

SAMpLE: A SystemC-AMS Machine LEarning-based Framework for Virtual Prototyping

Original

SAMpLE: A SystemC-AMS Machine LEarning-based Framework for Virtual Prototyping / Albu, A.M., Vinco, S.. - ELETTRONICO. - (In corso di stampa). (Forum on specification and Design Languages (FDL) Roma 09/09/2026-11/09/2026).

Availability:

This version is available at: 11583/3016055 since: 2026-09-29T11:32:05Z

Publisher:

IEEE

Published

DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

SAMpLE: A SystemC-AMS Machine Learning-based Framework for Virtual Prototyping

Andrei Mihai Albu and Sara Vinco

Politecnico di Torino, Italy – Emails: andrei.albu@polito.it, sara.vinco@polito.it

Abstract—Machine Learning (ML) is increasingly used in virtual prototypes of embedded systems to model behaviors that are difficult to capture analytically. However, integrating ML models into virtual platform simulation is still typically done through ad hoc solutions, which limits reuse, comparability, and reproducibility. This paper presents *SAMpLE*, an open-source SystemC-AMS-based framework that integrates ML models as first-class Timed Dataflow (TDF) components through a standardized plug-and-play interface. *SAMpLE* provides two execution backends: a native C++ backend for online training of lightweight models, and an offline backend for executing externally developed models without requiring re-implementation in C++ or manual integration steps. The framework uses ONNX as a standard model exchange format to enable integration of externally trained ML models into SystemC-AMS simulations, and allows the evaluation of different ML-based solutions within the same testbench, dataset, and simulation workflow. The modular design and unified and reproducible environment will allow future extensions of *SAMpLE* to new models, without modifying the SystemC-AMS structure.

Index Terms—SystemC-AMS, Virtual Prototyping, Machine Learning, Timed Dataflow (TDF), Hardware/Software Co-Design.

I. INTRODUCTION

Machine Learning (ML) techniques are increasingly adopted in embedded-system design to approximate complex behaviours that are difficult to capture with analytical models or traditional simulation approaches, including power consumption, sensor dynamics, ageing effects, and fault prediction [1]–[3]. Compared to conventional modeling techniques, ML-based surrogates provide a favourable trade-off between accuracy, computational cost, and data-driven adaptability, making them increasingly relevant in system-level virtual prototyping.

Despite this trend, virtual platform languages, such as SystemC and its extensions, do not provide a native or standardized mechanism for integrating ML models into simulation semantics [4], [5]. As a result, ML integration requires manual integration or on external, ad-hoc extensions layered on top of the simulation infrastructure. Existing approaches are therefore fragmented and tool-specific: ML inference is embedded into custom TLM interfaces targeting fixed application domains [6], delegated to external Python runtimes with strong toolchain coupling [7], or manually re-implemented in C++ at the cost of significant engineering effort and limited model portability. These limitations hinder systematic integration of heterogeneous ML techniques within a unified simulation environment.

This work introduces **SAMpLE** (*Systemc-Ams Machine Learning-based framework for virtual prototyping*), a frame-

work that enables direct execution of ML models as native TDF components. The goal of *SAMpLE* is to extend SystemC-based virtual platforms with support for ML models that can either *learn online*, adapting incrementally sample-by-sample during simulation execution, or be *trained offline* and loaded into the simulation via ONNX at elaboration time.

TDF was chosen as its semantics naturally aligns with ML inference: both operate on causal, sample-driven execution, where fixed-size input vectors are transformed into outputs at discrete activation steps. This equivalence allows ML models to be executed directly within the `processing()` semantics of TDF, removing the need for co-simulation layers, inter-process communication, or model translation.

SAMpLE implements this integration through a lightweight module abstraction that decouples simulation semantics from inference execution. The framework provides two interchangeable back-ends:

- 1) *ONLINE backend*: a native C++ implementation designed for light-weight models and online-adaptive learning algorithms, tightly integrated with the SystemC-AMS simulation kernel;
- 2) *OFFLINE backend*: an ONNX Runtime-based implementation [8] that enables transparent deployment and load of models trained using heterogeneous ML frameworks, e.g., PyTorch, TensorFlow, and scikit-learn [9]–[11].

Both backends conform to a unified interface, enabling heterogeneous ML models to coexist and run under identical SystemC-AMS simulation conditions.

The main contributions of this work are thus as follows:

- A native integration methodology that *embeds ML models directly within SystemC-AMS TDF semantics*, eliminating the need for external co-simulation frameworks, middle-ware layers, or manual model translation;
- A *unified MLModule abstraction* that separates simulation behavior from inference execution, enabling interchangeable ML backends within a single simulation environment;
- A *dual-backend architecture* supporting both lightweight native C++ implementations and portable ONNX-based deployment of models trained using heterogeneous ML frameworks;
- An *open-source reproducible evaluation framework* and Language Reference Manual (LRM)-compliant lifecycle management infrastructure supporting deterministic dataset handling, CSV-driven experimentation, and cross-framework model parity verification [12].

This work has received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No. 101139789 (HAL4SDV).

979-8-3195-3997-7/26/\$31.00 ©2026 IEEE

II. BACKGROUND AND RELATED WORKS

A. SystemC and SystemC-AMS

SystemC is a C++-based system-level modelling framework standardized under IEEE 1666 and widely adopted for the design and simulation of complex embedded and heterogeneous systems [13]. By extending standard C++ with HW-oriented abstractions, SystemC enables the unified modelling of HW and SW components within a common simulation environment.

To support mixed-signal and cyber-physical systems, the SystemC Analog/Mixed-Signal (SystemC-AMS) extension introduces additional Models of Computation (MoCs) tailored to analogue and dataflow-oriented simulations [14], and is suited for multi-domain virtual platform development [15]–[17]. Among the available MoCs, the TDF represents systems as interconnected processing blocks that periodically consume and produce samples according to statically scheduled execution semantics. Modules communicate through ports and operate at predefined rates and timesteps, enabling deterministic, causal, and computationally efficient simulation while preserving temporal behavior.

B. ONNX

Open Neural Network Exchange (ONNX) is an open standard for representing machine learning and deep learning models in a platform-independent format [8]. Originally introduced by Microsoft and Meta, ONNX is designed to improve interoperability among machine learning frameworks, compilers, and hardware accelerators. The format enables models trained in one framework (e.g., PyTorch [9] or TensorFlow [10]) to be executed across different inference engines and hardware platforms without requiring substantial modifications.

In this work, ONNX models are executed using the ONNX Runtime library [18], which provides a high-performance inference engine with support for graph optimizations and multiple execution backends (e.g., CPU and hardware accelerators). This runtime layer serves as the bridge between the exported model graph and the SystemC-AMS simulation environment, enabling deterministic evaluation of machine learning inference within the TDF execution flow.

C. Related Work

The integration of ML techniques into system-level design environments has received increasing attention due to the growing adoption of data-driven methodologies in embedded and cyber-physical systems [19], [20]. ML models are increasingly employed to approximate computationally intensive behaviors, including power estimation, sensor modeling, fault prediction, adaptive control, workload characterization, and reduced-order system representations [21], [22]. As a consequence, virtual prototyping environments are progressively required to support the execution of heterogeneous ML workloads alongside traditional system-level simulation models.

SystemC and SystemC-AMS are widely adopted for Electronic System-Level (ESL) design, virtual prototyping, and mixed hardware/software simulation [15]–[17]. Several methodologies have extended SystemC-based environments to support increasingly complex embedded workloads, including distributed simulation, heterogeneous system modeling, and

transaction-level virtual platforms [4], [23]. More recently, SystemC has also been employed to model and evaluate AI accelerators and deep-learning-oriented architectures during early design-space exploration [24], [25].

Despite these advances, current SystemC-based methodologies do not provide a standardized mechanism for integrating native ML models directly as native simulation components. Existing approaches typically rely on one of three strategies: first, export models as Functional Mock-up Units (FMUs) according to the Functional Mock-up Interface (FMI) standard, enabling their execution within heterogeneous co-simulation environments. In such setups, ML models and system-level components are instantiated as independent FMUs and orchestrated by a master algorithm, allowing joint simulation across different tools. However, while this improves interoperability, it still introduces non-negligible synchronization overhead [7], [26]–[28]. Second, trained models are frequently reimplemented manually in C++ to enable tight integration within the simulation kernel [24]. Third, application-specific interfaces are developed around particular ML libraries or accelerator frameworks [6], [29]. Overall, although effective for specific use cases, these solutions generally reduce portability, increase maintenance complexity, and complicate reproducibility across simulation environments.

III. METHODOLOGY

This section presents the SAMpLE methodology, outlined in Figure 1. Subsection III-A introduces the architecture, that is detailed in the following Subsections B-D, while Subsection III-E discusses current limitations and outlines a proposal for standardisation. SAMpLE is released open-source at [12].

A. Architecture Overview

The objective of SAMpLE is to provide a unified SystemC-AMS framework for integrating ML models into the early design evaluation of virtual platforms, enabling reproducible and comparable ML-driven assessment across heterogeneous model families. As shown in Fig. 1, the workflow is driven by two inputs: a CSV dataset describing the system behaviour to be modelled and a JSON configuration file (`config.json`) that serves as the single source of truth for all subsequent stages.

The process begins with dataset ingestion and preprocessing. Then, depending on the selected configuration, models are either trained offline in a Python environment and imported into the SystemC-AMS module via ONNX Runtime or trained directly within the framework during the initialisation phase. In both cases the model is exposed to the simulation through a single `IMLModel` interface, so that the TDF scheduler interacts with every model family in an identical manner. This decoupling of *simulation semantics* from *inference execution* is the property that the rest of the methodology builds upon.

B. Data Preprocessing and Feature Analysis

The preprocessing stage is implemented as a modular, configuration-driven pipeline that promotes reproducibility and portability across different simulation backends. All parameters, including file paths, feature schemas, and processing options, are specified in a centralized JSON configuration file, thereby

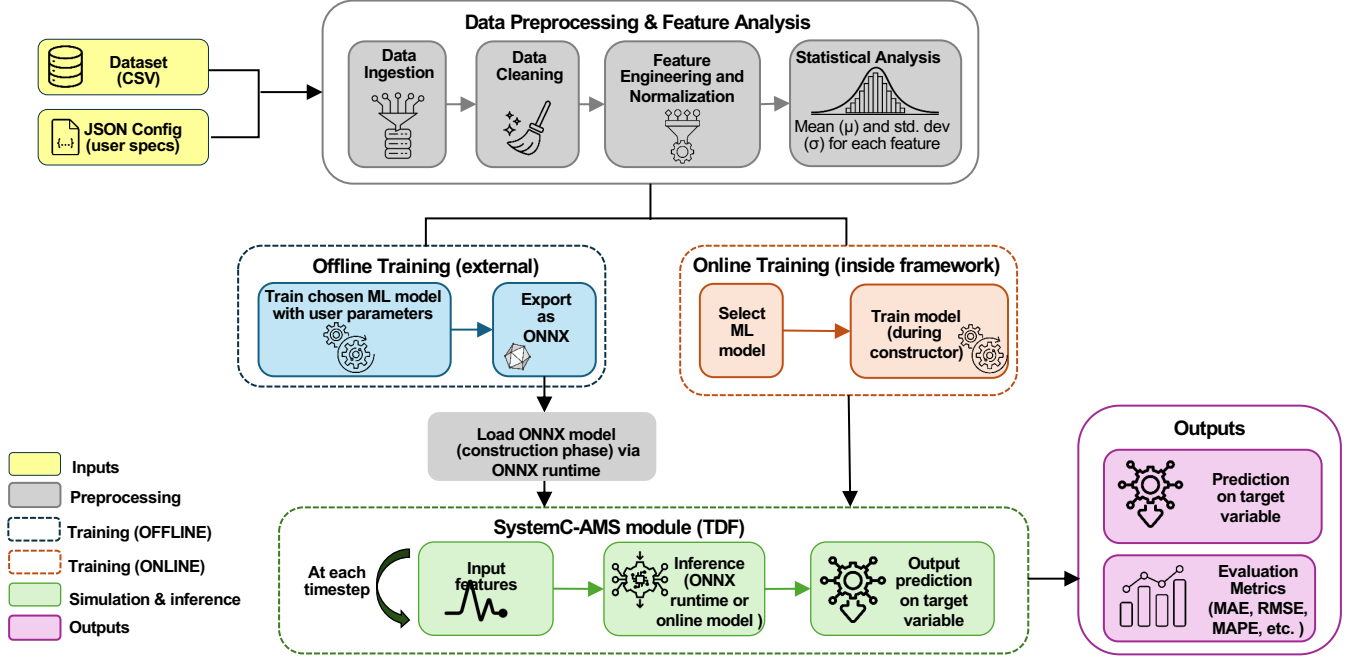


Fig. 1. Architecture overview of the SAMpLE framework. The workflow is driven by a dataset and a JSON-based configuration file. The preprocessing stage produces a standardised dataset representation and normalisation parameters. Two execution modes are supported: offline training through ONNX-exported models loaded via ONNX Runtime, and online training performed directly inside the SystemC-AMS module. During Timed Dataflow (TDF) simulation, the selected backend performs inference at each timestep, producing predictions and runtime evaluation metrics (MAE, RMSE, R^2).

eliminating hard-coded dependencies. The `config.json` exposes 18 keys across five sections (model, dataset, simulation, validation, debug), all optional with framework defaults.

`dataset.csv` requires one row per timestep, a target column, and named feature columns (inline or via file), with no other structural constraints. The pipeline comprises four stages. First, *data ingestion* loads the raw dataset and extracts timestep information. Second, *data cleaning* removes missing or invalid entries. Third, *feature engineering* derives temporal and aggregate features and applies cyclical encoding to periodic temporal attributes to preserve their inherent periodicity (e.g., hour-of-day or day-of-week). Cyclical encoding eliminates artificial discontinuities at period boundaries (e.g., between hours 23 and 0) by mapping each periodic feature to a two-dimensional representation:

$$x_{\sin} = \sin\left(\frac{2\pi t}{T}\right) \quad x_{\cos} = \cos\left(\frac{2\pi t}{T}\right) \quad (1)$$

where t is the raw feature value and T is the period defined in the configuration file (e.g. $T=24$ for hourly, $T=7$ for weekly cycles). This design ensures consistency across datasets without requiring source-code modifications.

The fourth stage performs *statistical analysis*, computing descriptive metrics such as the mean, standard deviation, minimum, and maximum values of each feature. These statistics are subsequently used both as auxiliary inputs for machine-learning workflows and as reference values during model evaluation. To support dataset inspection and validation, the framework also provides a visualization utility that generates plots of the raw data together with the computed statistical summaries.

Finally, the processed dataset is serialized into `.csv` format and partitioned into training, validation, and test subsets using a seed-controlled, chronology-preserving procedure. The test partition is subsequently consumed during SystemC-AMS simulation to evaluate the predictive performance of the generated machine-learning models under execution conditions representative of the target system.

C. Training Phase

SAMpLE supports two complementary training paradigms that share the same preprocessing pipeline and simulation interface, differing only in when model fitting is performed: *offline training* path, that allows the user to load already existing models or to train the model with a user-defined pipeline (Section III-C2), and an *online training* path, where all training is handled by SAMpLE, with few user-defined parameters Section III-C1.

1) *Online training*: In the online training path, model construction and parameter initialization are performed automatically at simulation start-up, eliminating the need for a separate training phase. The framework provides a set of predefined training strategies that can be selected through the configuration file, enabling rapid deployment of predictive models with minimal manual intervention. Although this approach may not achieve the accuracy attainable through extensive offline hyperparameter optimization, it significantly reduces model development effort and accelerates exploration of design alternatives.

At the core of the online mechanism is a unified predictor abstraction, `IMLModel`, that separates feature ingestion, predic-

tion, and adaptation logic. Each concrete predictor implements four functions:

- `required_window_size()`: allows to set how many past samples the model needs before its first valid output;
- `predict(window)`: maps a fixed-size sliding window of past feature vectors to one or more output predictions;
- `update(target)`: performs an incremental parameter step against the most recent ground-truth observation;
- `supports_training()`: exposes at runtime whether the model supports in-framework adaptation.

The current version of SAMpLE supports a number of ML model types, including adaptive linear estimators and nonlinear or regime-aware learning approaches, providing a broad basis for modelling time-varying dynamic systems. This set will be extended as part of future work. The type of model to be constructed, together with any parameters, can be set by the user in the initial configuration file.

The supported models include adaptive linear, ensemble-based, regime-switching, and nonlinear online learning approaches. NLMS-ARX and EW-RLS-ARX extend ARX models with recursive adaptation mechanisms, respectively based on normalized least-mean-squares and exponentially weighted recursive least squares [30], [31]. The Hedge Ensemble dynamically combines multiple predictors through online weight updates [32], while GHMM-Regime captures latent operating regimes using Gaussian hidden Markov models [33]. Nonlinear adaptation is addressed by EKF-MLP, which trains multilayer perceptrons through extended Kalman filtering [34], and by KRLS-ALD, which performs sparse kernel recursive least-squares regression using an approximate linear dependency criterion [35]. Overall, these models provide complementary capabilities in terms of interpretability, adaptivity, nonlinear modelling power, and computational efficiency.

2) *Offline training*: In the offline training path, model fitting is performed outside the SAMpLE framework using arbitrary machine-learning workflows and toolchains. This approach supports two deployment scenarios: (i) training custom models using the preprocessed datasets generated by SAMpLE, and (ii) importing previously trained models without requiring execution of the preprocessing and training stages. As a result, existing machine-learning assets can be reused directly within the simulation environment.

To support framework-independent deployment, SAMpLE adopts the Open Neural Network Exchange (ONNX) format as a common model representation. Models trained using external frameworks are exported to ONNX and wrapped to conform to the `IMLModel` interface (e.g., `required_window_size()` and `predict(window)`). The resulting ONNX model, together with the serialized normalization and scaling parameters, constitutes a self-contained deployment artifact.

The ML model is loaded at simulation initialization time, and required two validation steps:

- a static input-shape check, verifying that the number of ONNX input nodes matches the configured feature-vector width;
- a dry-run inference on a zero-padded vector confirming that the graph is callable and produces a finite output.

These guards ensure that the simulation-time environment faithfully replicates the Python training environment without embedding an interpreter or requiring inter-process communication.

D. Simulation Semantics

From a semantic perspective, ML inference aligns naturally with the TDF execution model: given a fixed set of input features sampled at regular intervals, inference produces a deterministic output at each activation; a behavior that maps directly onto the causal, periodically-scheduled signal processing abstraction that TDF modules implement. This correspondence suggests that trained ML models can be represented as TDF processing blocks, without requiring auxiliary co-simulation layers or external execution infrastructures.

SAMpLE leverages this correspondence by integrating ML model construction, deployment, and inference directly within the TDF execution flow. Unlike co-simulation approaches or manually translated implementations, the proposed framework introduces a unified abstraction layer that preserves the semantics of the underlying TDF model while supporting multiple machine-learning backends. This design ensures deterministic execution, reproducibility, and portability across simulation environments.

Figure 2 illustrates the structure of the proposed TDF primitive. Input features are received through the `features` port, while predictions are emitted through the `prediction` port. A dedicated `ready` signal indicates when sufficient historical samples have been accumulated to produce valid predictions. Two optional ports, `ground_truth` and `valid`, support online evaluation and validation of model outputs during simulation.

The proposed primitive fully complies with the standard TDF execution semantics and lifecycle:

a) *Simulation setup*: The constructor allows to instantiate the module and loads the information contained in the input configuration file. The `set_attribute` primitive extrapolates such information to set the module time step and the update rate of output ports. The `initialize` function instantiates an `IMLBackend` object with two different behaviors, depending on the type of training:

- *offline training*: loads the ML model trained offline via ONNX, sets up its parameters based on the initial configuration, and triggers the validity checks (in case of failure, simulation is stopped via `sc_stop()`);
- *online training*: the chosen ML model is built starting from the initial configurations, the statistical information derived from the feature analysis, and the training set.

b) *Simulation and inference*: The processing function triggers one inference of the ML model (e.g., via invocation of the `predict` function) and updates the `prediction` port at any activation time step. Given that the model may require a window size to achieve warmup, the `ready` is set to true only after `required_window_size()` ticks, to notify that the generated value can now be used as valid data. The current implementation supports single-step-ahead prediction, although the abstraction can be extended to multi-step forecasting strategies in future work.

c) *Model test and validation*: When the optional `ground_truth` signal is asserted, the processing stage additionally evaluates prediction accuracy against reference values from the test set. Error statistics are accumulated throughout the simulation, and the `valid` output is asserted whenever a prediction violates the configured acceptance criteria. At simulation termination, the framework reports standard regression metrics, including coefficient of determination (R^2), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

d) *Backend interchangeability*: The definition of a common `IMLModel` interface for both online and offline trained models and of a standard interface allows to make e.g. an ONNX-exported Random Forest externally indistinguishable from one containing a native online predictor. This interchangeability allows to compare different model families (e.g., classical estimators, tree ensembles, deep sequence models) on the same testbench with the same dataset split and metric code, without model-specific evaluation infrastructure. Additionally, the simulation-level cost of a new model (inference latency, memory footprint) is measured directly inside the virtual platform rather than in an isolated benchmark.

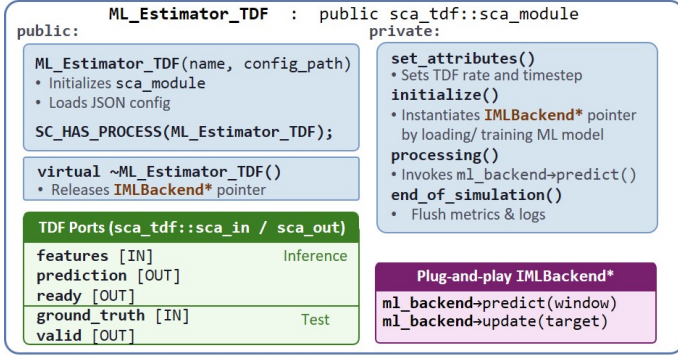


Fig. 2. Structure of the proposed SystemC-AMS primitive. It preserves the TDF lifecycle, features an instance of `IMLBackend` to load and handle the ML model, and features a fixed interface with mandatory inference ports and optional test ports.

E. Toward a standardised SystemC-AMS primitive

The integration patterns established by SAMpLE suggest that the definition of a similar primitive in the SystemC-AMS standard would be beneficial. ML surrogates are now routine components of production virtual platforms, yet the standard provides no mechanism to host them, and the only sample generation primitive remains the sinusoidal signal source `sca_lsf::sca_source`. On the other hand, the `IMLModel` contract (windowed input, fixed-size prediction output, optional ground-truth update, `ready` flag) has proved sufficient to integrate a number of heterogeneous models, from a compact NLMS-ARX to more complex ones. Finally, the trade-off between ecosystem portability (ONNX) and per-tick adaptability (native C++) is structural; the dual-backend design will remain necessary regardless of how individual ML frameworks evolve. The proposal is intentionally conservative: it adds one module type with no additional impact on the language, and avoids tool-vendor lock-in by shipping two interchangeable reference backends.

IV. EXPERIMENTAL EVALUATION

We evaluate SAMpLE with respect to three directions:

- Q1 Fidelity**: does SystemC-AMS execution reproduce the numerical behaviour of Python models after ONNX export?
- Q2 Heterogeneity**: can structurally diverse ML models be executed through a unified `IMLModel` interface without modifying the SystemC netlist?
- Q3 Portability**: does the pipeline transfer across datasets within the same application domain without changes to the simulator source code?

The goal of the evaluation is not to optimise individual ML models, which can be tuned independently via the `config.json` file, but to rather assess the generality and interoperability of the proposed framework across heterogeneous model classes and datasets.

A. Experiments setup

We consider two datasets with distinct temporal characteristics. UCI Appliances [36] contains 19,735 samples at 10-minute resolution and represents noisy, occupancy-driven residential energy consumption ranging from 10 to 1,080Wh. Tetuan City [37] contains 52,416 samples at the same resolution and represents smooth, periodic urban electricity demand between 13.9 and 52.2kW. Both datasets are split chronologically to avoid temporal leakage.

For offline training, we supported ARX Ridge, which applies ridge-regularized least squares to ARX regressors to improve robustness under correlated inputs [38], Random Forest, which performs nonlinear prediction through an ensemble of randomized decision trees [39], and XGBoost, which uses regularized gradient-boosted decision trees for scalable and accurate regression [40].

All experiments run on an Intel Core i7-10700 with 16 GB RAM under Ubuntu 22.04, and all artifacts (configurations, datasets, models, predictions, and metric reports) are released alongside the source code for exact reproducibility [12].

For each experiment, we report total wall-clock simulation time¹, R^2 , RMSE, and MAE, expressed in Wh for UCI, W for Tetuan.

B. Results and Discussion

Tables I and II jointly answer the three research questions.

For the stateless offline models, SAMpLE reproduces Python predictions (**Q1**). Across both datasets, R^2 , RMSE, and MAE for ARX, XGB, and RF are virtually identical to the Python baseline, with the largest discrepancy being $\Delta R^2 = 0.003$ for XGB on the UCI dataset, within parity constraints. These results indicate that ONNX deployment preserves predictive fidelity with minimal overhead: all offline runs complete in under 1.5s, even for ensemble models.

The framework unifies structurally heterogeneous model families under a single SystemC-AMS netlist (**Q2**). Three offline backends (linear autoregressive, gradient-boosted trees, and random forest) and eight online backends (adaptive filters,

¹Time: offline reports Python CPU training time; simulation fidelity reports TDF validation time; online reports wall-clock simulation time, including elaboration and scheduling.

TABLE I

OFFLINE TRAINING AND SIMULATION FIDELITY RESULTS. CHRONOLOGICAL 15 % TEST SPLIT (2 954 UCI / 7 855 TETUAN SAMPLES). TIME DENOTES CPU TRAINING TIME FOR OFFLINE MODELS AND SYSTEMC-AMS TDF VALIDATION TIME FOR SIMULATION FIDELITY EXPERIMENTS. RMSE AND MAE ARE REPORTED IN WH (UCI) AND W (TETUAN).

Model	UCI Appliances				Tetuan City			
	R^2	RMSE	MAE	Time (s)	R^2	RMSE	MAE	Time (s)
<i>Offline training (Python)</i>								
ARX	0.278	77.3	42.9	0.020	0.986	721.6	503.4	0.600
XGB	0.235	79.6	48.4	0.100	0.985	755.4	535.9	2.900
RF	0.148	83.9	59.3	0.600	0.974	984.1	708.7	8.100
<i>Simulation fidelity (SystemC-AMS TDF)</i>								
ARX	0.278	77.3	42.9	0.206	0.986	721.6	503.9	0.343
XGB	0.238	79.8	48.4	0.209	0.985	755.4	535.9	0.350
RF	0.148	83.9	59.3	0.262	0.974	984.1	708.7	1.478

Bold: best per metric per dataset.

TABLE II

ONLINE ADAPTIVE FORECASTING RESULTS. CHRONOLOGICAL 20 % TEST SPLIT (3 947 UCI / 10 484 TETUAN SAMPLES). TIME DENOTES WALL-CLOCK SYSTEMC-AMS SIMULATION TIME, INCLUDING MODEL EXECUTION, INITIALIZATION OVERHEAD, AND TDF SCHEDULING. RMSE AND MAE ARE REPORTED IN WH (UCI) AND W (TETUAN).

Model	UCI Appliances				Tetuan City			
	R^2	RMSE	MAE	Time (s)	R^2	RMSE	MAE	Time (s)
Hedge Ensemble	0.879	31.5	7.0	0.009	0.992	550.12	369.71	0.022
NLMS-ARX	0.763	43.2	20.8	0.009	0.9961	383.2	267.7	0.048
Online-GP	0.741	45.2	21.7	0.768	0.9858	736.4	484.7	3.602
EW-RLS-ARX	0.648	52.6	17.1	0.010	0.9954	416.9	298.9	0.049
GHMM	0.595	56.0	23.8	0.009	0.9960	391.7	285.8	0.046
GBT	0.565	58.5	28.2	0.009	0.9326	1 601.8	1 322.6	0.045
EKF-MLP	0.472	64.4	32.9	0.081	0.9689	1 087.7	771.1	0.313
KRLS-ALD	0.268	75.9	38.1	0.045	0.8883	2 062.1	1 578.6	0.222

Bold: best confirmed result per dataset.

kernel methods, Gaussian processes, regime-switching HMMs, EKF-MLP, KRLS-ALD, Online-GBT, and Online-GP) execute within the same simulation structure, where model switching requires no manual code modification (Q3). Offline models are selected via a single field in `config.json`, while online models are instantiated through a compile-time `IMLModel` dispatch. The same mechanism applies across datasets: transitioning from UCI to Tetuan required only swapping the input CSV and feature schema, with no changes to C++ code, SystemC modules, or build configuration. The differing predictive behavior across datasets reflects workload complexity rather than implementation error. On the smooth, periodic Tetuan signal, even linear ARX achieves $R^2 = 0.986$, with all offline models clustering closely. In contrast, on the highly non-stationary UCI occupancy trace, offline models degrade substantially to $R^2 \in [0.13, 0.28]$, as their fixed parameters cannot track regime shifts.

This gap is bridged by adaptive online learners natively embedded in the same timed-dataflow execution model. On UCI, the best-performing online method (Hedge Ensemble) reaches $R^2 = 0.879$, more than a threefold improvement over the best offline baseline, due to per-sample adaptation to occupancy-driven dynamics. Other online methods (NLMS-ARX, EW-RLS-ARX, GHMM, Online-GP) occupy a 5–30 percentage-point performance band, while KRLS-ALD and Online-GBT lag, reflecting differences in how well each algorithm matches non-stationary behavior under identical scheduling and metric conditions.

On Tetuan, both offline and online approaches converge, with

NLMS-ARX reaching $R^2 = 0.9961$ compared to 0.986 for offline ARX. In this regime, accuracy differences are secondary to computational cost. NLMS-ARX completes inference in 48ms end-to-end with no external training stage, whereas the offline pipeline requires 0.6s of Python training plus 0.343s of simulation and ONNX export overhead. Across online methods, runtime spans three orders of magnitude, from 9ms (linear filters) to 3.6s (Online-GP on Tetuan), all measured directly within the virtual platform under TDF semantics. This provides the primary contribution of SAMpLE: a unified, in-situ benchmark enabling fair comparison of heterogeneous learning paradigms under identical execution and timing constraints.

V. CONCLUSION

This paper presented SAMpLE, an open-source SystemC-AMS framework that integrates ML inference and online learning as native TDF citizens, removing the ad-hoc integration layer that fragments ML-enabled virtual prototyping.

Experimental results show that SAMpLE’s in-simulation predictions closely match Python-based evaluations, with only minor cross-environment discrepancies, confirming the effectiveness of the ONNX parity guarantees enforced at construction time. Through the unified `IMLModel` abstraction, heterogeneous model families are compared under identical SystemC-AMS scheduling, dataset partitions, and metric pipelines.

Future work will extend SAMpLE with heterogeneous model ensembles, uncertainty-aware inference through confidence-aware TDF interfaces, and validation on larger virtual platforms for design space exploration.

REFERENCES

- [1] S. Pagani, P. D. S. Manoj, A. Jantsch, and J. Henkel, "Machine learning for power, energy, and thermal management on multicore processors: A survey," *IEEE TCAD*, vol. 39, no. 1, pp. 101–116, 2020.
- [2] J. L. C. Hoffmann and A. A. Fröhlich, "Online machine learning for energy-aware multicore real-time embedded systems," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 493–505, 2022.
- [3] L. Bold, L. Grüne, M. Schaller, and K. Worthmann, "Data-driven MPC with stability guarantees using extended dynamic mode decomposition," *IEEE Transactions on Automatic Control*, vol. 70, no. 1, pp. 534–541, 2025.
- [4] A. Mahmoudi, A. Nešković, C. Thermann *et al.*, "A systematic mapping study on SystemC/TLM modeling capabilities in new research domains," *ACM TODAES*, vol. 30, no. 4, 2025.
- [5] Q. Thibeault and G. Pedrielli, "Multicosim: A python-based multi-fidelity co-simulation framework," 2025. [Online]. Available: <https://arxiv.org/abs/2506.10869>
- [6] Y.-C. Lee, T.-S. Hsu, C.-T. Chen, J.-J. Liou, and J.-M. Lu, "NNSim: A fast and accurate SystemC/TLM simulator for deep convolutional neural network accelerators," in *Proc. of VLSI-DAT*, 2019, pp. 1–4.
- [7] R. Balin, F. Simini, C. Simpson *et al.*, "In situ framework for coupling simulation and machine learning with application to CFD," 2023. [Online]. Available: <https://arxiv.org/abs/2306.12900>
- [8] "Github - onnx/onnx: Open standard for machine learning interoperability," <https://github.com/onnx/onnx>.
- [9] A. Paszke, S. Gross, F. Massa, A. Lerer *et al.*, *PyTorch: an imperative style, high-performance deep learning library*, 2019.
- [10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis *et al.*, "TensorFlow: a system for large-scale machine learning," in *Proc. of USENIX OSDI*, 2016, p. 265–283.
- [11] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, no. null, p. 2825–2830, 2011.
- [12] "SAMPLE: SystemC-Ams Machine LEarning-based framework for virtual prototyping," <https://github.com/andreialbu28/SAMPLE>.
- [13] SystemC Standardization Working Group, "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2023 (Revision of IEEE Std 1666-2011)*, pp. 1–618, 2023.
- [14] SystemC AMS extensions Working Group, "IEEE Standard for Standard SystemC Analog/Mixed-Signal Extensions Language Reference Manual," *IEEE Std 1666.1-2016*, pp. 1–236, 2016.
- [15] V. Muttillio, L. Pomante, M. Santic, and G. Valente, "SystemC-based co-simulation/analysis for system-level hardware/software co-design," *Comput. Electr. Eng.*, vol. 110, no. C, 2023.
- [16] F. Pêcheux, C. Grimm, T. Maehne, M. Barnasconi, and K. Einwich, "SystemC AMS based frameworks for virtual prototyping of heterogeneous systems," in *Proc. of IEEE ISCAS*, 2018, pp. 1–4.
- [17] F. Tosoni, S. Vinco, and F. Fummi, "Modeling and simulation of thermal faults in batteries for enhanced safety," in *Proc. of IEEE DDECS*, 2025, pp. 115–118.
- [18] O. R. developers, "Onnx runtime," <https://onnxruntime.ai/>, 2021, version: 1.20.0.
- [19] A. K. A. Kumar, S. Al-Salamin, H. Amrouch, and A. Gerstlauer, "Machine learning-based microarchitecture-level power modeling of CPUs," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 941–956, 2023.
- [20] R. Rai and C. K. Sahu, "Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (CPS) focus," *IEEE Access*, vol. 8, pp. 71 050–71 073, 2020.
- [21] A. Mankodi, A. Bhatt, and B. Chaudhury, "Predicting physical computer systems performance and power from simulation systems using machine learning model," *Springer Computing*, vol. 105, no. 5, p. 935–953, 2022.
- [22] B. Sayin, T. Zoppi, N. Marchini, F. A. Khokhar, and A. Passerini, "Bringing machine learning classifiers into critical cyber-physical systems: A matter of design," *IEEE Access*, vol. 13, pp. 94 858–94 877, 2025.
- [23] Y. Chen, D. Baek, J. Kim *et al.*, "A SystemC-AMS framework for the design and simulation of energy management in electric vehicles," *IEEE Access*, vol. 7, pp. 25 779–25 791, 2019.
- [24] N. Böhm Agostini, S. Dong, E. Karimi, M. Torrents Lapuerta, J. Cano, J. L. Abellán, and D. Kaeli, "Design space exploration of accelerators and end-to-end DNN evaluation with TFLITE-SOC," in *Proc. of IEEE SBAC-PAD*, 2020, pp. 10–19.
- [25] H.-Y. Kao, S.-H. Huang, and W.-K. Cheng, "Design framework for ReRAM-based DNN accelerators with accuracy and hardware evaluation," *Electronics*, vol. 11, p. 2107, 07 2022.
- [26] M. Urbani, M. Bolognese, L. Praticò, and M. Testi, "A tool for the implementation of open neural network exchange models in functional mockup units," in *Modelica Conferences*, 2025, pp. 645–651.
- [27] A. M. Albu, G. Pollo, A. Burrello *et al.*, "Integrating SystemC TLM into FMI 3.0 co-simulations with an open-source approach," in *Proc. of DVCon Europe 2025*, 2025, pp. 29–35.
- [28] Modelica, "Modelica/FMI-standard: Specification of the functional mock-up interface (fmi)," <https://fmi-standard.org/>.
- [29] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni, "ESP4ML: Platform-based design of systems-on-chip for embedded machine learning," in *Proc. of DATE*, 2020, pp. 1049–1054.
- [30] J. Yoo, B. Park, W. Lee, and J. Shin, "A novel NLMS algorithm for system identification," *MDPI Electronics*, no. 12, 2023.
- [31] F. Fraccaroli, A. Peruffo, and M. Zorzi, "A new recursive least squares method with multiple forgetting schemes," in *Proc. of IEEE CDC*, 2015, pp. 3367–3372.
- [32] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: A meta-algorithm and applications," *Theory of Computing*, vol. 8, no. 6, pp. 121–164, 2012.
- [33] M. Wang, Y.-H. Lin, and I. Mikhelson, "Regime-switching factor investing with hidden markov models," *J. Risk Fin. Manag.*, vol. 13, no. 12, p. 311, Dec. 2020.
- [34] S. Gaamouri, M. B. Salah, and R. Hamdi, "Denoising ecg signals by using extended kalman filter to train multi-layer perceptron neural network," *Automatic Control and Computer Sciences*, vol. 52, pp. 528–538, 2018.
- [35] X. Ai, J. Zhao, H. Zhang, and Y. Sun, "Sparse sliding-window kernel recursive least-squares channel prediction for fast time-varying MIMO systems," *Sensors*, vol. 22, no. 16, 2022.
- [36] L. Candanedo, "Appliances Energy Prediction," UCI Machine Learning Repository, 2017, DOI: <https://doi.org/10.24432/C5VC8G>.
- [37] A. Salam and A. El Hiboui, "Power Consumption of Tetouan City," UCI Machine Learning Repository, 2018, DOI: <https://doi.org/10.24432/C5B034>.
- [38] A. K. Ehsanes Saleh, M. Arashi, and B. M. Golam Kibria, *Theory of ridge regression estimation with applications*, ser. Wiley Series in Probability and Statistics, A. K. M. E. Saleh, M. Arashi, and B. M. G. Kibria, Eds. Nashville, TN: John Wiley & Sons, Mar. 2019.
- [39] G. Dudek, "A comprehensive study of random forest for short-term load forecasting," *Energies*, vol. 15, no. 20, 2022.
- [40] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. of ACM SIGKDD*, 2016, pp. 785–794.