSA (one single DB with dedicated views),
- efficient implementation of diverse queries (different databases types can be easily handled),
- can be connected with Event Sourcing,
- improves separation of concerns.

Using CQRS can also have disadvantages related to that:

- system architecture is more complex,
- replication lag needs to be taken into consideration.

A detailed description of this pattern and its advantages and disadvantages can also be found in [9].

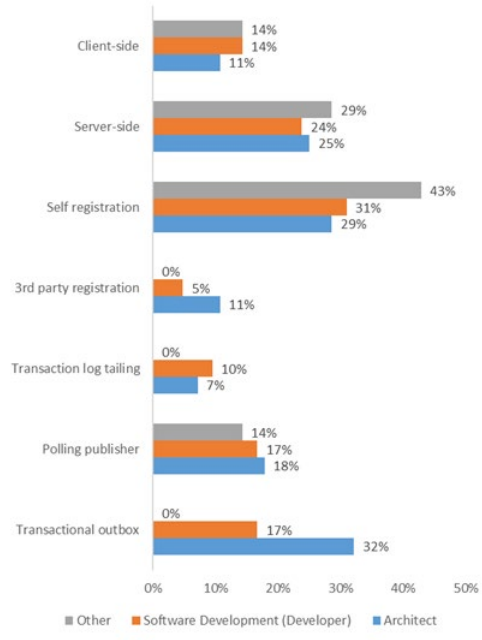
According to survey respondents, CQRS is the most popular approach for querying in the MSA (41%). API Composition is used in 32% participants' projects. There are also respondents who do not use any pattern (23%) and left the answer to this question blank (3%). There is also one other response: "The system uses the REST in communication with the user", which may indicate the usage of API Composition.

Discovery mechanism. The Microservices' environment is very dynamic - virtual machine instances are started and stopped due to failures and scalability features of the MSA. The discovery mechanism uses the Service Registry to store all available instances in the system and helps routing application traffic [23]. The next question was obligatory for all participants and the following two were answered only if the answer was 'Yes' to the first one. The service discovery mechanism is used by 38% of the participants, which is quite low number if we consider the dynamic nature of the MSA - new instances are added to the system when others are shut down within sometimes seconds.

In service discovery, we can use two major approaches: client-side and server-side [9]. Client-side is using Service Registry to get all running instances and using load balancing algorithm (e.g. round-robin or random) is choosing the server which will be used. The main advantage of this approach is the possibility of using multiple platforms (e.g. Kubernetes and an in-house solution with local data centre servers). The disadvantages of this solution are: handling of service discovery mechanisms on client side (especially hard with different technological stack in each microservice), configuration, and maintenance of the service registry as part of MSA. The second approach is to use server-side (platform) discovery. In this approach, the client calls a router, which is load balancing the traffic to all registered services (after querying the service registry). The main advantages are: client code is simpler due to the fact that it does not need to deal with discovery and use one of the available solutions e.g. Azure Load Balancer, Amazon Elastic Load Balancer. The disadvantages are: maintenance of the router (if it is not cloud based), router needs to support the communication protocols (e.g. HTTP/S, gRPC) also more network hops are required in comparison to client-side [27]. The server side is the most popular approach among service discovery users (66%) and the client-side implementation is declared by 34%.

The second part of service discovery is the registration mechanism. It can be implemented in two forms: self-registration and third-party registration [9]. In the self-registration each instance should inform the service registry that it is up and running. The advantage is that each service knows its own state and can give more information than up or down, e.g., starting, available, warm-up [28]. As a disadvantage of this approach we can point: coupling to service registry, each instance needs to implement service registration logic, faulty instance (running, but not able to handle requests) has problem with unregistering from service registry. The second approach – a 3rd party registration - is adding 3rd party registry which is responsible for registering and unregistering a service. Advantages of this approach are the following: the service code is less complex than in self-registration. The registry can also perform periodic health checks. The disadvantages are simplified state knowledge (running or not running) and having another component in the architecture (which sometimes must be additionally installed) [29]. The majority of service discovery users (83%) prefer to use self-registration and other users declare using 3rd party registration (17%).

Service discovery does not appear to be widely adopted also when we analyse it by the different participant roles. While the mechanism is inherently complex to implement [9], its adoption can be significantly simplified by leveraging existing solutions such as Kubernetes, AWS Service Discovery, and Consul. Among those who do use it, self-registration and server-side approaches are the most common, with usage distributed fairly evenly across all three groups (Fig. 2).



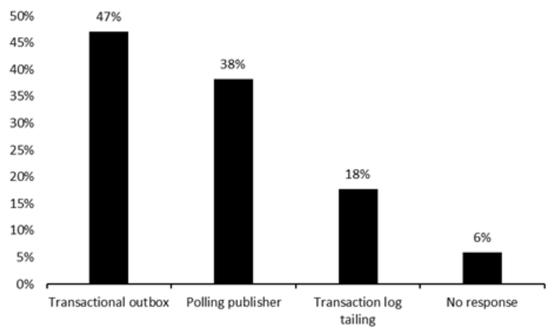


Figure 2. Service discovery and transactional messaging patterns usage divided by role in the team among all participants

Figure 3. Transactional patterns used in participants' projects

Transactional messaging. In MSA each microservice should maintain its own state, and microservices should avoid sharing the database and instead have a database per microservice [3]. This leads to possible problems with data consistency, ACID transactions, and supporting multiple denormalization [17]. Several design patterns were introduced to overcome these issues. One of them is a transactional outbox (outbox pattern) used in databases to store all messages in a table called OUTBOX. The atomicity of the operation is kept due to the fact that the transaction is local. Another pattern is message relay, where messages are read from the table and published to the message broker. The next design pattern is focused on the message moving from the database to the message broker. The polling publisher periodically searches the database for waiting messages and publishes them on the message broker. Finally, the messages are removed from the database. A more sophisticated approach assumes using the transaction log (transaction journal). Each database operation there is stored as an entry in the transaction log. The transaction log miner reads the transaction log and publishes each change as a message in the message broker. This approach can be implemented for relational databases or NoSQL databases. Detailed descriptions of these patterns can be found in [9].

Respondents were asked if they use any kind of transactional messaging pattern. Almost half of the participants (44%) declare that they use these patterns in their projects.

The next question was only available for those participants who answered 'Yes' in the previous question. The respondents were asked which patterns are actually used in their projects, with the possibility of selecting multiple patterns. The most popular pattern is the transactional outbox (47% of 34 responses), but was sometimes not marked together with the polling publisher or the transaction log tailing (47% of the answers), which transactional outbox relies on. The respondents prefer to use the polling publisher (38%) than the transaction log tailing (18%). Two respondents decided not to mark any answer even though there was some other option (6% out of 34 answers). The results are visualised in Fig. 3.

The transactional outbox is used most frequently by architects, which may indicate that it is primarily configured during the initial stages of the project, and that developers may not be aware of its presence in the solution (Fig. 2).

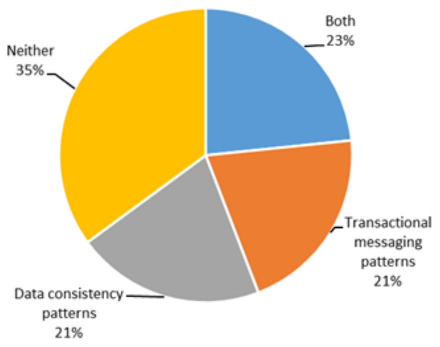


Figure 4. Data consistency and transactional messaging patterns usage in participants' projects

Maintaining data consistency is one of the most crucial challenges that can occur in distributed systems such as MSA. The additional difficulty is also to provide transactions in NoSQL databases that will work side by side with relational databases [30]. Also, eventual consistency - stabilisation of the system after distributed transaction, may cause problems with availability and scalability [24]. Due to these problems, design patterns maintaining data consistency should be introduced. Domain events are used in the Domain-Driven designed systems. They are published when the data is updated and can be consumed by other services. Domain events are often combined with aggregates (Aggregate pattern) that are modelled around one transaction in the system [31]. Aggregates emit domain events when they are created, updated, or deleted. When the process cannot be handled by one single microservice then the Saga pattern is used. Provides a mechanism that ensures the consistency of data between multiple microservices. One of the challenges related to Saga patterns is that they only provide ACD (Atomicity, Consistency, Durability), but without the isolation property [9]. The following pattern that can be used to maintain data consistency is Event Sourcing, in which changes in the application state are stored as sequences of state-changing operations [32]. In Domain-Driven Design (DDD) systems, this pattern can be easily adapted to store the changes of the aggregates, which may give the following benefits:

- the domain events published reliably,
- the history of the aggregates kept,
- facilitated combining of relational and object approaches,
- possibility to be combined with Saga pattern,
- providing access to "time machine" - travelling in history using changes between objects.

From the other side Event Sourcing might be inconvenient due to:

- steep learning curve,
- messaging-based approach which may result in higher complexity,
- evolving and deleting of data more complex than in traditional persistence,
- querying the event store is challenging.

A more detailed description of the advantages and disadvantages of Event Sourcing can be found in [9].

The use of patterns to maintain data consistency by the survey participants' projects is similar to transactional messaging patterns - 44% of respondents (34 participants), but the answer was marked by other participants. After combining the two results, transactional messaging patterns alone are used by 21%, data consistency patterns alone are used also by 21%, 23% are using both and 35% are not using either transactional messaging or data consistency patterns (Fig. 4).

The next questions were only available to those users who answered yes in the question about patterns usage to maintain data consistency. The most common approach for this is to use domain events (59% out of 34 responses). Event Sourcing pattern is used in 44% of the projects of the users of data consistency patterns. Aggregates are used in 38% of the projects, and Saga patterns are used in 32%. Other answers (2 out of 34 answers) are: 'Outbox' and 'Real models with consistency check run by serverless code'. 'Outbox' answer written by the participant is probably referring to the transactional outbox pattern described in the previous section. The results are visualized in Fig. 5.

Observability and monitoring. Each application must provide its Service Level Agreement (SLA), which is the contract between the company and their customers and set forth the expected service parameters [33]. To measure the overall MSA parameters and provide Quality of Service (QoS) metrics, observability patterns were introduced. Observability is often defined as a combination of metrics, logging and tracing [34]. The patterns that can be used for microservice observability are the following:

- Health Check API - exposes the endpoint which gives the information about health of the service often represented as state,

Figure 5. Data consistency patterns used in participants' projects

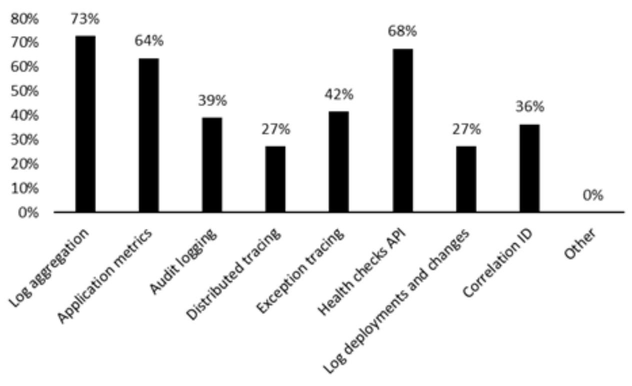


Figure 6. Service observability patterns used in participants' projects

- Log aggregation - centralized logging server which aggregates the information from log service activity and write logs and can provide alerting and searching functionalities,
- Distributed tracking - tracking the flow of the requests between services by assigning each external request an unique ID,
- Exception tracking - each exception is reported to exception tracking service which is de-duplicating an exception, alerts developers and tracks the resolution,

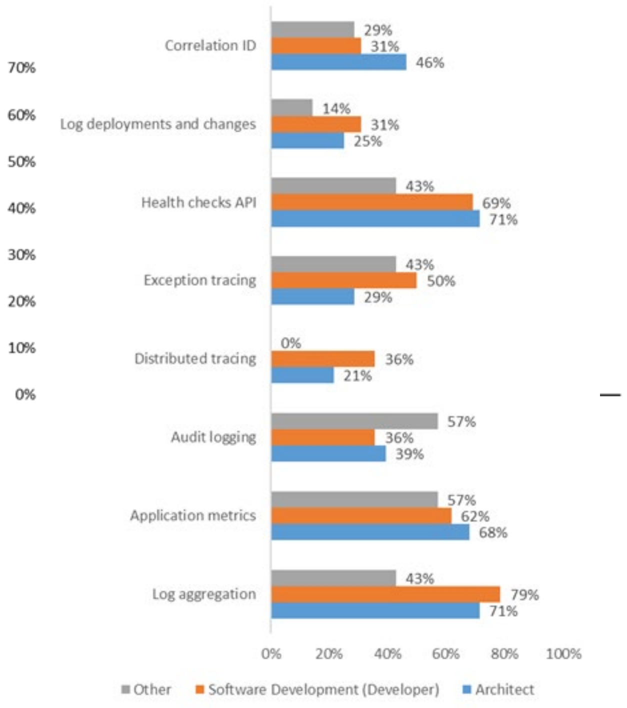


Figure 7. Observability and monitoring patterns usage divided by role in the team among all participants

- Applications metrics - metric server is aggregates the metrics maintained by microservices, such as counter and gauges, and prepare the visualization and alerts,
- Audit logging - records user actions in a database or file and enables searching, ensures compliance and detection of suspicious behaviour,
- Correlation ID - is similar to the distributed tracking, but is also used in queuing and in Saga pattern implementations. [23], [35].

The patterns mentioned above are described in detail in [9].

The survey participants largely declare that they use the service observability patterns (71% in total). In the following question the number of users rise (all the participants could mark one of the patterns) to 92% (only six participants did not mark any of the patterns). The most popular patterns are: log aggregation (73%), health check API (68%) and application metrics (64%). In around half of the projects these patterns are used: exception tracking (42%), audit logging (39%) and correlation ID (36%). Less popular service observability patterns, but still used in one quarter of the projects, are: distributed tracking (27%) and log deployments and changes (27%). The participants declare that in 8% of the projects there are no observability patterns used at all (6 responses without any pattern marked). There were no other patterns mentioned in the answers. The results are visualized in Fig. 6.

An analysis by team role across all participants shows that the most commonly used patterns (Health check API, Application metrics and Log aggregation) are similarly popular among architects, developers, and other roles (Fig. 7).

V. Discussion

During the survey, participants were asked about the design patterns that are used in their projects. Patterns are commonly used to solve recurring types of problems in software architecture [17]. The patterns analysed in the survey can be divided into three main categories: communication patterns, data patterns, and observability patterns.

Communication and reliability. The first two patterns analysed in the survey were API Gateway and Circuit Breaker. The API Gateway pattern is declared to be commonly used by the survey's participants. Implementation of this pattern gives the possibility to expose only a single layer of communication outside and hides the Microservices in the internal network. In contrast, Circuit Breaker is not as popular among participants (only twice as few as API gateway users). As a result, this can weaken the resiliency of the microservices architecture by impairing fault detection and leaving the system vulnerable to cascading failures [26].

The next set of patterns focused on querying the API topic. Participants declare that the CQRS (Command Query Responsibility Segregation) pattern is used more often than the API Composition pattern. CQRS separates read from write operations by querying different databases and keeping them in sync when any changes occur. One of the main benefits of CQRS is its clear separation of responsibilities between commands and queries, which contributes to cleaner, more straightforward, and easier-to-test code. It is a common practice to implement CQRS alongside API Composition, as combining these patterns can enhance system scalability and maintainability by clearly separating read and write concerns while efficiently aggregating data from multiple services. It is very surprising that only one third of the participants declared the usage of API Composition, but two third declared the usage of API gateway (which is one of possible implementations of API Composition). This may suggest a lack of understanding of this pattern among participants.

Service discovery. The next set of patterns focused on the discovery mechanism. Service discovery is used only in less than half of the survey participants' projects. Without this mechanism, the registration must be done manually, which raises the complexity of the final solution. On the other hand, huge complexity of the mechanism implemented from the beginning may lead to problems with deployment of the final solution. The service discovery can be implemented on either the client-side or the server-side. The server-side approach is far more popular among participants. The disadvantages of the server side are: maintenance of the router (if it is not cloud-based), problematic support of multiple protocols. It also generates more hops in the network compared to the client side. The last part of the service discovery is the registration mechanism. The most popular approach among participants is self-registration (each instance has the logic of how to register in a router), which gives them more control over the process.

Transactional messaging. Distributed transaction handling is the problem that is solved by the next group of patterns. These patterns were introduced to overcome the problems with distributed databases (database per microservice) and provide ACID transactions in the Microservices. Only less than half of the participants declare the use of transactional messaging patterns. This may lead to the conclusion that other patterns (e.g. data patterns) may be in use instead.

Maintaining data consistency patterns. The usage of data consistency patterns can be either an alternative or an extension for transactional patterns. Almost half of the survey's participants declared the usage of these patterns. The domain event pattern usage is declared by almost two-thirds of the participants, which may suggest the usage of Domain-Driven Design (DDD) in their projects. The aggregates are also defined in the DDD, but are used by only two thirds of the domain event users. This may lead to problems with the proper decomposition of MSA and maintaining the boundaries of microservices in the future.

Event Sourcing is implemented in less than half of the projects that use data consistency patterns. This pattern has a great benefit of storing the complete story of data changes in the whole system, but it also comes with higher complexity and problems with missing events. Saga pattern is designed to be an alternative for distributed transaction. The implementation of this pattern is highly simplified by dedicated libraries, which expose easy-to-use API and are often free to use. The Saga pattern is declared to be used only in one-third of the data consistency patterns users. The missing implementation of the Saga pattern is not very severe because it can be replaced with, e.g. the outbox pattern, but this pattern also gives the possibility to compensate (revert) the changes and orchestrate the processes. For other patterns, compensation and orchestration must be additionally implemented.

Observability and monitoring. The observability patterns are must-have in modern applications, which can also be found in the results of this survey. The large group (more than two thirds of the participants) declares the usage of these patterns. Log aggregation is declared to be the most popular pattern among the participants but is also often combined with the Health Check API. Health checks provide a quick way to detect when recovery mechanisms need to be triggered, while log aggregation allows for in-depth analysis of the issue and supports implementing improvements to prevent future occurrences. Application metrics are also a widely adopted pattern among participants and are essential for establishing a reliable Service Level Agreement (SLA) with future users.

It is quite surprising that design patterns in general are not as frequently applied in practitioners' projects as we could expect. Thus, it is advisable to design and implement a decision model to support MSA architects in the effective application of these patterns.

Comparison with the other MSA survey. When comparing the results of our survey with the other MSA survey ([17]), we can state that a similar set of design patterns is described as commonly used in Microservices. In [17], the authors used the Likert scale to describe the use of patterns, while in our survey, we simplified the answers to yes/no. In both surveys, the results are comparable; the most popular pattern is the API gateway. Sagas and Circuit breakers are used by one-third of the participants. In our survey, we can find the increase in the use of the CQRS pattern compared to [17]. In that work CQRS usage was described as "sometimes and less", when in our research almost half of the participants declare to use it. In our survey, we also explored the observability patterns (e.g., health checks, exception tracking, correlation ID) and extended patterns found in [17] with application metrics and correlation ID. The usage of application metrics gives the possibility to calculate Quality of Service (QoS) metrics, and correlation ID improves the tracking of messages in the system. Both of those mechanisms are used in the participants' projects. In addition, health checks are declared to be more commonly used in MSA than in [17]. We can find in our results the decrease in the usage of exceptions, which was the second most used pattern in [17]. Throwing of the exceptions is computational consuming and patterns like the Result pattern were introduced to overcome this drawback.

VI. Conclusions

Microservices-based architecture (MSA) provides great flexibility and scalability, making it an excellent choice for most modern, dynamic applications and systems. However, if not designed or implemented correctly, MSA can lead to significant performance bottlenecks, data consistency issues, and security vulnerabilities, among others. Thus, to fully harness the potential of MSA, architects must adhere to patterns that provide guidance on designing, implementing, and managing microservice-based systems effectively. MSA patterns cover a wide range of areas, including service decomposition, communication, resilience, observability, security, consistency, and more.

This paper presents a survey and its findings that illustrate how architects and the IT community nowadays practically engage with MSA, its paradigms, and patterns. It provides an overview of the patterns and techniques defined for and commonly used with MSA. The survey results indicate that architects and developers express a strong demand for patterns that ensure the reliability and security of the system.

The survey results show that the most commonly used pattern in microservices architecture (MSA) design is the API gateway, implemented in 73% of participants' projects. This pattern improves security by providing a single point of exposure to the public network. In contrast, a majority of respondents (62%) indicated that they do not employ service discovery patterns. While these patterns can be complex to implement independently, their adoption may be facilitated by the availability of established libraries and platforms. This omission can reduce the reliability of the system, as new instances must be added manually, increasing the overall complexity of the solution.

The general findings presented may offer valuable insight to architects and developers, highlighting which design patterns are beneficial to adopt in MSA projects.

Acknowledgments

We sincerely thank all the Participants who responded to the survey and contributed to the creation of this article.

Data availability

Data will be made available on request.

References

[1] J. Lewis and M. Fowler, “Microservices - a definition of this new architectural term,” https://martinfowler.com/articles/microservices.html, 2014, accessed: 2025-01-17.

[2] Z. Li, C. Shang, J. Wu, and Y. Li, “Microservice extraction based on knowledge graph from monolithic applications,” Information and Software Technology, vol. 150, p. 106992, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922001240

[3] S. Newman, Building Microservices. O’Reilly Media, 2021. [Online]. Available: https://books.google.pl/books?id=aPM5EAAAQBAJ

[4] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar, “Understanding and addressing quality attributes of microservices architecture: A systematic literature review,” Information and Software Technology, vol. 131, p. 106449, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301993

[5] L. Qian, J. Li, X. He, R. Gu, J. Shao, and Y. Lu, “Microservice extraction using graph deep clustering based on dual view fusion,” Information and Software Technology, vol. 158, p. 107171, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584923000253

[6] V. Lenarduzzi, F. Lomio, N. Saarimaki, and D. Taibi, “Does migrating a monolithic system to microservices decrease the technical debt?” Journal of Systems and Software, vol. 169, p. 110710, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220301539

[7] S. S. de Toledo, A. Martini, and D. I. Sjøberg, “Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study,” Journal of Systems and Software, vol. 177, p. 110968, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221000650

[8] S. Newman, Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O’Reilly Media, Incorporated, 2019. [Online]. Available: https://books.google.pl/books?id=iul3wQEACAAJ

[9] C. Richardson, Microservices Patterns: With examples in Java. Manning, 2018. [Online]. Available: https://books.google.pl/books?id=UeK1swEACAAJ

[10] W. Meijer, C. Trubiani, and A. Aleti, “Experimental evaluation of architectural software performance design patterns in microservices,” Journal of Systems and Software, vol. 218, p. 112183, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121224002279

[11] G. Vale, F. F. Correia, E. M. Guerra, T. de Oliveira Rosa, J. Fritzsch, and J. Bogner, “Designing microservice systems using patterns: An empirical study on quality trade-offs,” in 2022 IEEE 19th International Conference on Software Architecture (ICSA), March 2022, pp. 69–79.

[12] X. Zhou, S. Li, L. Cao, H. Zhang, Z. Jia, C. Zhong, Z. Shan, and M. A. Babar, “Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry,” Journal of Systems and Software, vol. 195, p. 111521, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222001972

[13] S. Henning and W. Hasselbring, “Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud,” Journal of Systems and Software, vol. 208, p. 111879, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223002741

[14] A. Hannousse and S. Yahiouche, "Securing microservices and microservice architectures: A systematic mapping study," Computer Science Review, vol. 41, p. 100415, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000551

[15] M. Matias, E. Ferreira, N. Mateus-Coelho, and L. Ferreira, "Enhancing effectiveness and security in microservices architecture," Procedia Computer Science, vol. 239, pp. 2260–2269, 2024, cENTERIS – International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement/ HCist - International Conference on Health and Social Care Information Systems and Technologies 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924016612

[16] A. S. Abdelfattah, T. Cerny, M. S. H. Chy, M. A. Uddin, S. Perry, C. Brown, L. Goodrich, M. Hurtado, M. Hassan, Y. Cai, and R. Kazman, "Multivocal study on microservice dependencies," Journal of Systems and Software, vol. 222, p. 112334, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121225000020

[17] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Marquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," Journal of Systems and Software, vol. 182, p. 111061, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221001588

[18] R. Pinciroli, A. Aleti, and C. Trubiani, "Performance modeling and analysis of design patterns for microservice systems," in 2023 IEEE 20th International Conference on Software Architecture (ICSA), 2023, pp. 35–46.

[19] Microsoft, "Cloud design patterns," https://learn.microsoft.com/en-us/azure/architecture/patterns/, 2025, accessed: 2025-05-10.

[20] H. Farsi, D. Allaki, A. En-nouaary, and M. Dahchour, "Dealing with anti-patterns when migrating from monoliths to microservices: Challenges and research directions," in 2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), Nov 2023, pp. 1–8.

[21] G. Parker, S. Kim, A. A. Maruf, T. Cerny, K. Frajtak, P. Tisnovsky, and D. Taibi, "Visualizing anti-patterns in microservices at runtime: A systematic mapping study," IEEE Access, vol. 11, pp. 4434–4442, 2023.

[22] S. Pulnil and T. Senivongse, "A microservices quality model based on microservices anti-patterns," in 2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE), June 2022, pp. 1–6.

[23] I. Karabey Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, "Deployment and communication patterns in microservice architectures: A systematic literature review," Journal of Systems and Software, vol. 180, p. 111014, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221001114

[24] X. Zhao and P. Haller, "Replicated data types that unify eventual consistency and observable atomic consistency," Journal of Logical and Algebraic Methods in Programming, vol. 114, p. 100561, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352220820300468

[25] X. Zuo, Y. Su, Q. Wang, and Y. Xie, "An api gateway design strategy optimized for persistence and coupling," Advances in Engineering Software, vol. 148, p. 102878, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965997820304452

[26] C. Lira, E. Batista, F. C. Delicato, and C. Prazeres, "Architecture for iot applications based on reactive microservices: A performance evaluation," Future Generation Computer Systems, vol. 145, pp. 223–238, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X23001036

[27] C. Richardson, "Pattern: Server-side service discovery," https://microservices.io/patterns/server-side-discovery.html, 2024, accessed: 2024-12-06.

[28] C. Richardson, "Pattern: Self registration," https://microservices.io/patterns/self-registration.html, 2024, accessed: 2024-12-07.

[29] C. Richardson, "Pattern: 3rd party registration," https://microservices.io/patterns/3rd-party-registration.html, 2024, accessed: 2024-12-07.

[30] M. T. Gonzalez-Aparicio, M. Younas, J. Tuya, and R. Casado, "A transaction platform for microservices-based big data systems," Simulation Modelling Practice and Theory, vol. 123, p. 102709, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X22001782

[31] V. Vernon, Domain-driven Design Distilled. Addison-Wesley, 2016. [Online]. Available: https://books.google.pl/books?id=h0u7jwEACAAJ

[32] S. Lima, J. Correia, F. Araujo, and J. Cardoso, "Improving observability in event sourcing systems," Journal of Systems and Software, vol. 181, p. 111015, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221001126

[33] O. Ghandour, S. El Kafhali, and M. Hanini, "Adaptive workload management in cloud computing for service level agreements compliance and resource optimization," Computers and Electrical Engineering, vol. 120, p. 109712, 2024. [Online]. Available:https://www.sciencedirect.com/science/article/pii/S0045790624006396

[34] J. Kosińska, B. Baliś, M. Konieczny, M. Malawski, and S. Zielińnski,"Toward the observability of cloud-native applications: The overview of the state-of-the-art," IEEE Access, vol. 11, pp. 73 036–73 052, 2023.

[35] S. Janapati, "Distributed logging architecture for microservices,"https://dzone.com/articles/distributed-logging-architecture-for-microservices, 2017, accessed: 2024-12-27

Optimal Graph Model Schema Injection for Large Language Model-Generated Cypher Queries

Shady Hegazy, Nouman Nusrallah, Christoph Elsner
Siemens Foundational Technologies
Siemens AG
Munich, Germany
Firstname.lastname@siemens.com

Jan Bosch
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden
Jan.bosch@chalmers.se

Helena Holmström-Olsson
Department of Computer Science and Media Technology
Malmö University
Malmö, Sweden
Helena.holmstrom.olsson@mau.se

Abstract—**Platform ecosystems have transformed the way value is created in different industries. The data traces of such ecosystems are typically represented through graph models and databases. Retrieval of relevant data from such databases requires writing extensively complex queries to travers such complex networks to fetch and slice the correct sub-graphs corresponding to the original business inquiry. Advances in generative artificial intelligence, namely large language models (LLMs), can provide a no-code interface to such complex databases by generating and executing database queries that fetch the correct and relevant data in response to user prompts and inquiries. However, for the LLM to generate the right query, data about the schema of the database and the underlying graph model must be provided. In this study, we present a pipeline for evaluating different techniques for injecting the database schema in the LLM prompts, in addition to preliminary evaluation results.**

Keywords-graph database; software ecosystem; large language models; graph algorithms

I. Introduction

Platform ecosystems have fundamentally transformed the way value is created and distributed across industries [1]. By enabling diverse actors such as developers, organizations, and users to co-create and exchange value around a shared technological platform, these ecosystems have become critical enablers of innovation and economic growth. The increasing digitization of platform activities has led to the generation of rich data traces, which are often represented using graph-based models and stored in graph databases [2]. These representations capture the intricate relationships among ecosystem participants, resources, and interactions, offering a powerful basis for analyzing ecosystem dynamics [3]. Despite the expressive power of graph databases, retrieving relevant data from them remains a technically challenging task. Business inquiries that require traversing complex networks and extracting specific sub-graphs often demand the formulation of sophisticated query languages such as Cypher or Gremlin. Writing such queries is not only time-consuming but also requires significant technical expertise, which limits access for non-technical stakeholders such as platform managers and decision-makers. This barrier hampers the ability of organizations to derive timely and actionable insights from their ecosystem data. Recent advances in generative artificial intelligence, particularly large language models (LLMs), offer a promising opportunity to bridge this gap. LLMs have demonstrated remarkable capabilities in understanding natural language and generating structured outputs, enabling the development of no-code interfaces for complex systems. By translating natural language inquiries into graph database queries, LLMs have the potential to make graph-based ecosystem analytics accessible to a broader range of users. However, enabling LLMs to generate accurate and relevant queries for graph databases presents unique challenges. Crucially, the model requires contextual information about the database schema and the underlying graph structure to formulate correct queries. Without this knowledge, even highly capable models often produce incomplete or invalid outputs.

This paper presents a pipeline for evaluating techniques to inject database schema information into LLM prompts to improve their ability to generate correct graph queries. We describe our approach for schema representation and prompt construction, as well as a set of experiments comparing different injection techniques. Preliminary results from these experiments demonstrate the impact of schema injection on query accuracy and provide insights into the design of LLM-driven interfaces for graph databases.

II. Methodology

This section presents the end-to-end workflow for transforming platform ecosystem data into a graph representation and enabling natural language to Cypher query translation

Manuscript received July 16, 2025; revised July 26, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.102

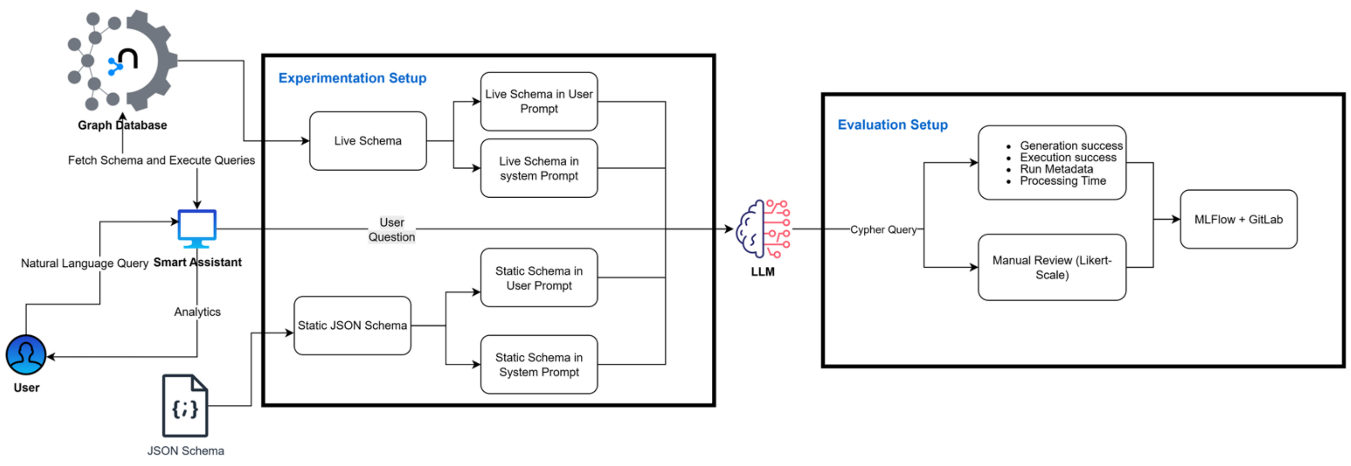


Figure 1. Architecture overview of the experimentation and evaluation pipeline including graph database integration with the Smart Assistant for natural language to Cypher queries.

using a Large Language Model (LLM). Figure 1 outlines the architecture overview of the workflow for the experimentation and evaluation pipeline.

A. Data Extraction and Graph Construction

Structured SECO data is collected from different data sources. The data covers entities such as products, owners, maintainers, visitors, devices, and usage logs. The sources cover different types of data including DevOps, financial, analytics data. An ETL pipeline was created and executed daily. The pipeline consists of the following steps:

- Extract: Connects to APIs, scrapes relevant data, and retrieves documentation and metadata.
- Transform: Drops irrelevant fields, merges multi-source records, parses URLs, and validates entity relationships.
- Load: Loads cleaned data into a Neo4j property graph model. Historical schema snapshots are stored for reproducibility.

B. Generative AI Query Translation Pipeline

The Smart Assistant enables natural language interaction for analytics. The system explores schema injection modes varying along the following variables:

- Schema Source: Either a live schema fetched from Neo4j via the APOC library, which pulls complete metadata of the graph, or the static JSON schema definition file.
- Prompt Placement: Schema injected into either the system prompt or the user prompt.

For each query, the pipeline: Fetches or loads the schema; constructs system and user prompts accordingly; submits the prompt pair to the LLM; parses the LLM response to extract Cypher queries; and executes queries on the graph and returns results.

C. Graph Schema Model

The SECO graph schema includes nodes for Visitors, Devices, Departments, Companies, and APIs, connected through relationships such as `MADE`, `USED`, `PART_OF`, and `ASSOCIATED_WITH`. Figure 2 shows a version of the graph data model schema.

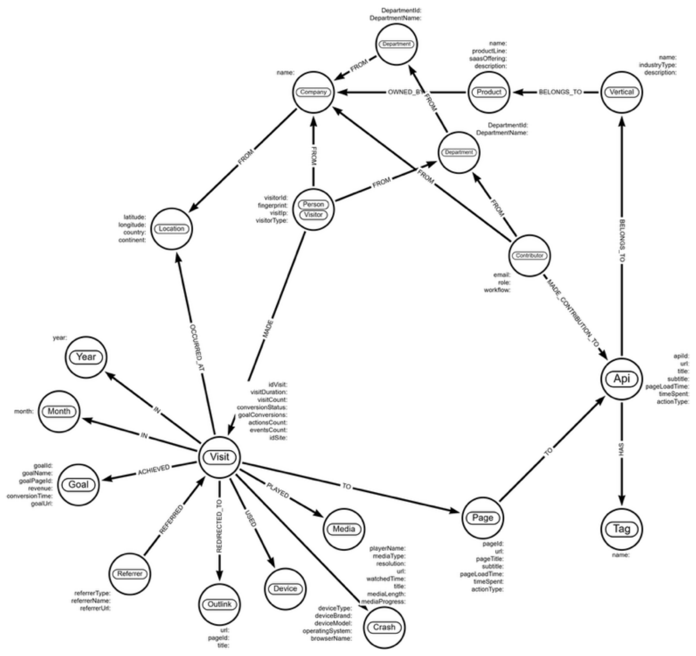


Figure 2. Graph database schema highlighting nodes and relationships.

D. Feedback and Evaluation Workflow

The entire experiment is tracked with MLflow for prompt versions, model runs, and metrics, as shown in Figure 3. GitLab CI/CD pipelines automate data preparation, schema snapshotting, and test coverage for each configuration. Additionally, A human-in-the-loop workflow is integrated and comprises the following steps:

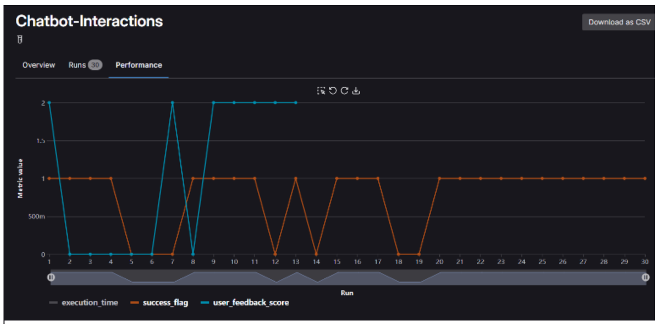


Figure 3. Performance logging of chatbot interactions in MLflow.

- Reviewers assess Cypher generation correctness and logical validity.
- Each query is rated on a 5-point Likert scale, with 1 indicating incorrect/irrelevant output and 5 indicating fully correct and usable queries, as shown in Table I.

TABLE I
LIKERT SCALE FOR HUMAN EVALUATION

Score	Interpretation
1	Incorrect or irrelevant Cypher
2	Major logical errors
3	Partially correct, needs edits
4	Mostly correct, minor edits
5	Perfectly correct and usable

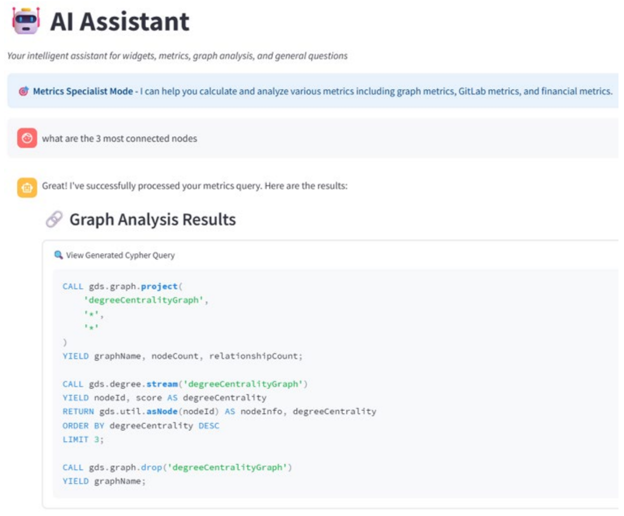
E. *Chatbot User Interface*

A user interface was developed based on Streamlit Chatbot UI framework, as shown in Figure 4. It includes the following aspects:

- Session storage: Each session logs user prompts, schema context, generated Cypher, execution results, and feed- back.
- Results persistence: Users can iteratively refine queries and view results in real-time. Previously generated visualizations are also persisted for further refinements.
- Historical tracking: Sessions are stored for later review and model improvement.
- Feedback elicitation: A widget is embedded in each response to allow for feedback elicitation enabling human-in-the-loop evaluation.

III. DISCUSSION

A preliminary evaluation of the different setups was carried out by executing a set of ten representative queries under the four experimental configurations. Human evaluation by domain experts were carried out for the 40 test cases. The 5-point Likert scale responses were aggregated so that it results in either a success or failure flag to facilitate comparison. The results suggest that including the database schema in the system prompt achieves higher consistency compared to embedding it in the user prompt. Additionally, using a well- defined static JSON schema generally performs better than fetching the schema live using the database own algorithmic functions. Simple entity-


AI Assistant

Your intelligent assistant for widgets, metrics, graph analysis, and general questions

Metrics Specialist Mode - I can help you calculate and analyze various metrics including graph metrics, GitLab metrics, and financial metrics.

what are the 3 most connected nodes

Great! I've successfully processed your metrics query. Here are the results:

Graph Analysis Results

View Generated Cypher Query

```
CALL gds.graph.project(
    'degreeCentralityGraph',
    '*',
    '*'
)
YIELD graphName, nodeCount, relationshipCount;

CALL gds.degree.stream('degreeCentralityGraph')
YIELD nodeId, score AS degreeCentrality
RETURN gds.util.asNode(nodeId) AS nodeInfo, degreeCentrality
ORDER BY degreeCentrality DESC
LIMIT 3;

CALL gds.graph.drop('degreeCentralityGraph')
YIELD graphName;
```

Figure 4. Snapshot of the user interface.

relation queries, such as filtering visitors by device brand or counting unique users, achieved near-perfect success across all configurations. For instance, the query *"List all Visitors who use a specific Device brand, like 'Apple'."* returned valid Cypher queries and expected record counts in all scenarios. In contrast, complex multi-hop or community-matching queries demonstrated higher variance. For example, the query *"Show all Companies with Departments that belong to the same community as Visitor"* succeeded when using the JSON schema but failed in all live schema scenarios. This indicates that more complicated traversal logic is sensitive to prompt placement and schema representation. Approximately 20% of test cases failed, primarily due to incomplete subgraph pattern matching or empty result sets when complex conditions were involved. Failures were more common in the JSON schema with user prompt scenario for temporal-spatial queries, such as "List all unique Visitors from Country X who made Visits in August 2024, along with the device types they used." On average, the query generation time per NLQ remained under 5 seconds, while Cypher execution times were acceptable for interactive analytics workloads. All logs were stored in structured CSV files to enable further analysis and reproducibility. These findings highlight the need for prompt engineering and potential domain fine-tuning to handle edge cases more reliably. Future work will expand the query set, integrate additional LLM models, and explore retrieval-augmented generation.

REFERENCES

[1] A. Hein *et al.*, "Digital platform ecosystems," *Electron Markets*, vol. 30, no. 1, pp. 87–98, Mar. 2020, doi: 10.1007/s12525-019-00377-4.

[2] F. von Briel and P. Davidsson, "Digital platforms and network effects: Using digital nudges for growth hacking," in *Int. Conf. Inf. Syst., ICIS*, Association for Information Systems, 2019.

[3] P. Boldi and G. Gousios, "Fine-Grained Network Analysis for Modern Software Ecosystems," *ACM Trans. Internet Technol.*, vol. 21, no. 1, 2021, doi: 10.1145/3418209.

Authors Index

Supported by:

ISSN 2980-7298

9 772980 729004