Robust Learning and Reasoning
for Complex Event Forecasting

Project Acronym:	EVENFLOW
Grant Agreement number:	101070430 (HORIZON-CL4-2021-HUMAN-01-01 – Research and Innovation Action)
Project Full Title:	Robust Learning and Reasoning for Complex Event Forecasting

DELIVERABLE

D5.2 – Final Version of Verification and Scalability Techniques

Dissemination level:	PU - Public, fully open
Type of deliverable:	R - Document, report
Contractual date of delivery:	31 December 2025
Deliverable leader:	Athena Research Center
Status - version, date:	Final – v1.0, 2025-12-17
Keywords:	scalability, verification, robustness, streaming data

Funded by the
European Union

This document is part of a project that is funded by the European Union under the Horizon Europe agreement No 101070430. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Commission. The document is the property of the EVENFLOW project and should not be distributed or reproduced without prior approval. Find us at www.evenflow-project.eu.

Executive Summary

Deliverable D5.2 reports EVENFLOW's final outcomes (for the second half of the project, since April 2024) on scalability and verification techniques for online neuro-symbolic learning and reasoning in real-time streaming settings. It consolidates methods, tools, and use-case applications addressing the project's core challenges.

The scalability of EVENFLOW is based on two pillars. Stream synopses and parallelism. On the stream synopses side, a synopsis-driven optimization paradigm is introduced for continuous learning. The SuBiTO Framework operationalizes the principle that training must adapt to stream dynamics by jointly optimizing (i) synopsis/compression configuration, (ii) training duration (e.g., epochs), and (iii) neural architecture, explicitly searching for strong accuracy vs training-time trade-offs and presenting these choices through supporting tooling. Parallelism cooperatively works with synopses through parallel synopsis maintenance and through a data-driven synchronization protocol suite that reduces coordination and communication costs in distributed learners. These components are integrated in Distribuito SuBiTO, combining parallel synopsis derivation, smart synchronization, and batched/parallel inference to increase throughput. In parallel, the NeuroFlinkCEP framework delivers scalable neurosymbolic Complex Event Recognition by integrating neural simple-event inference with symbolic, Apache Flink CEP pattern matching, supported by logical-to-physical workflow optimization and monitored deployment across cloud–edge/IoT environments.

Scalability grounds these techniques in three vertical use cases via: SSTRESSED for Industry 4.0 simple event detection, RATS+ for personalized medicine with transfer-learning elements, and infrastructure monitoring via reverse random hyperplane projection plus uncertainty-aware synchronization.

On the verification side, the deliverable defines an EVENFLOW verification stack that spans: (i) formal neural network verification, (ii) probabilistic neuro-symbolic verification, and (iii) temporal verification for streaming neuro-symbolic pipelines where neural perception feeds symbolic automata.

For neuro-symbolic (NeSy) pipelines, where neural outputs feed symbolic arithmetic circuits, we found exact verification to be intractable in our evaluation. Therefore relaxation-based and hybrid pipelines were proposed to balance completeness with scalability requirements. These constitute abstract verification techniques and enable us to achieve robustness guarantees for NeSy systems also using CNN networks.

The contributions include Spatio-Temporal Bound Propagation (STBP) and Spatio-Temporal shared IBP (S-IBP). These are hybrid schemes that solve MILPs for the first layer under structured (shared / fixed / bounded) perturbation constraints, then propagate tight layer-1 bounds forward with efficient interval or linear relaxations. The work also develops probabilistic verification via PAC-interval estimation (LipPOT): a targeted Adam-Sobol sampling engine plus Extreme Value Theory (POT/GPD) constrained by DKW confidence bands to produce high-confidence upper bounds on local Lipschitz constants. For parallel verification, SCANNV applies Bayesian optimisation (with transfer learning and ReLU-stability grey-box signals) to find input splits that reduce wall-clock verification time.

Experiments on synthetic MNIST addition, ROAD-R autonomous driving, temporal complex-event traces, and medical/video benchmarks show STBP and PAC methods substantially improve certified robustness and runtime compared to vanilla IBP and naive solver approaches, at a marginal cost of additional MILP or sampling overhead.

Verification is further grounded in EVENFLOW use cases through a neuro-symbolic robot navigation/collision-avoidance scenario, where robustness is assessed across multiple perturbation levels and validation splits. We discuss routes to improve guarantee tightness through stronger bound methods and robustness-aware training.

Deliverable leader:	Athena Research Center
Contributors:	Nikos Giatrakos, Alessio Lomuscio, Sherwin Varghese, Abdelrahman Hekal, Milan Rakic
Reviewers:	Nikos Katzouris, Vassilis Magginas (NCSR), Satyam Dudhagara (DFKI)
Approved by:	Athanasios Poulakidas, Eleni-Vasiliki Provopoulou (INTRA)

Document History			
Version	**Date**	**Contributor(s)**	**Description**
0.1	06/10/2025	Nikos Giatrakos, Sherwin Varghese	TOC creation
0.2	15/10/2025	Nikos Giatrakos	Section 2.1, 3.1, 3.2
0.3	18/10/2025	Nikos Giatrakos	Section 3.3, 3.4, 3.5
0.4	25/10/2025	Nikos Giatrakos	Section 4, Section 5
0.5	07/10/2025	Nikos Giatrakos	Section 6, Section 7
0.6	15/10/2025	Alessio Lomuscio, Sherwin Varghese, Abdelrahman Hekal, Milan Rakic	Section 8
0.7	28/10/2025	Alessio Lomuscio, Sherwin Varghese, Abdelrahman Hekal, Milan Rakic	Section 9
0.8	15/11/2025	Alessio Lomuscio, Sherwin Varghese, Abdelrahman Hekal, Milan Rakic	Sections 10.1 – 10.4
0.9	30/11/2025	Alessio Lomuscio, Sherwin Varghese, Abdelrahman Hekal, Milan Rakic	Sections 10.5 – 10.6, Section 11 - 12
0.10	15/12/2025	Nikos Giatrakos, Alessio Lomuscio, Sherwin Varghese, Abdelrahman Hekal, Milan Rakic	Final version before QA
1.0	17/12/2025	Athanasios Poulakidas, Eleni-Vasiliki Provopoulou	Version to be submitted after final QA

Table of Contents

Table of Figures

List of Tables

Definitions, Acronyms and Abbreviations

Acronym/ Abbreviation	Title
AMR	Autonomous Mobile Robot
ARC	Athena Research Center
BO	Bayesian Optimization
CE	Complex Event
CER	Complex Event Recognition
CEP	Complex Event Processing
GP(R)	Gaussian Process (Regressor)
HPC	High Performance Computing
ICL	Imperial College London
NAS	Neural Architecture Search
NeSy	Neurosymbolic
PS	Parameter Server
SE	Simple Even
SDEaaS	Synopses Data Engine-as-a-Service
SDE	Synopses Data Engine or Simple Derived Events (depending on the context)
SCANNV	Scalable Neural Network Verification
TNF	Tumor Necrosis Factor
WP	Work Package

Term	Definition
-	

1 Introduction

1.1 Project Information

EVENFLOW develops hybrid learning techniques for complex event forecasting, which combine deep learning with logic-based learning and reasoning into neuro-symbolic forecasting models. This approach combines neural representation learning techniques that construct event-driven features from streams of perception-level data with powerful symbolic learning and reasoning tools, which utilize such features to synthesize high-level, interpretable patterns for forecasting critical events.

To deal with the brittleness of neural predictors and the high volume/velocity of temporal data flows, the EVENFLOW techniques rely on novel, formal verification techniques for machine learning, in addition to a suite of scalability algorithms for training based on data synopsis, federated training and incremental model construction. The learnt forecasters will be interpretable and scalable, allowing for explainable and robust insights, delivered in a timely fashion and enabling proactive decision making.

EVENFLOW is evaluated on three use cases related to (1) oncological forecasting in healthcare, (2) safe and efficient behaviour of autonomous transportation robots in smart factories and (3) reliable life cycle assessment of critical infrastructure.

Table 1: The EVENFLOW consortium.

Number[1]	Name	Country	Short name
1 (CO)	NETCOMPANY-INTRASOFT	Belgium	**INTRA**
1.1 (AE)	NETCOMPANY-INTRASOFT SA	Luxemburg	**INTRA-LU**
2	NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS"	Greece	**NCSR**
3	ATHINA-EREVNITIKO KENTRO KAINOTOMIAS STIS TECHNOLOGIES TIS PLIROFORIAS, TON EPIKOINONION KAI TIS GNOSIS	Greece	**ARC**
4	BARCELONA SUPERCOMPUTING CENTER-CENTRO NACIONAL DE SUPERCOMPUTACION	Spain	**BSC**
5	DEUTSCHES FORSCHUNGSZENTRUM FUR KUNSTLICHE INTELLIGENZ GMBH	Germany	**DFKI**
6	EKSO SRL	Italy	**EKSO**
7 (AP)	IMPERIAL COLLEGE OF SCIENCE TECHNOLOGY AND MEDICINE	United Kingdom	**ICL**

[1] CO: Coordinator. AE: Affiliated Entity. AP: Associated Partner.

1.2 Document Scope

This document provides the advancements made within the scope of EVENFLOW with special emphasis on the time period between M19 (April 2024) to M39 (December 2025). It elaborates on the developed algorithms and prototypes that serve as scalability and verification pillars for the project. It also explains the way they are applied in real world scenarios both derived from EVENFLOW use cases and broader application domains and/or on standard benchmarks and testbeds. It also reports on the evolution of Scalability and Verification Toolkit as exploitable assets of the project. In that, it describes the outcomes of high-quality research in EVENFLOW with anticipated highly impactful software modules.

1.3 Document Structure

This document is comprised of the following chapters:

Chapter 1 presents an introduction to the project and the document.

Chapter 2 summarises the main advancements till M18 of the project as reported in Deliverable D5.1.

Chapter 3 emphasizes on the scalability aspects by means of data stream synopses and distributed/parallel learning and inference presenting the developed prototypes.

Chapter 4 discusses scalable neurosymbolic CER over IoT platforms pushing the developments on scalability made throughout the project, not only at the cloud side but across the cloud to edge continuum.

Chapter 5 presents the current open-source contributions of the scalability approaches developed throughout the project.

Chapter 6 emphasizes on use case-specific adaptations of the generic techniques described in Chapters 3-4 to EVENFLOW use cases and novel scalable techniques for these specific application fields.

Chapter 8 discusses techniques for improving scheduling and augmenting parallelization of verification frameworks, such as Venus, for enhanced scalability in neural network verifiers upon being treated as black or grey boxes.

Chapter 7 introduces the foundational concepts of formal verification for neural networks, outlining core techniques and discussing their applicability to neuro-symbolic architectures. It further examines methods for verifying complex temporal events within neurosymbolic systems.

Chapter 8 extends these verification techniques to spatio-temporal models. It presents hybrid approaches that combine linear programming with interval bound propagation, enabling robustness analysis for high-dimensional perturbations and larger neural network architectures.

Chapter 10 explores verification through probabilistically approximately correct (PAC) bounds. It evaluates the tightness of PAC intervals and assesses their suitability for providing reliable robustness guarantees in comparison to deterministic methods.

Chapter 11 applies these verification techniques to the EVENFLOW Industry 4.0 use case, demonstrating end-to-end verification of a neurosymbolic system deployed in an industrial robotic environment.

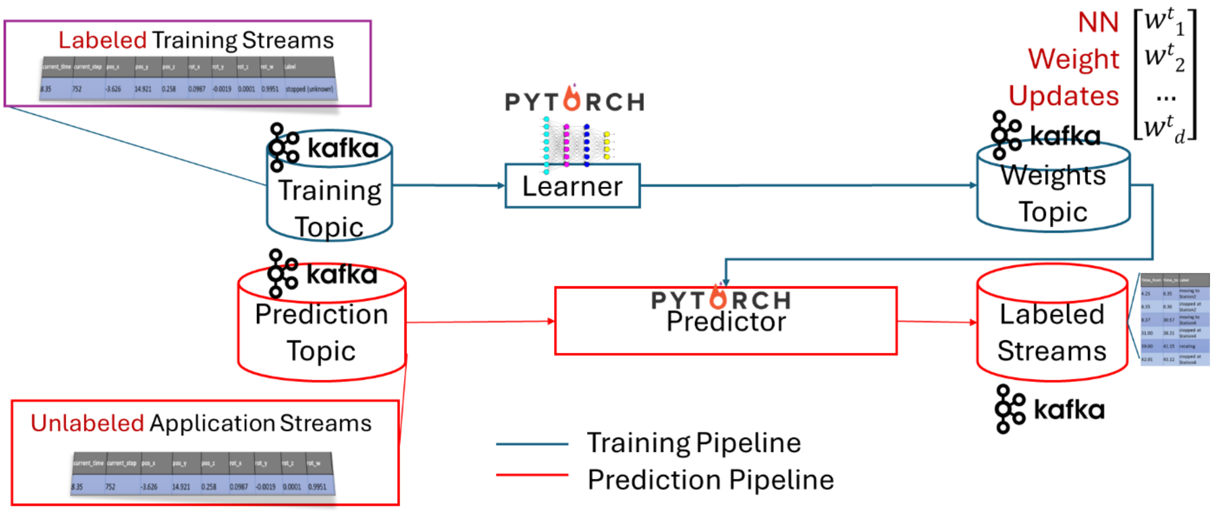
Chapter 12 presents the current open-source contributions of the verification approaches and the toolkits developed for verification of the EVENFLOW Industry 4.0 use case.

2 Overview of Progress till M18

2.1 Recap on Scalability Aspects in EVENFLOW

As documented in Deliverable D5.1, EVENFLOW training and inference pipelines target three types of scalability: (a) horizontal scalability, i.e., scaling with the volume and velocity of the incoming data streams, (b) vertical scalability, i.e., scaling with the number of processed streams and (c) federated scalability, i.e., scaling the computation in (geo-)distributed environments by reducing the amount of transmitted data to preserve bandwidth and reduce network latencies.

The focal point of developing effective and efficient training and inference pipelines in streaming settings revolves around the challenge of achieving appropriate balance/trade-off between accuracy and training time. In streaming setups, the statistical properties and the distribution of incoming streams are highly volatile. Consequently, a neural model that is currently suitable for inference purposes may quickly become obsolete. Therefore, the training process evolves continuously and on par with the prediction/inference process. As soon as an up-to-date trained model becomes available, it should be directly deployed on the prediction pipeline to maintain high quality inference, not only at the current time but, most importantly, in the long run. Figure 1 illustrates the evolution of training and prediction pipelines, instantiated using state-of-the-art frameworks for stream ingestion (i.e., Apache Kafka) and neural learning (PyTorch). The architectural scheme though is independent of the underlying technologies.



current_time	current_step	pos_x	pos_y	pos_z	rot_x	rot_y	rot_z	rot_w	label
8.35	752	-3.626	14.921	0.258	0.0987	-0.0019	0.0001	0.9951	stopped (unknown)

Time_from	Time_to	Label
4.25	8.35	moving to Station2
8.35	8.36	stopped at Station2
8.37	30.57	moving to Station4
31.00	38.31	stopped at Station4
39.00	41.15	rotating
42.01	43.12	stopped at Station4

current_time	current_step	pos_x	pos_y	pos_z	rot_x	rot_y	rot_z	rot_w
8.35	752	-3.626	14.921	0.258	0.0987	-0.0019	0.0001	0.9951

Figure 1: Streaming Training and Inference Pipelines operating on par with one another [REF-01].

The main pillars for achieving scalability in EVENFLOW involve (a) **data streams synopses**, (b) **parallelism/distribution** of computation and (c) **optimal (or preferable) resource allocation** for the involved neural, symbolic or neurosymbolic (NeSy) tasks.

The use of synopses is motivated by the fact that, to establish accurate and rapid training pipelines, it is important to train over representative, good quality data. This does not necessarily include the lot of training data streams that may reach a training pipeline. Therefore, by presenting a carefully crafted data summary in the training pipeline, an accurate model can be continuously produced, simultaneously avoiding exacerbated training times.

However, synopses alone do not suffice for scalable processing. First, even if synopses do manage to controllably reduce the portion of streams that are ingested in a training pipeline, data summaries can still be voluminous. For instance, a recent stream of 10GB reduced by an order of magnitude still yields a training volume of 1GB which may entail important training latency for deep neural network architectures. Second, synopses can reduce the burden in the training pipeline only. The prediction/inference pipeline still needs to operate on the entire stream because client applications do not have the liberty of choosing to apply predictions and forecasts only to a subset of the ingested, unlabelled streams. For these reasons, besides synopses, parallelism on the training and prediction pipelines is the second scalability pillar of EVENFLOW. Figure 2 enhances the streaming architecture of Figure 1 with parallelism across the involved training and prediction pipelines.

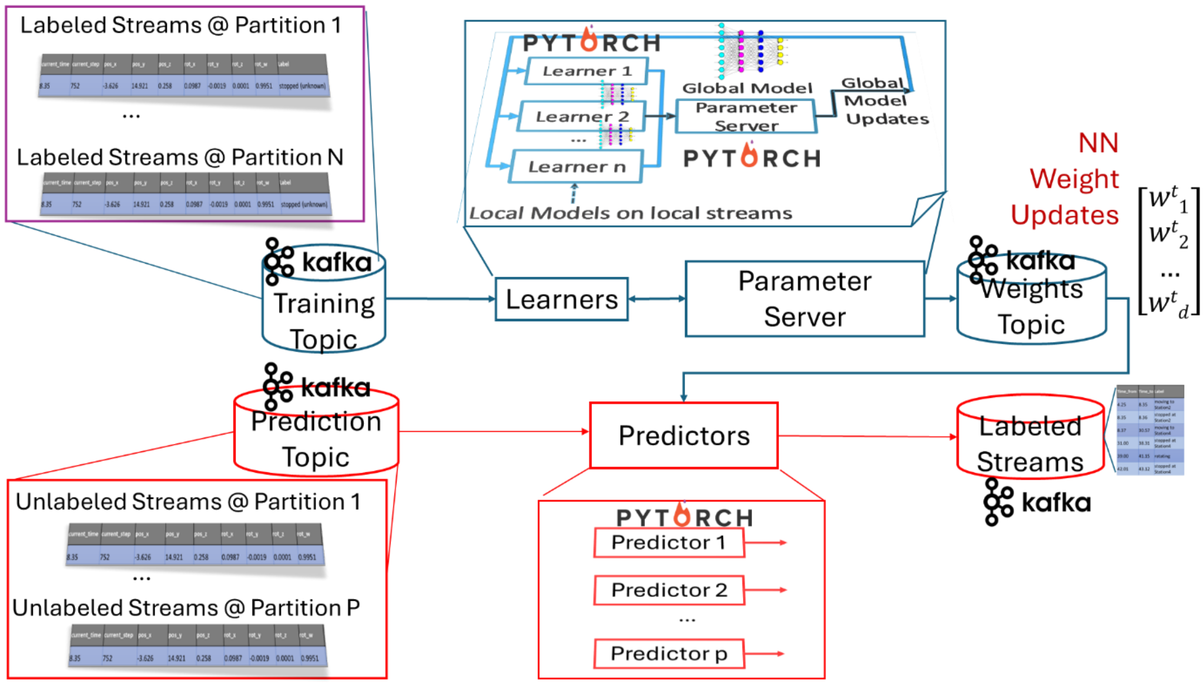


Figure 2: Parallel Streaming Training and Inference Pipelines [REF-01].

The third scalability pillar related to resource allocation in EVENFLOW comes in two forms. First in continuously configuring the degree of parallelism, the size of the synopses, the size of the neural network and the duration of the training process so that the aforementioned accuracy vs training time trade-offs and the preferable combination of such parameters ensure high quality models in an online real-time fashion. Second, models deployed on parallel predictors can be assigned not only at parallel threads being executed on powerful clouds, but also across devices of the computing continuum, in IoT settings.

The progress achieved in EVENFLOW by M18, as reported in Deliverable D5.1, can be summarized as follows:

EVENFLOW proposed and developed the Synopses-as-a-Service paradigm (Figure 3) introducing a state-of-the-art stream summarization engine implemented on Apache Flink [REF-03].

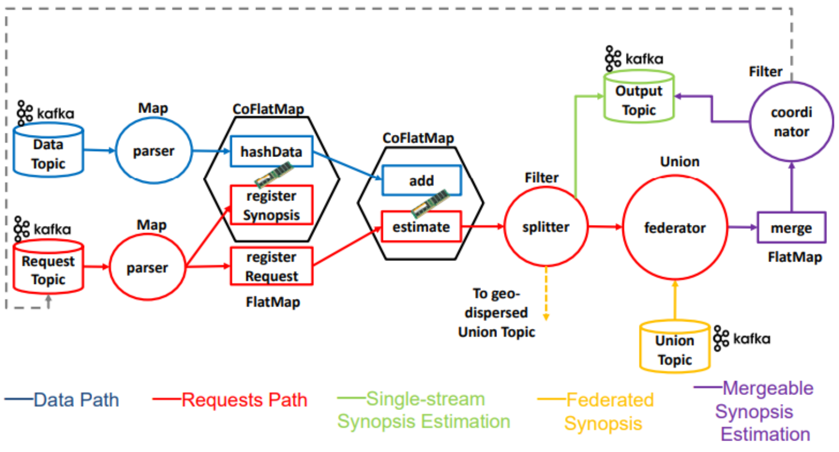


Figure 3: Synopses Data Engine-as-a-Service (SDEaaS) Paradigm in EVENFLOW by M18 [REF-02].

The Synopses-based Training Optimization Paradigm was introduced, which leveraged Bayesian Optimization to continuously prescribe preferable configurations for (a) the size of synopses, (b) the number of training epochs for the training pipelines, for a fixed – a priori defined – neural network architecture.

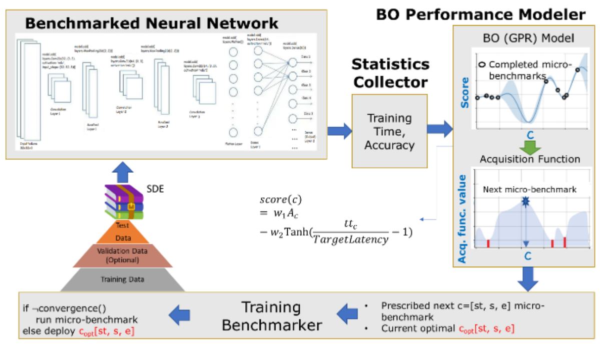
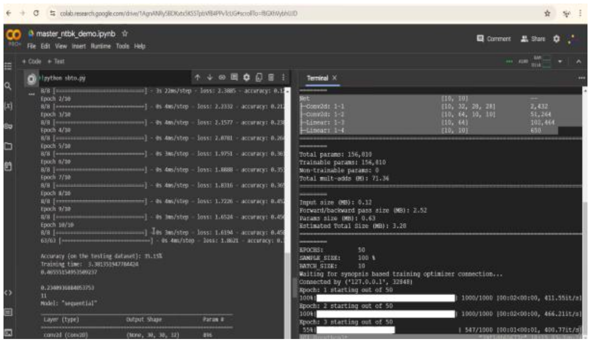

Benchmarked Neural Network
BO Performance Modeler
BO (GPR) Model
Statistics Collector
Training Time, Accuracy
Score
Completed micro-benchmarks
Acquisition Function
Next micro-benchmark
Acq. func. value
SDE
Test Data
Validation Data (Optional)
Training Data
$score(c) = w_1 A_c - w_2 \text{Tanh}(\frac{tt_c}{TargetLatency} - 1)$
if ¬convergence() run micro-benchmark else deploy $c_{opt}[st, s, e]$
Training Benchmarker
Prescribed next c=[st, s, e] micro-benchmark
Current optimal $c_{opt}[st, s, e]$

Figure 4: Synopses-based Training Optimization by M18.


colab.research.google.com/drive/1AgnANRy5BDKxtx5K55TpbVfB4PPvTcUG#scrollTo=fBGXtWybhUJD
master_ntbk_demo.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Terminal

```
!python sbto.py
8/8 [==============================] - 3s 22ms/step - loss: 2.3885 - accuracy: 0.1
Epoch 2/10
8/8 [==============================] - 0s 4ms/step - loss: 2.2332 - accuracy: 0.21
Epoch 3/10
8/8 [==============================] - 0s 4ms/step - loss: 2.1577 - accuracy: 0.23
Epoch 4/10
8/8 [==============================] - 0s 4ms/step - loss: 2.0781 - accuracy: 0.26
Epoch 5/10
8/8 [==============================] - 0s 3ms/step - loss: 1.9751 - accuracy: 0.30
Epoch 6/10
8/8 [==============================] - 0s 4ms/step - loss: 1.8888 - accuracy: 0.35
Epoch 7/10
8/8 [==============================] - 0s 4ms/step - loss: 1.8316 - accuracy: 0.36
Epoch 8/10
8/8 [==============================] - 0s 4ms/step - loss: 1.7226 - accuracy: 0.45
Epoch 9/10
8/8 [==============================] - 0s 3ms/step - loss: 1.6524 - accuracy: 0.45
Epoch 10/10
8/8 [==============================] - 0s 3ms/step - loss: 1.6194 - accuracy: 0.45
63/63 [==============================] - 0s 4ms/step - loss: 1.8621 - accuracy: 0.

Accuracy (on the testing dataset): 35.15%
Training time:  3.381351947784424
0.46555154953509237

0.23409316884053753
11
Model: "sequential"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896
```

```
Net                               [10, 10]                  --
├─Conv2d: 1-1                     [10, 32, 28, 28]          2,432
├─Conv2d: 1-2                     [10, 64, 10, 10]          51,264
├─Linear: 1-3                     [10, 64]                  102,464
├─Linear: 1-4                     [10, 10]                  650
==========================================================================
Total params: 156,810
Trainable params: 156,810
Non-trainable params: 0
Total mult-adds (M): 71.36
==========================================================================
Input size (MB): 0.12
Forward/backward pass size (MB): 2.52
Params size (MB): 0.63
Estimated Total Size (MB): 3.28
==========================================================================
EPOCHS:          50
SAMPLE_SIZE:     100 %
BATCH_SIZE:      10
Waiting for synopsis based training optimizer connection...
Connected by ('127.0.0.1', 32848)
Epoch: 1 starting out of 50
100%|██████████████████████████| 1000/1000 [00:02<00:00, 411.55it/s]
Epoch: 2 starting out of 50
100%|██████████████████████████| 1000/1000 [00:02<00:00, 466.21it/s]
Epoch: 3 starting out of 50
 55%|██████████████              | 547/1000 [00:01<00:01, 400.77it/s]
```

Figure 5: Demo prototype of Synopses-based Training Optimization by M18.

For parallel training, preliminary protocols and experimentation were presented to overcome the limitation of the vanilla synchronous and asynchronous protocols of the Parameter Server paradigm. The basic idea behind the proposed, preliminary protocols was to define a concept drift based on any given thresholded function applied on neural network global (accumulated in a Parameter Server) weights. Decompose this global concept drift trigger to local filters installed on parallel learners and trigger a synchronization only when some learner finds its local filter violated based on a geometric monitoring criterion.

2.2 Recap on Verification Aspects in EVENFLOW

Deliverable D5.1 describes the verification approach for the EVENFLOW project in which we mention about the need for formal verification, the scalability challenges of verification and the approaches for verification of Neurosymbolic systems.

Modern neural networks (NNs) power critical AI applications but remain highly vulnerable to adversarial perturbations—small, often imperceptible input changes that can lead to catastrophic misclassifications. This raises significant safety concerns, especially in domains such as autonomous driving, robotics, smart manufacturing, healthcare diagnostics, and decision-support systems. Due to their black box behaviour and high input dimensionality, understanding and guaranteeing NN behaviour is extremely challenging. Formal verification aims to deliver provable guarantees about system behaviour by mathematically ensuring that certain properties—primarily robustness—hold for all inputs within a perturbation set.

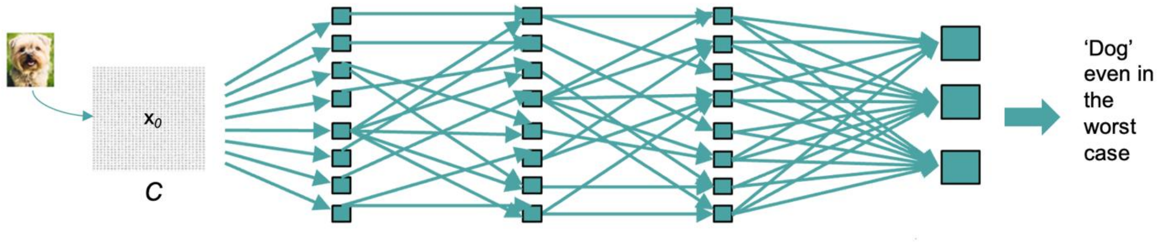


Figure 6: Perturbations on the input should not affect the classification.

Adversarial attacks in both 2D and 3D settings show the extent of NN fragility. Methods such as adversarial image noise, point-cloud manipulation, and mesh perturbation can reliably fool state-of-the-art models without visibly altering the input, strengthening the need for scalable verification tools.

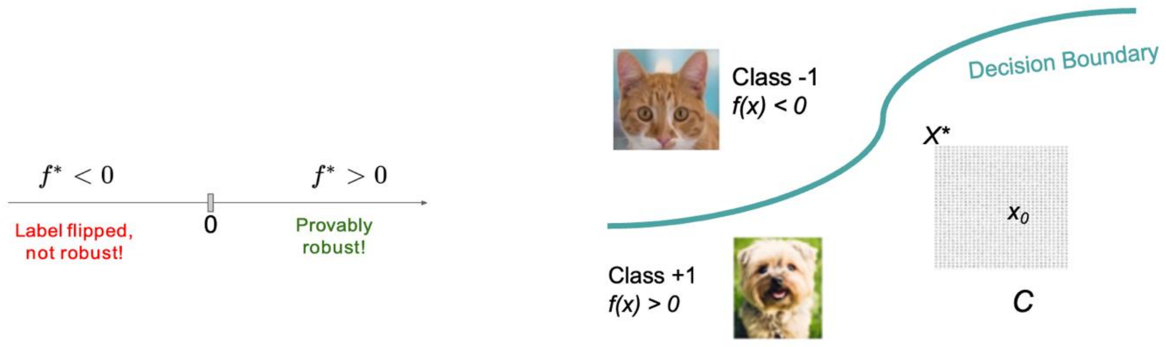


Figure 7: Formal Verification as an Optimization problem.

Verification seeks to determine whether the NN satisfies a desired property (e.g., robustness) over an input region. This can be expressed as an optimization problem that checks whether the network's minimum logit margin remains positive across all admissible perturbations. Exact verification approaches (MILP, SMT) are complete but computationally infeasible for large networks. In contrast, incomplete or relaxation-based methods (IBP, LP relaxation, CROWN) scale better but provide weaker guarantees due to over-approximation.

A central challenge stems from nonlinear activations such as ReLU, which must be approximated using linear inequalities when the activation is unstable. These relaxations allow propagation of lower/upper bounds but accumulate approximation errors as depth increases.

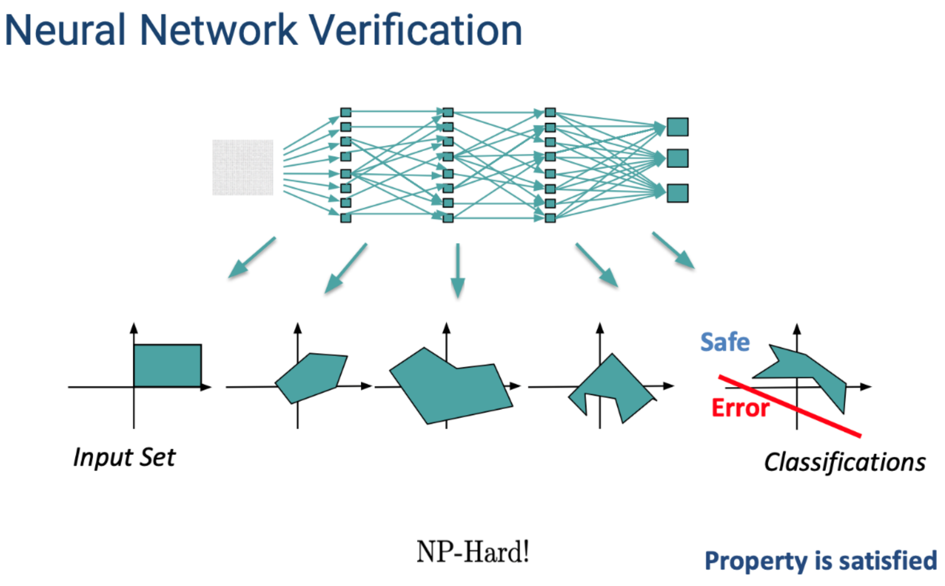


Figure 8: Neural Network Verification.

Exact verification is NP-hard and rapidly becomes intractable for deep networks or high-dimensional data. Nonlinearities, combinatorial explosion of ReLU states, and large input spaces limit real-world applicability. This motivates abstraction-based techniques, which replace nonlinear functions with sound convex relaxations, enabling propagation of approximate bounds while maintaining efficiency. However, these techniques often yield overly loose bounds, particularly in architectures with attention mechanisms or heavy matrix multiplications.

Neuro-symbolic systems combine neural perception modules (CNNs, MLPs) with symbolic reasoning engines such as Deterministic Finite Automata (DFAs). This decomposition reduces architectural complexity and allows verification at two separate levels:

1. **Neural level:** Obtain probability intervals for simple-event detectors using relaxation methods such as IBP.
2. **Symbolic level:** Propagate these intervals through DFA transitions to produce lower/upper acceptance probabilities for temporal patterns.

This approach allows the verification of *complex temporal events* by combining probabilistic bounds from multiple simple events. Experiments on MNIST-based sequence tasks show that

NeSy models drastically reduce the verification–accuracy gap (≤2%), demonstrating greater verifiability compared with monolithic temporal models.

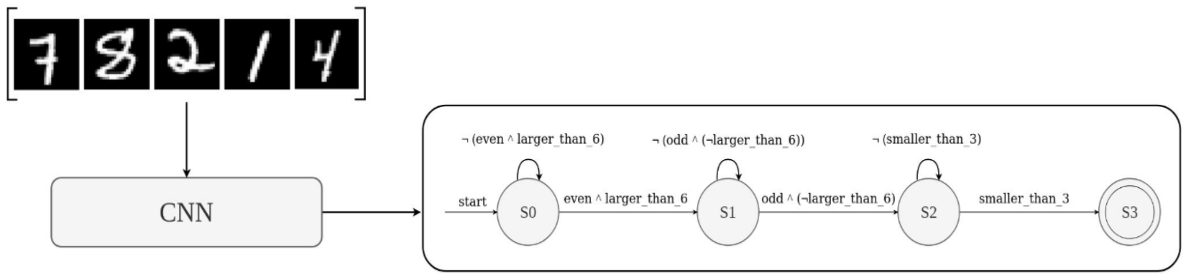


Figure 9: MNIST based sequential model for sequence classification.

Verified Training: Combining Adversarial and Formal Objectives

A major challenge is training models that are both **empirically robust** and **formally verifiable**. Standard adversarial training improves empirical robustness but degrades verified robustness. The report discusses two new expressive loss functions (CC-IBP, MTL-IBP) that jointly optimise adversarial and verified objectives. These losses achieve state-of-the-art trade-offs on datasets such as CIFAR-10, TinyImageNet, and ImageNet64, enabling models that are more suitable for deployment in safety-critical settings.

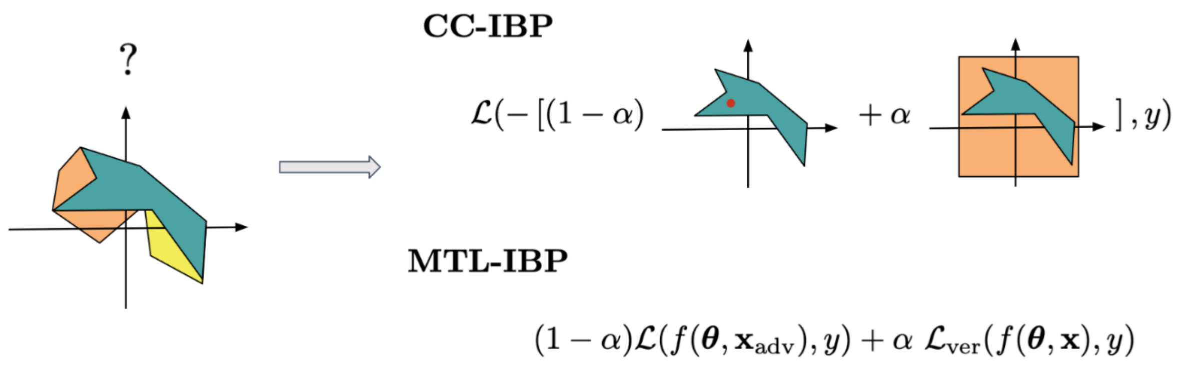

?
CC-IBP
$\mathcal{L}(-[(1-\alpha)$ $+\alpha$ $], y)$
MTL-IBP
$(1-\alpha)\mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}_{\text{adv}}), y) + \alpha\, \mathcal{L}_{\text{ver}}(f(\boldsymbol{\theta}, \mathbf{x}), y)$

Figure 10: Convex combinations between adversarial attacks and IBP bounds.

3 Scalable Neural Learning and Inference over Data Streams

3.1 Leveraging Synopses for Training Optimization – The SuBiTO Framework

The SuBiTO Framework, developed in EVENFLOW, comes in three different versions. The synopses-driven SuBiTO framework [REF-03] for synopses-based training optimization. The NeSy SuBiTO proof-of-concept for supporting entire NeSy pipelines and the Distribuito SuBiTO which leverages both synopses and parallelism across all its architectural components. In this section we initially focus on the synopses-driven SuBiTO [REF-03], which we will henceforth term as simply SuBiTO. Section 3.1.2 presents the NeSy SuBiTO and Section 3.6 details Distribuito SuBiTO.

3.1.1 The SuBiTO Architecture

The architecture of the SuBiTO framework is illustrated in Figure 11. Compared to our discussion in Section 2.1, SuBiTO performs Neural Architecture Search (NAS) besides configuring the size of ingested stream synopses and the number of epochs. In that, SuBiTO's functionality is severely enhanced because it can examine a wide variety of additional options that may involve deeper neural networks with highly compressed streaming input or shallower networks with greater number of ingested streams being processed. Additionally, the up-to-date neural models that are produced by the training pipelines are directly delivered via Kafka to a Prediction Pipeline for online, real-time inference purposes.

As shown in Figure 11, SuBiTO is composed of three architectural elements. The training pipeline, the prediction pipeline and the SuBiTO Optimizer.

The Training pipeline (middle of Figure 11) receives labelled streams and trains a neural model, for a number of epochs using only a fraction of the labelled streams specified by a stream compression ratio. These are specified by the SuBiTO Optimizer every time a concept drift is detected.

The SuBiTO Optimizer (top of Figure 11) runs every time a concept drift if detected. It accumulates a recent portion of the labelled stream and performs a number of Bayesian Optimization [REF-04][REF-05] trials to learn the accuracy vs training time trade-offs under different {stream compression ratio, neural architecture, number of epochs} configurations. At the end of the optimization process, it devices the optimal such triplet based on a scoring function, currently $score(c) = w_1 \cdot Accuracy_c + w_2 \cdot \tanh\left(\frac{training\ time_c}{target\ latency} - 1\right)$ is the default, which is deployed in the Training Pipeline.

The Prediction Pipeline processes unlabelled streams. It continuously receives the up-to-date neural model produced by the Training Pipeline and deploys it for inference purposes.

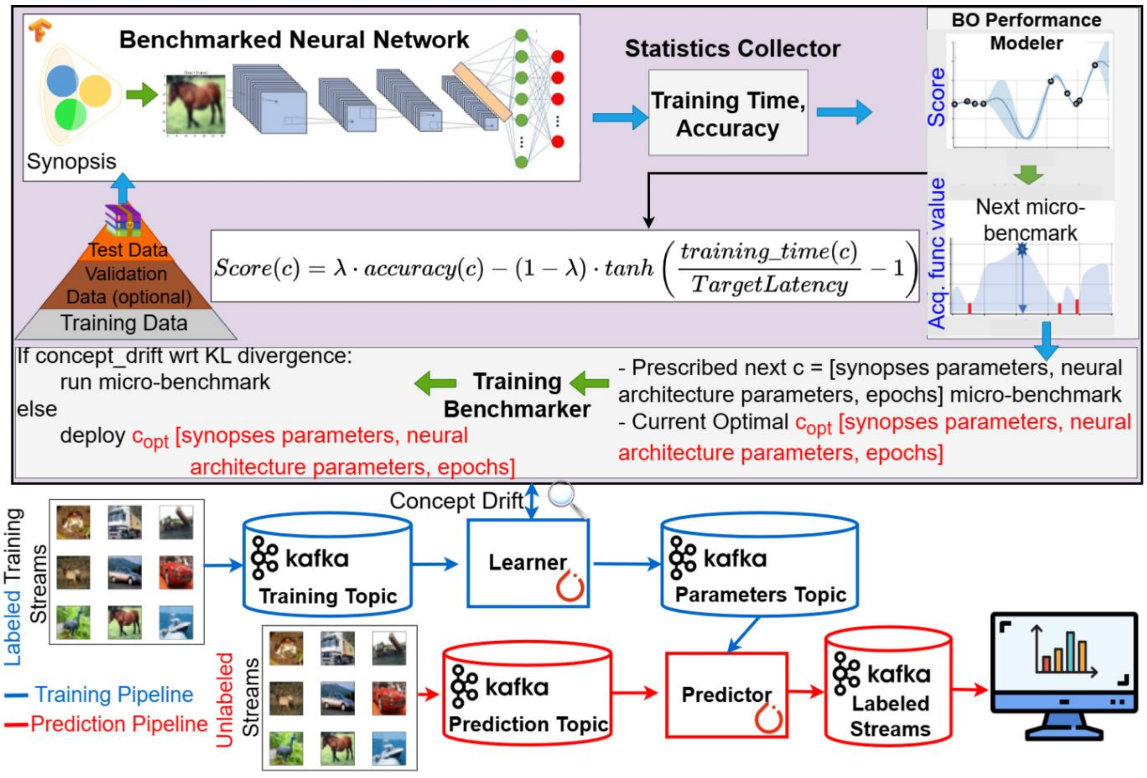
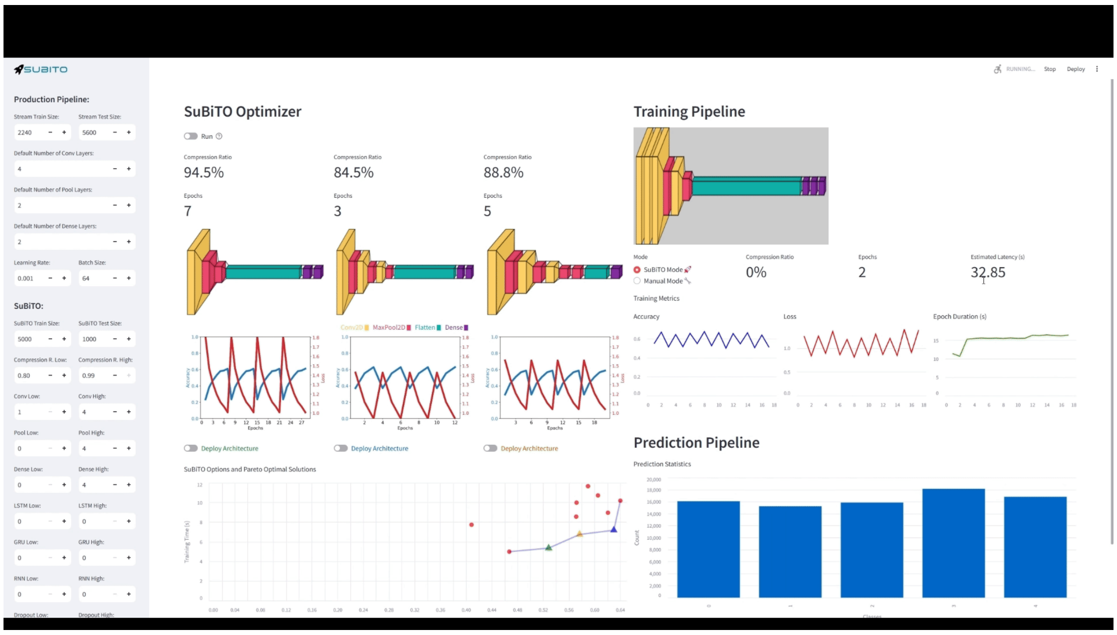

Benchmarked Neural Network
Synopsis
Statistics Collector
Training Time, Accuracy
BO Performance Modeler
Score
Next micro-bencmark
Acq. func value
Test Data
Validation Data (optional)
Training Data
$Score(c) = \lambda \cdot accuracy(c) - (1-\lambda) \cdot tanh\left(\frac{training\_time(c)}{TargetLatency} - 1\right)$
If concept_drift wrt KL divergence:
run micro-benchmark
else
deploy c_{opt} [synopses parameters, neural architecture parameters, epochs]
Training Benchmarker
- Prescribed next c = [synopses parameters, neural architecture parameters, epochs] micro-benchmark
- Current Optimal c_{opt} [synopses parameters, neural architecture parameters, epochs]
Labeled Training Streams
kafka Training Topic
Concept Drift
Learner
kafka Parameters Topic
Training Pipeline
Prediction Pipeline
Unlabeled Streams
kafka Prediction Topic
Predictor
kafka Labeled Streams

Figure 11: SuBiTO Framework Architecture [REF-03]. SuBiTO Optimizer, Training and Prediction pipelines. NAS along with configuring synopses compression ratio and number of epochs as supported participating in the examined trade-offs.



Production Pipeline:

Stream Train Size: 2240
Stream Test Size: 5600
Default Number of Conv Layers: 4
Default Number of Pool Layers: 2
Default Number of Dense Layers: 2
Learning Rate: 0.001
Batch Size: 64

SuBiTO:

SuBiTO Train Size: 5000
SuBiTO Test Size: 1000
Compression R. Low: 0.80
Compression R. High: 0.99
Conv Low: 1
Conv High: 4
Pool Low: 0
Pool High: 4
Dense Low: 0
Dense High: 4
LSTM Low: 0
LSTM High: 0
GRU Low: 0
GRU High: 0
RNN Low: 0
RNN High: 0
Dropout Low:
Dropout High:

SuBiTO Optimizer

Compression Ratio	Compression Ratio	Compression Ratio
94.5%	84.5%	88.8%
Epochs	Epochs	Epochs
7	3	5

Training Pipeline

Mode	Compression Ratio	Epochs	Estimated Latency (s)
[✓ SuBiTO Mode] Manual Mode	0%	2	32.85

Figure 12: The SuBiTO Dashboard [REF-03]. The SuBiTO Optimizer has devised three alternative Neural Network Architectures along with synopses compression ratios and number of training epochs. The user has not picked any of the devised options, therefore the training latency (right middle part of the dashboard) is in the order of tens of seconds.

3.1.2 The SuBiTO Dashboard

The SuBiTO Dashboard, implemented on streamlit[2], is illustrated in Figure 12. The vertical middle of the dashboard displays the SuBiTO Optimizer operation and suggestions. After a concept drift or a manually triggered optimization cycle, the optimizer presents the top-3 training configurations {neural network, compression ratio, number of epochs} based on a scoring function. It also plots their expected training accuracy and loss. To further aid the user cherry-pick among the top-rated alternatives, it also plots the expected accuracy, training time trade-offs of each. As soon as the user activates one among the suggested alternatives, the training pipeline is accordingly renewed as long as it finishes the current training round.

The upper right of the dashboard is devoted to the Training Pipeline. It shows the currently deployed neural network architecture, the devised compression ratio and the number of training epochs. Moreover, it plots the actual training accuracy, loss as well as the training latency throughout the training process. The lower right of the dashboard plots a histogram on stream label frequencies.

Finally, at the vertical left part of the dashboard, a panel is available for the user to specify the parameters of the available search space to be taken into account during exploration. The user can configure the stream train size, stream test size, default numbers of convolution, pooling, dense or other types of layers, plus the learning rate and batch size. In the SuBiTO dashboard the user can also choose the SuBiTO train and test sizes (portion of stream to be collected during the optimization process), the compression ratio range (low/high), the range on the number of epochs and the per-layer search ranges for architecture components: number of convolution layers (low/high), pooling layers (low/high), dense layers (low/high), as well as LSTM, GRU, vanilla RNN counts (each with low/high), and the dropout range (low/high).

3.2 A NeSy SuBiTO Proof-of-Concept

The NeSy SuBiTO proof-of-concept accepts MNIST images as inputs and attempts to pinpoint temporal sequences of numeric symbols on them. In particular, a positive sequence is one where an even number larger than 6 is observed, followed at some point by an odd number ≤ 6, followed by a number ≤ 3. Expressed as a Regular Expression: ^[0-9]*8[0-9]*[135][0-9]*[0-3][0-9]*$

Concretely, the automaton encodes:

- f(1,2) :- equals(even,1), equals(gt_6,1). → see an even & >6 digit → go to state 2.
- f(2,3) :- equals(odd,1), equals(leq_6,1). → later see an odd & ≤6 digit → go to state 3.
- f(3,4) :- equals(leq_3,1). → later see a digit ≤3 → go to state 4 (accepting state).
- Self-loops/backoffs keep the automaton in the same state when a guard is not met.

So the symbolic part is monitoring the temporal pattern: … → (even & >6) → … → (odd & ≤6) → … → (≤3) and accepts sequences reaching state 4.

[2] https://streamlit.io/

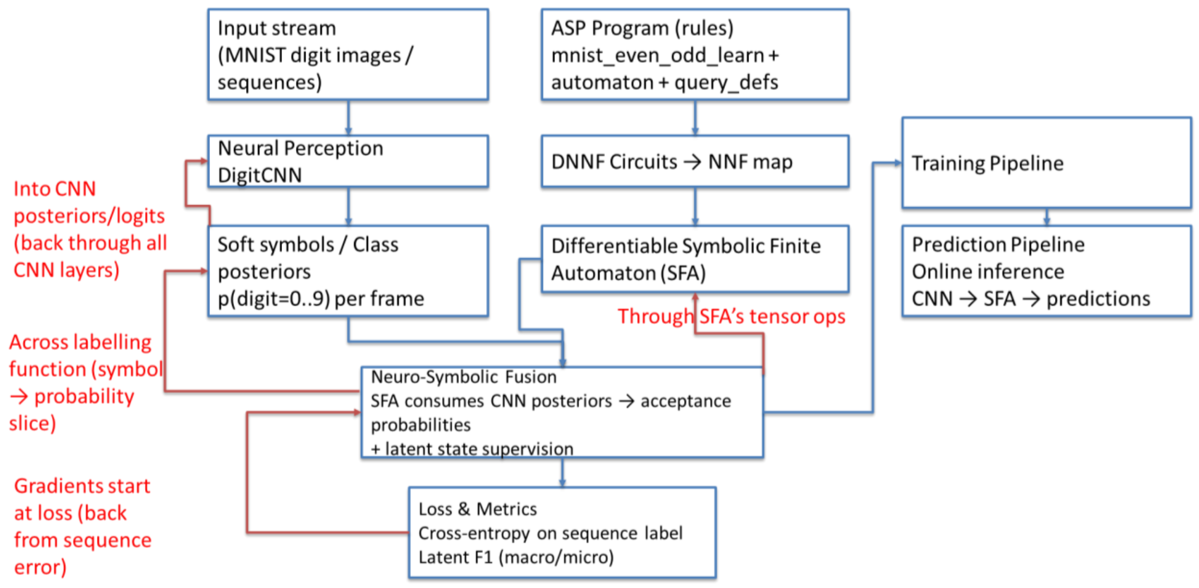

Input stream
(MNIST digit images / sequences)
ASP Program (rules)
mnist_even_odd_learn + automaton + query_defs
Neural Perception DigitCNN
DNNF Circuits → NNF map
Training Pipeline
Into CNN posteriors/logits (back through all CNN layers)
Soft symbols / Class posteriors p(digit=0..9) per frame
Differentiable Symbolic Finite Automaton (SFA)
Prediction Pipeline Online inference CNN → SFA → predictions
Through SFA's tensor ops
Across labelling function (symbol → probability slice)
Neuro-Symbolic Fusion
SFA consumes CNN posteriors → acceptance probabilities
+ latent state supervision
Gradients start at loss (back from sequence error)
Loss & Metrics
Cross-entropy on sequence label
Latent F1 (macro/micro)

Figure 13: Forward pass (blue arrows) and backpropagation (red arrows) for the NeSy SuBiTO proof-of-concept.

During the forward pass, images arrive as short sequences and are processed by a convolutional neural network (Convolutional Neural Network, CNN). The network produces, for every time step, a vector of class probabilities—the "soft symbols" that represent the likelihood of each digit. In parallel, the rules you wrote in Answer Set Programming (ASP) are compiled once into differentiable logic. Concretely, the ASP program is turned into circuits in Negation Normal Form (Negation Normal Form, NNF) by the Sentential Decision Diagram builder, and those circuits are then executed as a Symbolic Finite Automaton (SFA). The fusion step ties these two sides together: a labelling function maps each symbolic variable in the automaton to the corresponding slice of the CNN probability tensor, and the SFA evaluates the sequence to produce a single acceptance score that reflects whether the rule-defined pattern is present. That acceptance score is optionally clamped and length-decayed and then compared with the ground-truth sequence label inside the loss and metrics block, where the loss used is binary cross-entropy (BCE). For online use, the prediction pipeline simply runs CNN to SFA to acceptance score.

During backpropagation, gradients originate at the loss and flow first into the fusion step. From fusion they pass into the Symbolic Finite Automaton through its differentiable tensor operations that implement the compiled rules; the rules and circuits themselves are fixed, but the operations are differentiable so the gradient can traverse them. The gradient then crosses the labelling function, which carries it from symbolic variables back to the appropriate probability slices. Once it reaches the probability tensor, it continues into the logits and through all layers of the Convolutional Neural Network, updating only the neural weights via the optimizer. The ASP program and the Negation Normal Form circuits remain unchanged during training, while the Convolutional Neural Network is the sole learnable component.

The configurations provided by SuBiTO are as discussed in Section 3.1. Indicatively, in this scenario SuBiTO provides more than an order of magnitude reduction in training time with a negligible compromise in accuracy, lower than 5%.

3.3 Enriching the Algorithmic Foundations of Synopses-based Training Optimization

So far, we have described the functionality of SuBiTO assuming the SuBiTO optimizer runs under Bayesian Optimization and under a specific scoring function, as presented in [REF-03]. In this section we unveil the algorithmic foundation of synopses-based training optimization introducing, besides Bayesian Optimization, Exhaustive, Greedy, Evolutionary and Heuristic algorithms for the parameter space exploration by the SuBiTO optimizer. As we experimentally show, for a variety of scoring functions, these algorithms exhibit complementary characteristics with respect to the accuracy vs training time trade-offs of their proposed solutions, as well as the execution time of each algorithm. Since these algorithms are under peer-review by the time this deliverable is submitted, we here provide only algorithmic sketches of their functionality.

Exhaustive Search Algorithm: The Exhaustive Search Algorithm constructs all valid combinations of hyperparameters and layer structures up to a predefined maximum network depth. For each candidate configuration, the corresponding model is generated, trained and evaluated using the scoring function. The optimizer then selects the configuration with the highest score as the optimal solution. While Exhaustive search guarantees that the optimizer will return the global optimum configuration, its major drawback is the fact that it is computationally intensive as the search space grows exponentially with the number of parameters and possible layer types.

Greedy Search Algorithm: The Greedy Search Algorithm adopts an incremental strategy to construct and tune neural network architectures under streaming constraints. In the initial phase, the algorithm explores all valid combinations of epoch numbers, sampling rates, and first-layer types. For each candidate, it creates a simple one-layer network and evaluates it using the scoring function. The configuration achieving the highest score, reflecting the most favourable balance between computational efficiency and model accuracy is selected as the initial structure. Subsequently, once initialized, the algorithm proceeds iteratively by expanding the network one layer at a time. During each iteration, it examines all valid layer types and temporarily integrates each into the current network.

Heuristic Search Algorithm: The Heuristic Search Algorithm follows an iterative, Pareto-based strategy. Unlike Greedy and BO approaches, Heuristic operates under the principle of Pareto dominance, where a configuration is considered superior (or dominant) if no other configuration performs better in both objectives, i.e., accuracy and training time, simultaneously. The optimization process begins with an initial exhaustive search over all valid combinations of epoch counts, sampling rates, and single-layer architectures. Each configuration is trained, and its performance is evaluated with respect to accuracy and training time. These results define a two-dimensional objective space from which the Pareto front is derived. The configurations on the Pareto front will be explored further in the next iterations as promising alternatives. Next, each Pareto-optimal (from the previous step) configuration is expanded by adding one new layer. This process continues iteratively until a maximum number of iterations is reached.

Evolutionary Search Algorithm: The Evolutionary Search Algorithm, similar to the other variants, incrementally constructs the neural network by employing notions from evolutionary theory, e.g., crossovers and mutations. Generally, this variant evolves a population of configurations, i.e., triplets of neural network architectures, number of epochs and sampling rates, across successive generations, and its goal is to find configurations that 'fit' well in a setting/environment that is defined by the user's constraints. The optimization process begins with an initial exhaustive evaluation of all valid combinations of epoch counts, sampling rates, and single-layer architectures. From this set of candidates, the optimizer selects a subset of top-performing networks. These networks represent the most promising trade-offs between accuracy and training speed. Then, a subset of these elite architectures is directly 'promoted' to the next generation. The remaining configurations, which will be 'promoted' to the next generation, are produced by crossover and mutation operations applied to the current elites. This process continues until a fixed number of generations is reached.

3.3.1 Experimental Evaluation on SuBiTO algorithms

To stress test the SuBiTO algorithms we utilize two real-world datasets, namely the CIFAR10 and the UCF50 dataset [REF-06]. We examine the performance of the Bayesian Optimization, Greedy, Heuristic and Evolutionary (termed EVO) algorithms in terms of accuracy, training time trade-off of the proposed solution as well as based on their execution time.

We further alter the scoring function used while exploring the search space. Besides the scoring function discussed in Section 3.1.1, which we term 'Original', we employ the following scoring functions:

- Original Scoring Function: see Section 3.1.1.
- F1-like Harmonic Trade-off: inspired by the harmonic mean structure of the F1-score, this formulation balances accuracy and training time.
- Exponential Decay Penalty: emphasizes that the score decreases exponentially with increasing training time.
- Inverse Additive Penalty: a fraction of accuracy over normalized training time. The denominator increases with training time, gradually reducing the score without sharp penalties, unlike the Exponential Decay variant.
- Min-based Trade-off: directly captures the 'weaker' (the minimum) performance criterion among accuracy and normalized training time. As such, it benefits configurations that perform well across all desired properties, i.e., achieve high accuracy and low training latency.
- Log-Sigmoid Dominance Function: treats the difference between accuracy and normalized training time as an indicator of relative performance. This is essentially a sigmoid function which keeps the score within (0, 1) and produces smooth transitions,

i.e., models that perform slightly better in one aspect gain an adequate advantage, while large imbalances are downgraded toward the middle of the scale.

We execute our experiments under a Google Colab Pro+ subscription using the A100 GPU configuration. To show the complementarity of the various algorithms, in Figure 14, we plot the median accuracy, training time of the solution yielded by each algorithm. In that, each of the provided features in Figure 14, holds for at least half of the cases of scoring functions in a per dataset fashion. Relative execution time is encoded in the figure by marker area and labelled by its multiple vs. the fastest method in the same dataset (1× = fastest).

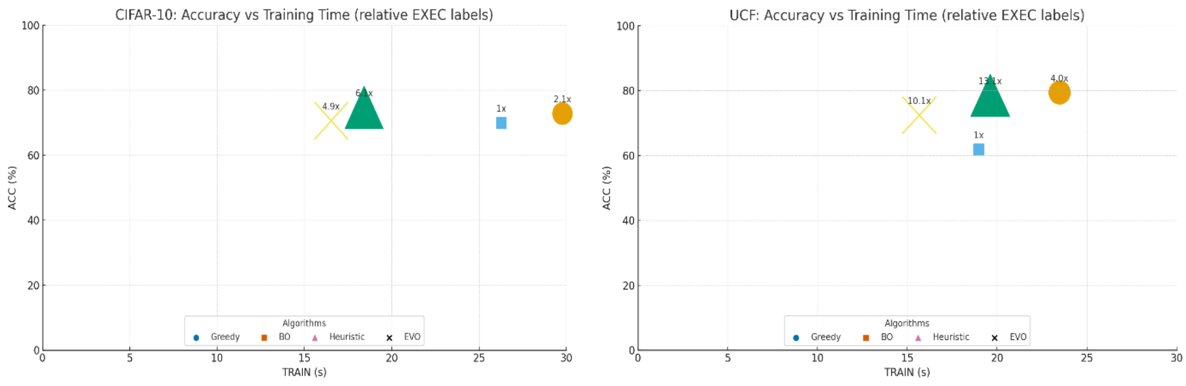


Figure 14: Median performance of SuBiTO algorithms across scoring functions for the CIFAR10 and UCF datasets. Relative execution time is encoded by marker area and labelled by its multiple vs. the fastest method in the same dataset (1× = fastest).

The complementarity of the algorithms is justified by the following observations:

Bayesian Optimization (BO) is always the fastest in terms of execution time. It is therefore the best solution in cases of highly volatile data streams which yield frequent concept drifts and, therefore, frequent optimization decisions. However, the accuracy vs training time trade-off that it provides is inferior compared to the other alternatives, especially in the, more demanding, video stream scenario.

Greedy provides solutions of good accuracy but it yields the highest training times. The training time of Greedy is up to 2x worse than the best training time induced by EVO. In other words, Greedy can keep high accuracy levels but moderately compromises the real timeliness of the training pipeline of SuBiTO. Its execution time is the second best, though it remains 2x and up to 4x slower than BO. It is suitable in case high accuracy and fast optimization times for volatile data streams are prioritized higher than rapid training times.

EVO provides solutions of slightly worse (up to 7%) accuracy compared to the algorithm that provides the most accurate solutions across these datasets, but it yields the fastest training times. Therefore, it ensures the real timeliness of the training process. The caveat is that its execution time is up to 10x times slower than BO and up to 2.5x slower compared to Greedy. It is therefore suitable for settings where rapid training times and high accuracy are of the essence, but with lower data stream volatility and rarer concept drifts.

Heuristic yields the best solutions in terms of execution vs training time trade-offs. It often provides the best accuracy with only 15-20% higher training time compared to EVO. Its execution time though is the highest among all algorithms. Heuristic is an order of magnitude slower than BO and 3-4 times slower than Greedy. It is also 20-25% slower than EVO.

Overall, our experimental evaluation results in the following main findings:

- Highly Volatile Streams, real timeliness of training pipeline àBO
- High/Moderately Volatile Streams, prioritization on accuracy over training time à Greedy
- Non-volatile Streams, real timeliness of training pipeline à EVO
- Non-volatile Streams, prioritization on accuracy over training time à Heuristic

3.4 Parallel Synopses Maintenance Revisited

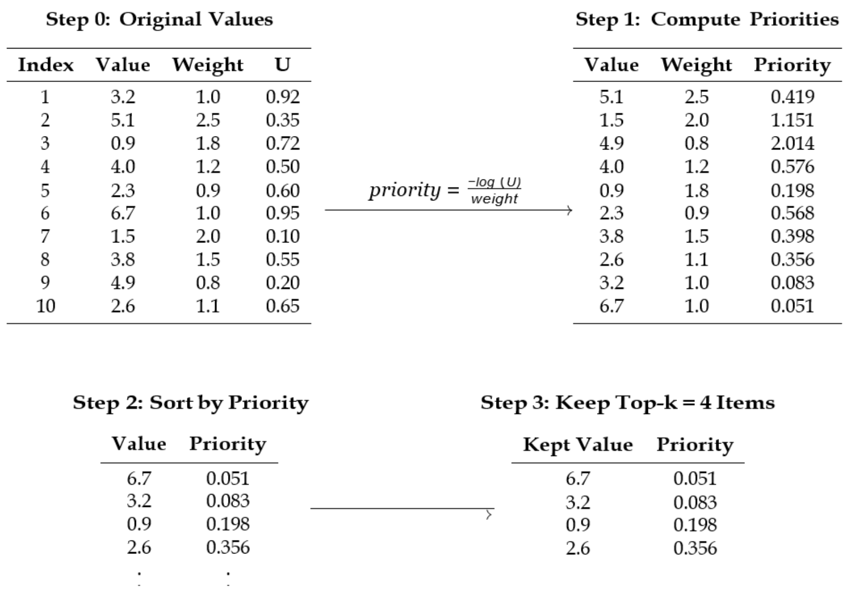
However novel and practical SDEaaS [REF-02] was, it had an important limitation: it emitted synopses via Apache Flink's streaming APIs (e.g., DataStream/Table) that were not natively tensor-compatible for neural training. Each time new synopses were produced, they first had to be written to Kafka and then loaded into tensor-friendly structures (e.g., Pandas DataFrames or NumPy ndarrays) for the training pipeline. This extra hop can undermine the benefits of parallel synopsis maintenance, because the conversion/ingestion steps relied on tools that are not inherently distributed. To remove this bottleneck, we re-implemented the SDE on Dask[3], enabling parallel synopsis maintenance with direct tensor-compatibility, eliminating interleaving format transformations and preserving the potential for end-to-end parallel training.

We further incorporated an inherently parallel synopsis technique, namely the (Weighted) Priority Sampling Scheme introduced in [REF-07]. The Weighted Priority Sampler (WPS) is a **parallel** streaming sampling algorithm designed to efficiently maintain a representative subset of items from a continuous data stream. It keeps the k items with the smallest priorities (implemented with a min-heap), where each item's priority is defined as:

$$priority = \frac{\log(U)}{weight}$$

Here, U is a uniformly distributed random variable in the interval (0, 1), and weight is a user-defined importance score for each item. This formulation ensures that higher-weight items have a higher chance of being included in the sample, while also introducing randomness to prevent bias. If all items are assigned equal weights, the sampling scheme reduces to uniform sampling. Figure 15 illustrates the operation of the WPS on a single worker.

[3] https://www.dask.org/



Step 0: Original Values

Index	Value	Weight	U
1	3.2	1.0	0.92
2	5.1	2.5	0.35
3	0.9	1.8	0.72
4	4.0	1.2	0.50
5	2.3	0.9	0.60
6	6.7	1.0	0.95
7	1.5	2.0	0.10
8	3.8	1.5	0.55
9	4.9	0.8	0.20
10	2.6	1.1	0.65

$$priority = \frac{-log\ (U)}{weight}$$

Step 1: Compute Priorities

Value	Weight	Priority
5.1	2.5	0.419
1.5	2.0	1.151
4.9	0.8	2.014
4.0	1.2	0.576
0.9	1.8	0.198
2.3	0.9	0.568
3.8	1.5	0.398
2.6	1.1	0.356
3.2	1.0	0.083
6.7	1.0	0.051

Step 2: Sort by Priority

Value	Priority
6.7	0.051
3.2	0.083
0.9	0.198
2.6	0.356
⋮	⋮

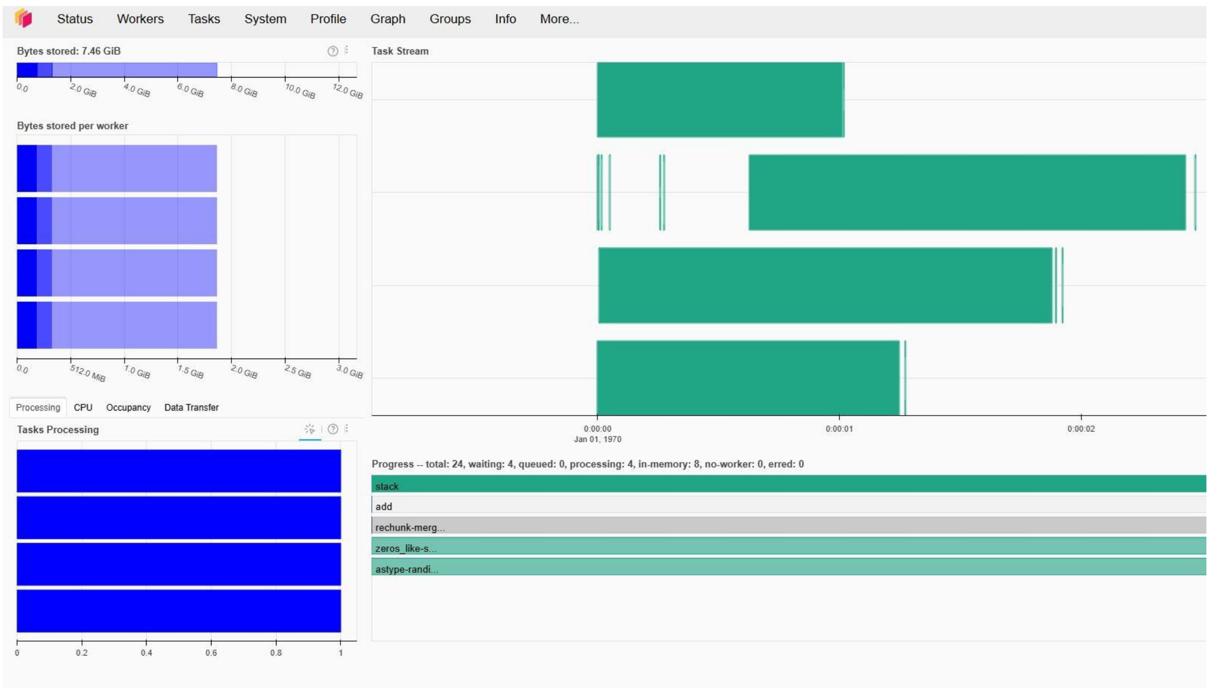
Step 3: Keep Top-k = 4 Items

Kept Value	Priority
6.7	0.051
3.2	0.083
0.9	0.198
2.6	0.356

Figure 15: Operation of the Weighted Priority Sampling on a single worker.

Each worker applies the procedure of Figure 15 on its local stream using an add operation (see [REF-02]). The overall sample is formed in a merge step (see [REF-02]) by choosing the overall top-K tuples. The following code snippet sketches the operation of WPS in Dask, while Figure 16, illustrates the parallel execution of priority sampling on 4 workers as presented at the Dask Dashboard.

```
# Map over each Dask array block, applying `self.add` to produce per-block results.
# - `.compute()` gathers all per-worker outputs to the driver (in-memory) as a NumPy array.
top_ks_splitted = stream.map_blocks(self.add, dtype=stream.dtype).compute()
# Merge the per-block results into a single global result using your sampler’s merge logic.
merged_PS = self.merge(top_ks_splitted)
```



Progress -- total: 24, waiting: 4, queued: 0, processing: 4, in-memory: 8, no-worker: 0, erred: 0

Figure 16: Priority Sampling parallel execution as illustrated at Dask Dashboard under 4 workers.

WPS is a highly useful parallel sampling scheme that is utilized in the SuBiTO optimizer and across SuBiTO's training pipelines as described in Section 3.6.

3.5 Data-driven Synchronization and the EVENFLOW Protocol Suite

As noted in Deliverable D5.1, the aim of the smart synchronization protocols for distributed/parallel neural learning, introduced in EVENFLOW is to provide data-driven synchronization mechanisms. The vanilla synchronous and asynchronous protocols entailed in the Parameter Server paradigm [REF-08][REF-09], trigger laggy, full synchronizations on predefined rounds (synchronous) or allow partial, inaccurate synchronizations (asynchronous). The EVENFLOW protocol suite [REF-01], instead, requires a sync only when a concept drift may have occurred based on data driven criteria.

In EVENFLOW, the application provides any thresholded function f(w(t)): R^dàR and a threshold T. w(t) corresponds to the global neural network weights at time t. A global sync is required only when f surpasses the given threshold T. EVENFLOW decomposes this global trigger to local tests which can be individually checked by each parallel learner locally, without communicating with its peers. To achieve that, EVENFLOW employs a geometric approach which, instead of distributively monitoring the function value, it monitors the input domain of w(t) and their gradients.

More formally, we consider n learners i ∈ {1,…,n} training a shared model with weight vector $w(t) \in R^d$. Learner i computes a local gradient $g_i^{(t)} = \nabla_w L_i\big(w(t)\big)$ Local steps between two synchronizations are indexed by κ = 0,1,…. An application-supplied monitoring function $f: R^d \times R^d \to R$ with tolerance T > 0 decides whether a global synchronization is required: a sync is needed iff $f\big(w(t+\kappa+1), w(t)\big) > T$.

3.5.1 The Basic EVENFLOW Synchronization Protocol

Local updates (per learner i): The learner performs standard gradient steps with learning rate η > 0:

$$w_i(t+\kappa+1) = w_i(t+\kappa) - \eta \cdot g_i^{(t+\kappa)} = w(t) - \eta \cdot \Sigma_{r=0}^{\kappa} g_i^{(t+r)}$$

Geometric filter (distributed, no communication unless needed): Define the alarm set $A = x \in R^d: f\big(x, w(t)\big) > T$. Each learner i forms a d-dimensional sphere $B_i = B(c_i, r_i)$ with:

c $_i = w(t) - (\eta/2) \cdot \Sigma_{r=0}^{\kappa} g_i^{(t+r)}, r_i = (\eta/2) \cdot \left\|\Sigma_{r=0}^{\kappa} g_i^{(t+r)}\right\|$.

If $B_i \cap A = \emptyset$, the learner stays silent; if $B_i \cap A \neq \emptyset$, it requests a synchronization. If no learner requires a sync, local training continues (no global round completion).

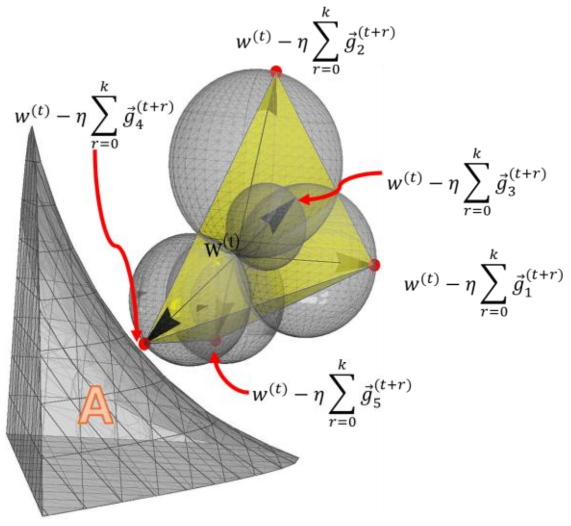

$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_2^{(t+r)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_4^{(t+r)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_3^{(t+r)}$
$w^{(t)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_1^{(t+r)}$
$w^{(t)} - \eta \sum_{r=0}^{k} \vec{g}_5^{(t+r)}$
A

Figure 17: Basic EVENFLOW Protocol Rationale. Since no sphere intersects A, no sync is triggered.

Global aggregation (upon any sync): The aggregator computes a weighted average of current local models and broadcasts it:

$$w(t+\kappa+1) = \big(\Sigma_i \gamma_i \cdot w_i(t+\kappa+1)\big)/(\Sigma_i \gamma_i), \quad \text{with } \gamma_i > 0.$$

γ_is can quantify cases of learners that receive inequivalent number of tuples, therefore, their participation in the global average is weighted accordingly. Finally, all learners set w ← w(t+κ+1), reset κ ← 0, and continue.

Correctness: The global model lies in the convex hull of $w(t) - \eta \Sigma_{r=0}^{\kappa} g_i^{(t+r)}{}_i$, which is contained in $\bigcup_i B_i$. Therefore, if each B_i is disjoint from A, then f(w(t+κ+1), w(t)) ≤ T. An intersection on any learner indicates a potential threshold crossing and triggers a synchronization (potentially false positive).

A fundamental limitation of the Basic EVENFLOW protocol is that in neural learning, the number of weights may well reach the order of millions. Having learners perform sphere intersection checks against A in case of high d values may slow down, instead of benefitting, the training process. To overcome this limitation, we introduce the Fast EVENFLOW protocol which utilizes only m << d important values, based on Discrete Fourier Transform (DFT) coefficients of the local weight vectors, to speed up local intersection checks.

3.5.2 The Fast EVENFLOW Synchronization Protocol

The Fast EVENFLOW protocol operates on offsets from the last global model computed during a synchronization: $\Delta w(t+\kappa+1) = w(t+\kappa+1) - w(t)$. The monitoring task now becomes $f(\Delta w(t+\kappa+1), 0) > T_{tr}$ for an equivalent but transformed threshold T_{tr}. For each learner i, Fast EVENFLOW computes the DFT of the cumulative gradients $G_i^{(\kappa)} = \Sigma_{r=0}^{\kappa} g_i^{(t+r)}$. We denote $\tilde{G}_i^{(\kappa)} = DFT\left(G_i^{(\kappa)}\right)$, and retain only the top-m (m ≪ d) coefficients by magnitude: $\tilde{G}_{i,m}^{(\kappa)}$. The linearity of DFT ensures $DFT\left(\Sigma_r g_i^{(t+r)}\right) = \Sigma_r DFT\left(g_i^{(t+r)}\right)$.

Reduced-space filter (per learner): We Define the m-dimensional sphere $\tilde{B}_i = B(\tilde{c}_i, \tilde{r}_i)$ with:

$$\tilde{c}_i = -(\eta/2) \cdot \tilde{G}_{i,m}^{(\kappa)}, \tilde{r}_i = (\eta/2) \cdot \left\|\tilde{G}_{i,m}^{(\kappa)}\right\|.$$

Intersection of $\tilde{B}_i$ with the transformed alarm set $A_{tr} = x: f(x,0) > T_{tr}$ triggers a sync. Otherwise, the learning step stays local. Aggregation/broadcast proceed as in Basic EVENFLOW.

Deterministic quality bounds. Using Parseval's identity and highest-m reduction, the time-domain approximation error satisfies (IDFT is the Inverse Discrete Fourier Transform):

$$\left\|G_i^{(\kappa)} - IDFT\left(\tilde{G}_{i,m}^{(\kappa)}\right)\right\| \le \sqrt{(\kappa+1)} \cdot \sqrt{(1-m/d)} \cdot max_r \left\|g_i^{(t+r)}\right\|.$$

Thus, the center and radius in the reduced space approximate those of the Basic protocol within (η/2) times the bound above, which upper-bounds both center displacement and radius approximation due to dimensionality reduction.

3.5.3 Handling Sliding Windows

In streaming settings, a sliding window model is often used in order to account for the most recent observations, while mark older observations as expired obsolete. The Fast EVENFLOW protocol can be trivially extended to make synchronization decisions depend only on the most recent W global steps, rather than all steps since the last global sync.

Windowed cumulative gradients (per learner i): We define $S_i^{(t)} = \Sigma_{r=t-W+1}^{t} g_i^{(r)}$ and its DFT $\tilde{S}_i^{(t)} = \Sigma_{r=t-W+1}^{t} \tilde{g}_i^{(r)}$. The window-slide recursion is exact due to DFT linearity: $\tilde{S}_i^{(t+1)} = \tilde{S}_i^{(t)} - \tilde{g}_i^{(t-W+1)} + \tilde{g}_i^{(t+1)}$. Each learner maintains $\tilde{S}_i^{(t)}$ and keeps only the top-m coefficients ${\tilde{S}_i^{(t)}}_{i,m}^{(t)}$.

Windowed reduced-space filter: We replace $\tilde{G}_{i,m}^{(\kappa)}$ with $\tilde{S}_{i,m}^{(t)}$ in the Fast EVENFLOW test:

$\tilde{B}_i^{win} = B\left(\tilde{c}_i^{win}, \tilde{r}_i^{win}\right)$, where $\tilde{c}_i^{win} = -(\eta/2) \cdot \tilde{S}_{i,m}^{(t)}, \tilde{r}_i^{win} = (\eta/2) \cdot \left\|\tilde{S}_{i,m}^{(t)}\right\|$.

If $\tilde{B}_i^{win} \cap A_{tr} \neq \emptyset$, learner I requests a sync; else it keeps up local training. Aggregation upon a sync remains unchanged.

Error and correctness: The same deterministic bound applies with κ+1 replaced by W, i.e., $(\eta/2) \cdot \sqrt{W} \cdot \sqrt{(1-m/d)} \cdot max_{r\in[t-W+1,t]} \left\| g_i^{(r)} \right\|$. Therefore, decisions in reduced space track those in full space up to the stated bound.

Complexity (per step, per learner). Compute in *O(dℓog(d)) FFTs and* reuse FFTs of per-batch gradients; update the sliding DFT via the *O(d)* recursion above; maintain top-m coefficients (selection/heap); run the m-dimensional sphere check. After caching $\tilde{g}_i$, monitoring is essentially *O(m)* per step.

3.6 Distribuito SuBiTO: Synopses, Parallelism & Smart Sync All in One

The fully distributed version of SuBiTO incorporates the EVENFLOW advancements described in Section 3.4 and Section 3.5.2. The architecture of Distribuito SuBiTO is illustrated in Figure 18. The entire implementation of Distribuito SuBiTO is on Dask for parallel synopses maintenance and Ray[4] for distributed/parallel training and inference purposes.

As shown in Figure 18, data streams arrive partitioned across Kafka topics. Parallel synopses maintenance over Dask (Section 3.4) plays a role at both the SuBiTO optimizer and the SuBiTO training pipeline. At the SuBiTO Optimizer side, Distribuito SuBiTO examines different parallelization degrees on par with NAS, compression ratio and epoch numbers. Bayesian Optimization is still used here since, as shown in Section 3.3.1, it is the most preferable option for highly volatile streams, across scoring functions.

The solution {parallelism, Neural Architecture, epoch number, compression ratio} returned by BO is conveyed to the Training Pipeline at runtime. This then becomes the currently deployed configuration. That is, the training pipeline now dynamically configures the parallelism of synopses maintenance at runtime.

At both the SuBiTO Optimizer and the Training Pipeline, Fast EVENFLOW (Section 3.5.2) is adopted, across the parallel learners, to provide rapid training times, reducing inter-learner communication and maintaining high accuracy.

Finally, at the Prediction Pipeline, unlabelled streams are consumed by parallel predictors. Again, the most up-to-date global neural model extracted by the Training Pipeline is transferred to the parallel predictors via Kafka. Based on this updated model, the parallel predictors label the streaming tuples belonging to their assigned partitions. Besides parallelism, predictors incorporate an important optimization that boosts throughput (number of tuples being labelled per second). The talk involves prediction batching. As we experimentally show in Section 3.6.1, throughput is dramatically increased when – instead of labelling streaming tuples one by one – streaming tuples are first organized into batches and

[4] https://www.ray.io/

are labelled all together. This optimization reduces data exchanges between RAM and GPU RAM, therefore boosting throughput.

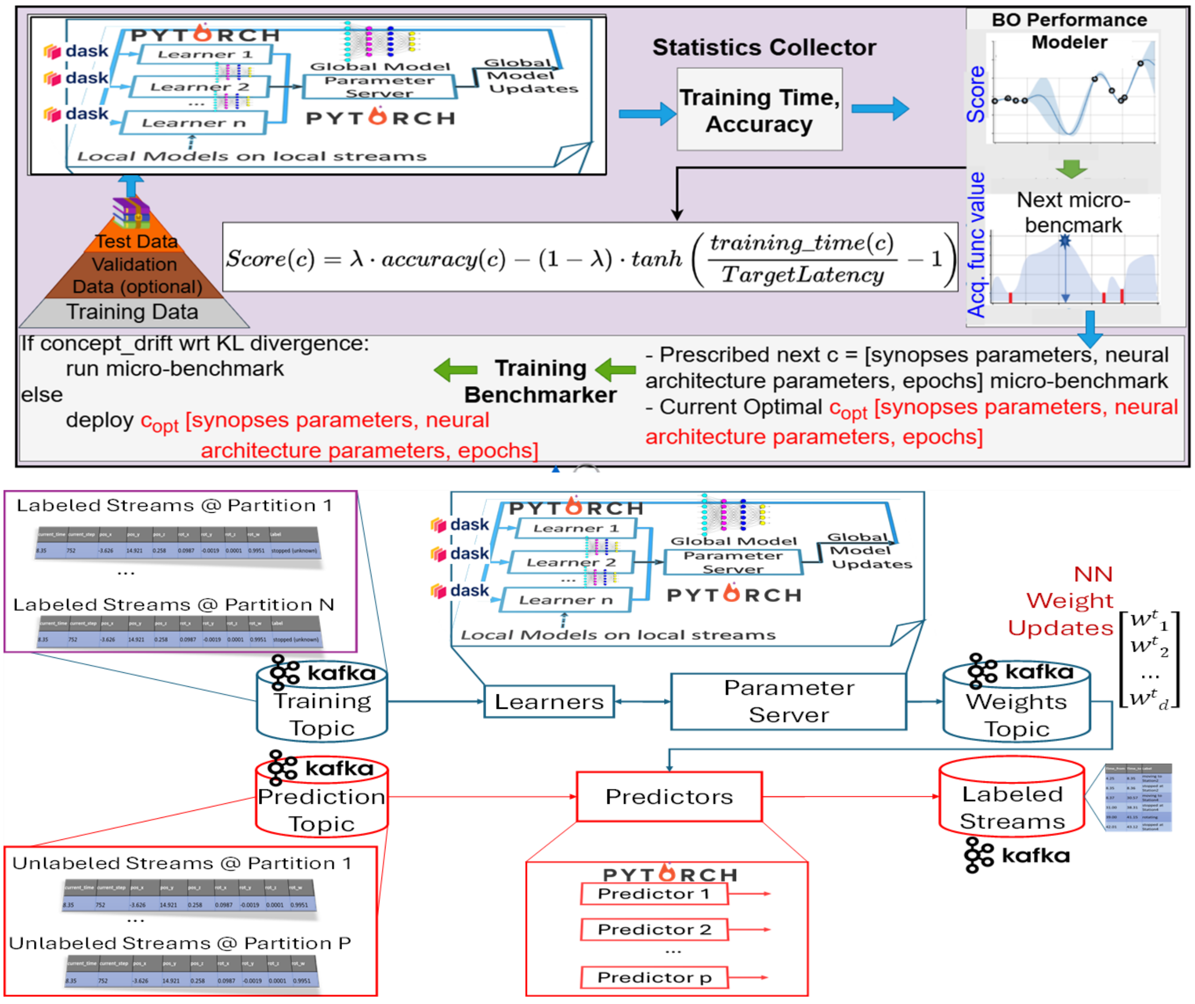

dask
Learner 1
Learner 2
Learner n
Global Model Parameter Server
Global Model Updates
PYTORCH
Local Models on local streams
Statistics Collector
Training Time, Accuracy
BO Performance Modeler
Score
Next micro-bencmark
Acq. func value
Test Data
Validation Data (optional)
Training Data
$Score(c) = \lambda \cdot accuracy(c) - (1-\lambda) \cdot tanh\left(\frac{training\_time(c)}{TargetLatency} - 1\right)$
If concept_drift wrt KL divergence:
run micro-benchmark
else
deploy c_{opt} [synopses parameters, neural architecture parameters, epochs]
Training Benchmarker
- Prescribed next c = [synopses parameters, neural architecture parameters, epochs] micro-benchmark
- Current Optimal c_{opt} [synopses parameters, neural architecture parameters, epochs]
Labeled Streams @ Partition 1
Labeled Streams @ Partition N
kafka
Training Topic
Learners
Parameter Server
Weights Topic
NN Weight Updates
$\begin{bmatrix} w^t{}_1 \\ w^t{}_2 \\ \dots \\ w^t{}_d \end{bmatrix}$
Prediction Topic
Predictors
Labeled Streams
Unlabeled Streams @ Partition 1
Unlabeled Streams @ Partition P
Predictor 1
Predictor 2
Predictor p

Figure 18: Distribuito SuBiTO Architecture. Synopses are derived using SDE on Dask (Section 3.4). The Fast EVENFLOW protocol (Section 3.5.2) is incorporated for smart, data-driven synchronization. Parallel predictors and prediction batching used to boost inference speed.

3.6.1 Experimental Evaluation of Distribuito SuBiTO

The experiments conducted regarding the distributed SuBiTO are based on the three main parts of the framework: synopses construction, training and inference. We start by discussing the experiments on the synopses' construction part of the framework, focusing on parallel sampling scalability. We continue by presenting the experiments carried out on the new distributed Training Pipeline, using the novel Fast EVENFLOW protocol, under the PS paradigm. The experiments regarding the Prediction Pipeline are presented in the end, testing the scalability of the inference mechanism, by examining the throughput of its new version.

3.6.1.1 Synopses Scalability

The experiments presented in this section focus on testing the new scalable sampling approach of the Distribuito SuBiTO framework, using Priority Sampling on Dask and assigning random weights on each data point (Section 3.4). On this experiment we stream the NSFW

Dataset (available from HauggingFace) through Kafka and test the sampling execution time on both the original SuBiTO framework, which uses stratified sampling over NumPy, and the Distribuito SuBiTO framework, which uses Weighted Priority Sampling over Dask.

The original SuBiTO performs sampling centrally using NumPy, while the Distribuito SuBiTO uses 6 Dask Workers, performing sampling on Dask. Recall that 6 Dask Workers are used only for experimental purposes, during the deployment of SuBiTO the number of Dask Workers is set by the SuBiTO Optimizer. In order to compare the two approaches, we chose to experiment using sample sizes of 20%, 40%, 60% and 80% of varying dataset size. On each experiment the dataset is duplicated 25, 30, 35, 40, 45 and 50 times for stress testing SuBiTO and Distribuito SuBiTO on various stream volumes. The sampling times reported are the mean across 5 trials, performed on the same computation hardware.

On this experiment we test both sampling approaches on a single machine. It should be noted that NumPy offers no way of scaling on a large cluster. On the contrary, Dask offers the ability to scale our application on large clusters, leveraging the computing power of multiple processors on multiple servers.

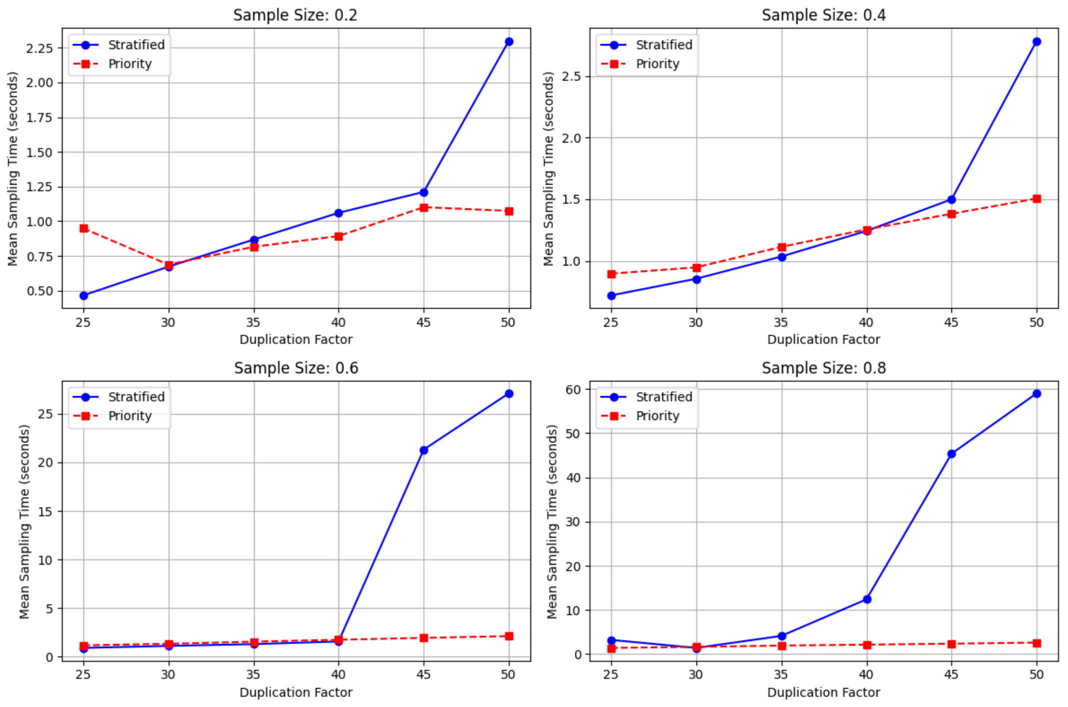


Figure 19: Synopses Scalability, original SuBiTO uses Stratified Sampling on NumPy while Distribuito SuBiTO uses Priority Sampling on Dask with a parallelism of 6.

As we can see in Figure 19, when the dataset can fit in memory of a machine, the original approach of stratified sampling on NumPy achieves a quicker execution time compared to the Dask approach, on nearly all sample size cases. This trend can be observed on the duplication factors of 25 and 30. This behaviour is expected as NumPy is very efficient when computing within the limits of system memory, compared to Dask which introduces unnecessary computation overhead (Task Graph building, Execution Optimization, etc.). When the dataset is extended 35 times, stretching the system's memory limits, we observe that NumPy needs more time for sampling, compared to Dask, for 40% sample size. As the datasets duplication factor grows from 40 to 45 and 50 times, surpassing the system's memory limits, it becomes evident that Dask's sampling time scales linearly, compared to NumPy's exponential scaling. In the particular case of duplicating the dataset 50 times, we observe that NumPy needs twice the time for sampling on 20% sample size, compared to Dask and 60 times as long in the case of 80% sampling size.

The observations made above show clearly that using Priority Sampling on Dask, gives the Distribuito SuBiTO framework's sampling approach the ability to handle large data streams with ease and therefore maintain the real-time character of the application on both medium and large scale environments.

3.6.1.2 Training Scalability

This section includes the experimental evaluation of the Distribuito SuBiTO Training Pipeline. This set of experiments focuses on testing the Distribuito SuBiTO learning approach of using the PS paradigm and the Fast EVENFLOW synchronization protocol, across multiple Learner instances. Therefore, the execution times reported in this section denote only the time that the set of Learners needed for training, excluding stream ingestion and sampling. Our results are compared to the Distribuito SuBiTO framework using the vanilla synchronous protocol under the PS paradigm. Our tests vary the number of Learners between 2 and 4 and conduct the experiments using the NSFW Image Dataset[5] and a subset of the UCF50 Video Dataset, containing 10 out of the 50 classes. Moreover, we report the number of the PS's synchronizations invoked on each case, in order to examine the overall communication overhead of our approach. The Datasets are continuously streamed on Kafka on a loop. Our training limit is 1400 batches on the NSFW Dataset and 4000 batches on the UCF Dataset [REF-06], essentially letting each learning approach process the entire data stream two times. After reaching the batch processing limit, deducted only when the Learners synchronize with the PS, each learning process is terminated. On both approaches we use a learning rate of 10^{-3} and batch sizes of 32 and 10, on the NSFW Dataset and UCF Datasets, respectively. Regarding the Fast EVENFLOW Protocol, our threshold decay function is set empirically, through careful examination of the learning process and the loss function, with the aim to stabilize learning throughout the batch processing:

$T = 2\,e^{-0.4\,current\_round} + 0.15$, for the NSFW dataset,

$T = 64\,e^{-0.2\,current\_round} + 0.5$

We start by observing the results of the NSFW Dataset, with 2 Learner instances, using the vanilla Synchronous protocol and the Fast EVENFLOW synchronization protocol, in Figure 20 and Figure 21, respectively. As we can see both synchronization protocols achieve high test and train accuracy scores, indicating that both the Fast EVENFLOW and the Synchronous protocols achieve adequate model generalization.

[5] https://huggingface.co/datasets/deepghs/nsfw_detect

Comparing the test accuracy metric of both synchronization techniques, the Fast EVENFLOW protocol only lacks on test accuracy by 4% shining in terms of both reduced training time and number of synchronizations. The number of synchronizations is perceived here as a measure of communication cost between the workers and the Parameter Server. The training time achieved by the Fast EVENFLOW protocol is 4 times less, compared to the synchronous protocol, while the number of synchronizations is reduced from 701 to only 7.

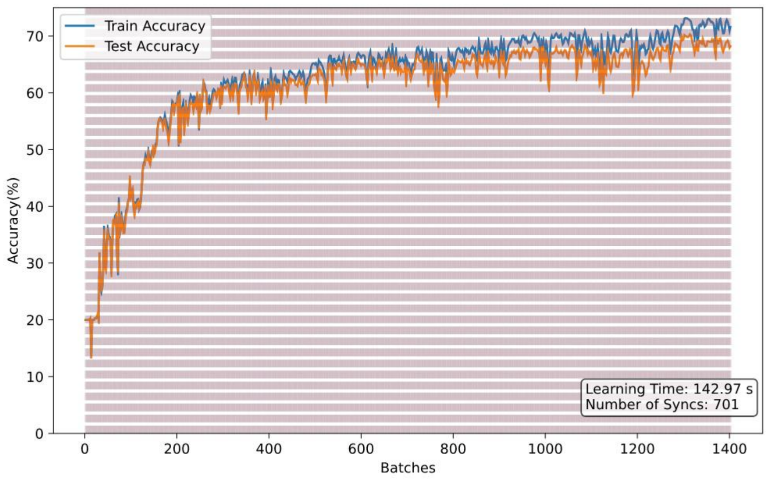


Figure 20: Synchronous Protocol using 2 Learners on the NSFW Dataset.

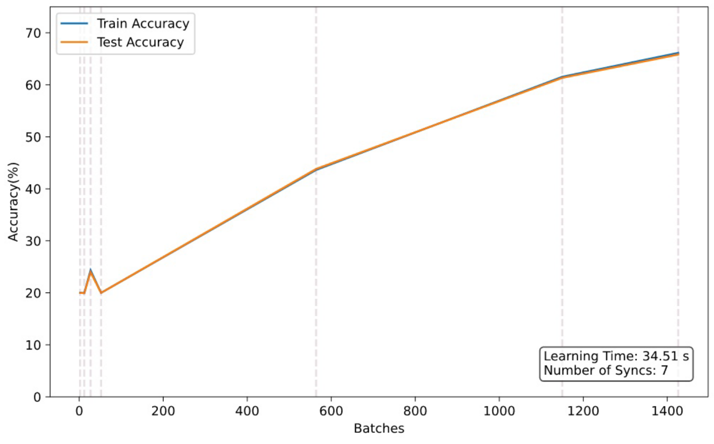


Figure 21: Fast EVENFLOW Protocol using 2 Learners on the NSFW Dataset.

We continue by examining the same experimental setup, with the only exception of using 4 Learners this time. As we can see in Figure 22 and Figure 23 regarding the Synchronous and Fast EVENFLOW synchronization protocols, we observe similar results. Both approaches achieve equally high test accuracy scores, generalizing their models adequately. The training time of the synchronous protocol is reduced to 92 seconds (s), compared to the 143s needed on the 2 Learner approach, while the training time of Fast EVENFLOW is decreased only slightly going from 35s to 31s. The use of more Learners did not benefit the FAST EVENFLOW approach on the number of synchronizations but reduced the synchronizations of Synchronous in half. This training time of the Fast EVENFLOW protocol is a consequence of processing more than 2000 batches, compared to the 1400 batches that the synchronous approach processes on the same experiment. As we mentioned earlier the NSFW Dataset contains 700 batches and is streamed on a loop, though due to the nature of the Fast EVENFLOW protocol we are able to deduct the number of batches that are processed by the Learners, only when the synchronization process is initiated. Therefore, we stop the training process when the Learners synchronize with the PS and we have processed **at least** 1400 batches, leaving the experiments regarding the Fast EVENFLOW protocol, always processing more batches compared to the synchronous approach. But the important finding resides to a different, more important observation.

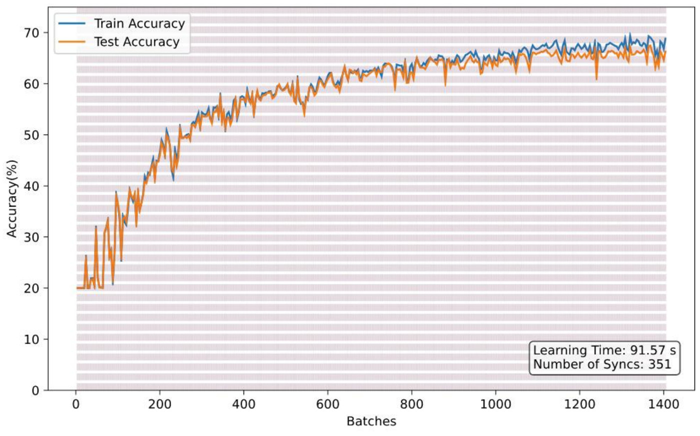


Figure 22: Synchronous Protocol using 4 Learners on the NSFW Dataset.

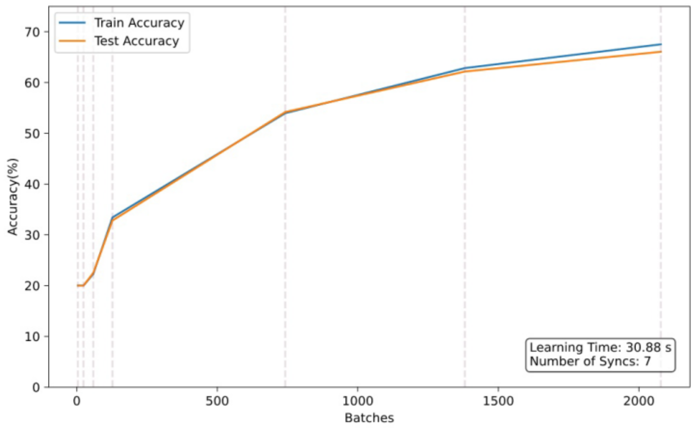


Figure 23: Fast EVENFLOW Protocol using 4 Learners on the NSFW Dataset.

The Fast EVENFLOW synchronization protocol using 2 Learners (Figure 21) provides the same accuracy as the Synchronous protocol using 4 Learners (Figure 22) while reducing the training time by a factor of 3 and the number of synchronizations by 2 orders of magnitude, all while using less hardware, specifically half the number of Learners and therefore half the number of GPUs. Regarding the communication overhead, Fast EVENFLOW transmits 21 messages during the training process while the synchronous approach transmits 1755, reducing drastically the communication cost of the training process. This indicates that the Fast EVENFLOW protocol can indeed provide a scalable solution to perform Distributed Learning on the Distribuito SuBiTO platform, while maintaining the real-time character of the application, through reduced training time and communication cost, with half the hardware requirements.

We continue the experimental evaluation by presenting the experiments conducted using the UCF Dataset. The UCF Dataset is larger in size compared to the NSFW Dataset used in the previous experiments both in terms of data items and item size, as videos carry much more information compared to images. Therefore, we utilized the Ray Store library, essentially sharing the data across the Actors (Learners), minimizing the memory footprint of the training process. This resulted in a memory efficient training process, which indeed impacted the training time. Given this remark, it is evident that there should be no comparison of the training times between the experiments conducted with the NSFW and the UCF datasets.

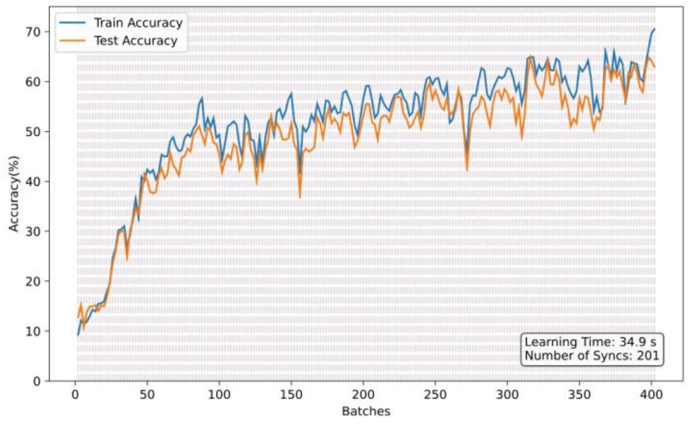


Figure 24: Synchronous Protocol using 2 Learners on the UCF Dataset.

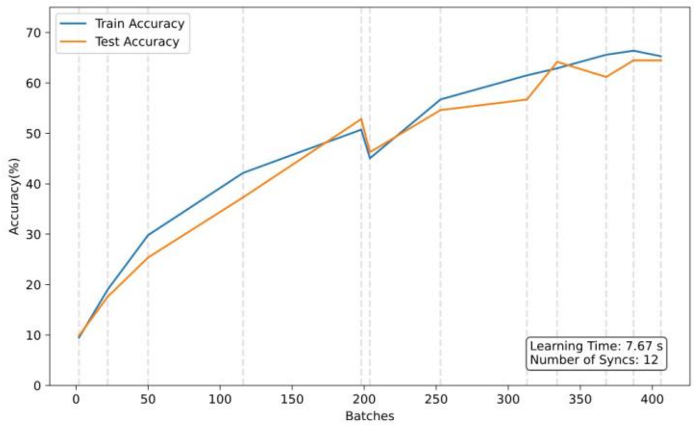


Figure 25: Fast EVENFLOW Protocol using 2 Learners on the UCF Dataset.

The results of using 2 Learners with the Synchronous and the Fast EVENFLOW protocol on the UCF dataset are provided in Figure 24 and Figure 25, respectively. Both approaches reach a Test accuracy score of 60%, while the synchronous approach seems to lack in terms of model generalization given a train accuracy score of nearly 70%. The Fast EVENFLOW occasionally improves Synchronous on model generalization, due to the fact that by synchronizing less frequently, it avoids overfitting that occurs during the streaming training process. In terms of communication cost, Fast EVENFLOW requires only 12 synchronizations compared to 201 of the synchronous approach, reducing synchronization by an order of magnitude, and therefore the communication cost, while achieving the same training performance. This observation is reflected on training time as well, with the Synchronous approach 35s training time, compared with the 8s of Fast EVENFLOW.

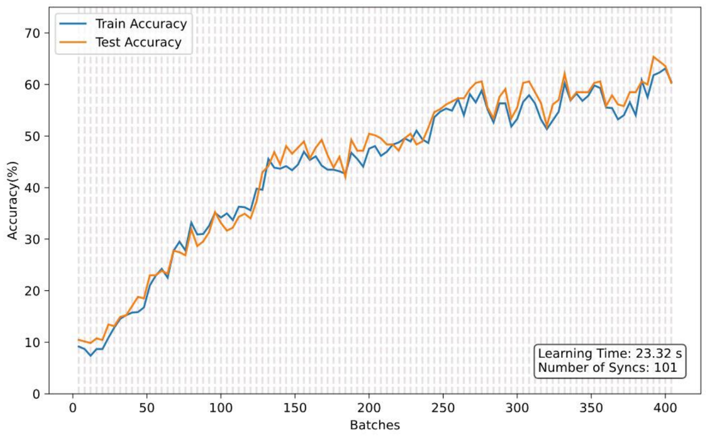


Figure 26: Synchronous Protocol using 4 Learners on the UCF Dataset.

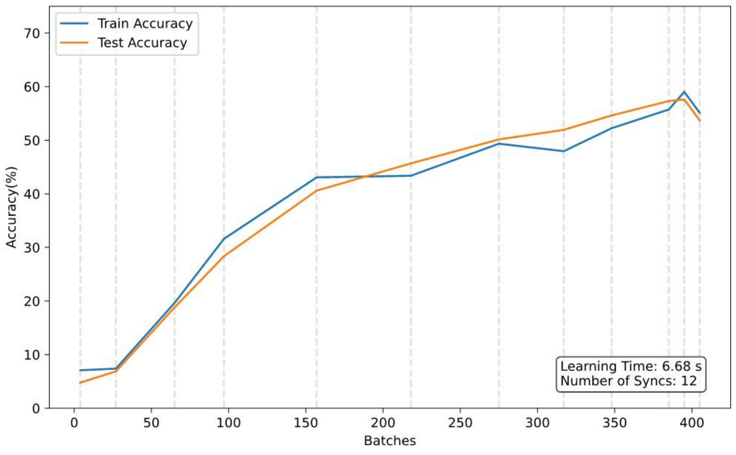


Figure 27: Fast EVENFLOW Protocol using 4 Learners on the UCF Dataset.

We can see the results of on the UCF dataset using 4 Learners in Figure 26 and Figure 27. We observe that the training time is improved in both cases, compared to the 2 Learner approach discussed above. Both protocols achieve concise accuracy metrics with the Fast EVENFLOW approach lacking just by 7%. The use of more Learners, reduced the synchronizations of Synchronous in half. This observation is related to the fact that Fast EVENFLOW already performs a very limited number of synchronizations, with further reduction risking the stability of the training process. Regarding learning time, Fast EVENFLOW is again faster, needing 7s compared to the 24s of Synchronous approach. **Again**, we can see that comparing the use of Fast EVENFLOW with 2 Learners (Figure 25) and the use of Synchronous with 4 Learners (Figure 26), **Fast EVENFLOW achieves the same test accuracy, while reducing training time by a factor of 3 and the number of synchronizations by an order of magnitude, all while using half the number of Learners and therefore half the number of GPUs**. Regarding the communication overhead, Fast EVENFLOW transmits 36 messages during the training process while the synchronous approach transmits 505, reducing drastically the communication cost of the training process, once again.

3.6.1.3 Inference Scalability

Our last set of experiments focus on the Prediction Pipeline and the updated SuBiTO Predictors. Our approach uses parallel Predictors performing inference and communicating their results to a server orchestrator process. This experimental setup focuses only on examining the throughput of the updated Predictor and therefore uses a static pre-trained neural network throughout the experiment. It should be noted that on a real deployment setting the orchestrator process constantly receives the up-to-date model of Training Pipeline and deploys it for inference on the fly. These experiments focus on measuring the increase in throughput that we get when using multiple Predictor instances, compared to the original SuBiTO approach of using a single Predictor. In these experiments we utilise the NSFW dataset and various types of hardware. Each result presented below is the mean value of 10 trials.

We start by examining the limits of a single Predictor running on a standard T4 GPU on Google Colab Pro+. We experiment with various batch sizes and a parallelism of 1 and report the achieved throughput in Figure 28. The results indicate that we achieve higher throughput on batch sizes ranging from 32 to 512, with the maximum being achieved by a batch size of 512. This result indicates that our next experiments should be conducted with a batch size within the aforementioned range.

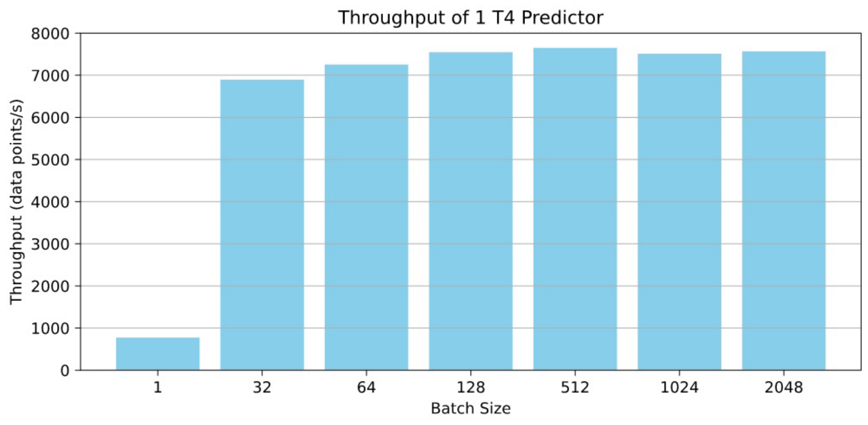


Figure 28: Throughput across batch sizes.

Using the same hardware we utilise a varying number of Predictors, each as a separate Google Colab Pro+ instance, and examine the effect that multiple Predictors can have on throughput. The results regarding T4 Predictors using batch sizes of 32 are presented in Figure 29. We observe that going from 1 to 2 Predictors yields nearly twice the throughput. When scaling to more Predictors we observe that throughput increases almost linearly. The "almost" linear increase is due to the effect of the orchestration process that handles model reception, transmission and fetching of the inference results, therefore introducing latencies.

We continue our experiments by deploying the Prediction Pipeline on state-of-the-art hardware, namely the A100 GPU. Each Predictor is modelled as a separate Google Colab instance, though limited to 3 instances due to Google's policy restrictions. Our experiments examine throughput by varying the number of Predictors within Google's policy limits and use a batch size of 32. The results are presented in Figure 30.

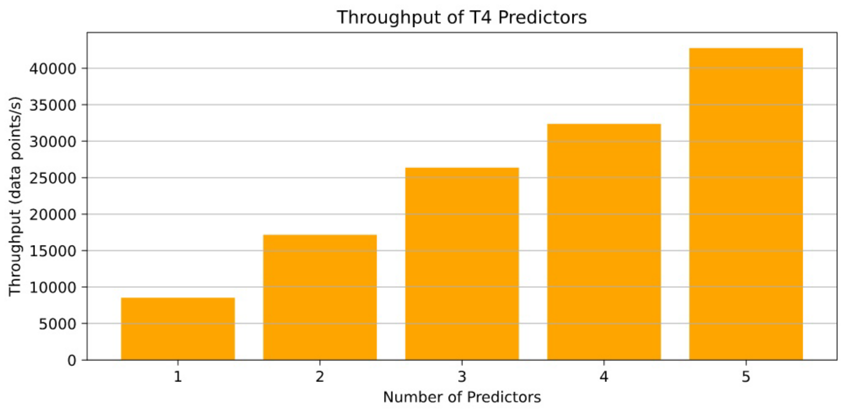


Figure 29: Throughput of T4 Predictors with batch size = 32.

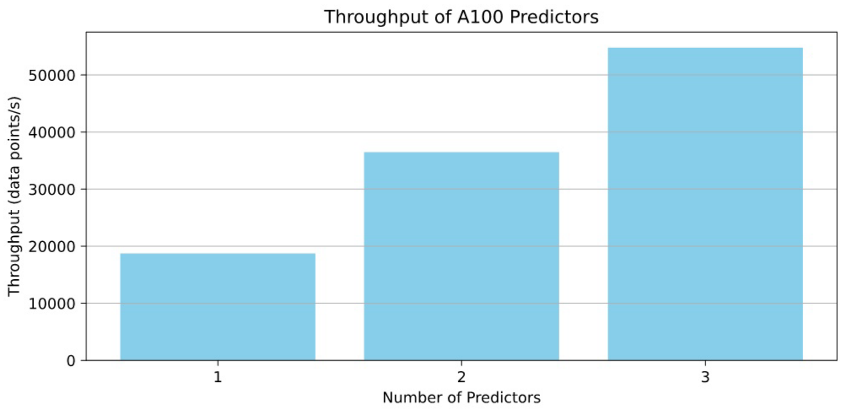


Figure 30: Throughput of A100 Predictors with batch size = 32.

The results indicate twice the throughput, compared to the respective same amount of T4 Predictors, which is a result of using more advanced hardware. Specifically, using 3 A100 Predictors we perform inference on more than 50000 images/s, whereas using 3 T4 Predictors the throughput is decreased by 26000 images/s.

4 Scalable Neurosymbolic Complex Event Recognition

Consider the following pattern, stemming from a smart factory scenario, similar to the Industry 4.0 EVENFLOW use case, where raw robot navigation data are monitored:

R_successful_delivery := (¬StationDetected)* · (¬StationDetected) · (StationDetected ∧ ¬DeliveryManeuver)* · (StationDetected ∧ DeliveryManeuver)

The Simple Event (SE) StationDetected occurs when a robot detects a station in the smart factory. The DeliveryManeuver SE occurs when the robot is moving at a certain speed, changing directions while manoeuvring to approach the detected factory station. The Complex Event (CE) R_successful_delivery is satisfied when initially a robot has not detected a station, then it detects one and, having detected the station, attains the required speed and repeated change of movement direction to approach it. The Complex Event Recognition (CER) system continuously evaluates the rapidly ingested robot streams, converts them to the aforementioned SEs and deduces a successful delivery CE.

In such a scenario, each SE represents the detection of a behaviour that can only be deduced by a machine or neural learning model. The role of the neural model is to receive streams of frames (e.g., from vision, LIDAR, or other contextual cues) and the manoeuvring behaviour for delivery and provide classification outcomes, i.e., class/symbol A = StationDetected and symbol B = DeliveryManeuver, for CER to be possible. Then, a CER engine will ingest the aforementioned SEs (symbols A, B) and evaluate the occurrence of the involved CEs. Evidently, such scenarios call for both neural inference and symbolic CER to operate synergistically.

FlinkCEP is the CER API of a state-of-the-art Big Data platform, namely Apache Flink. FlinkCEP focuses on scaling-out the computation to a number of machines in a computer cluster/cloud, working in parallel on partitions of the streams, to speed up continuous analytic outcomes. FlinkCEP provides a CER language of high expressive power in terms of formulating patterns for CEs. However, there are certain barriers to the adoption of FlinkCEP. First, FlinkCEP requires business analysts, who are not necessarily expert programmers, to write functional programming code. Second, pattern expression and parameterization involve cumbersome notation, making the whole code writing process error prone. Third, with the proliferation of IoT devices as SE producers, the classic paradigm in which we first accumulate raw data at the cloud and then submit a FlinkCEP job is severely suboptimal. For instance, in our running example, sending video frames from robots to the cloud and then performing CER would deplete the available bandwidth, causing network latencies that would prevent the real-time character of the involved applications. What should be done instead, is to ship trained neural models and FlinkCEP jobs to network devices, assign parts of the SE and/or CER process directly on them, and only a subset of SEs and/or CEs should be delivered to the cloud, alerting for the occurred events.

Despite the fact that few previous efforts have integrated neural and symbolic CER, *no existing approach has enabled neither parallel processing of neurosymbolic CER nor optimized, distributed neurosymbolic CER over IoT settings*. The work presented in this section contributes to advancing the state-of-the-art by tackling all the aforementioned challenges.

We introduce *NeuroFlinkCEP* [REF-10]*, the first framework that integrates neural (aka sub-symbolic) and symbolic CER over a state-of-the art Big Data platform for parallel processing that is also optimized to operate distributedly over IoT settings* composed of various devices. To alleviate business analysts from the burden of writing FlinkCEP programs, NeuroFlinkCEP receives expressed patterns in the form of extended regular expressions (RegEx) and transforms them to FlinkCEP jobs. To enable detection of SEs and CEs, NeuroFlinkCEP integrates any chosen, domain-specific neural model inside the FlinkCEP job deployed per device. To optimally decide whether operators of the CER workflow should be executed at the cloud or the device network side, NeuroFlinkCEP enhances a state-of-the-art IoT optimizer with CER-specific optimizations.

4.1.1 NeuroFlinkCEP Architecture

NeuroFlinkCEP's key architectural components are: (i) the RegEx2-NeuroFlinkCEP operator, (ii) the synapSEflow operator and (iii) the DAG*4CER Optimizer. RegEx2NeuroFlinkCEP and synapSEflow operators are nested into a newly introduced NeuroFlinkCEP operator. We have developed a NeuroFlinkCEP GUI for graphical workflow design using NeuroFlinkCEP operators and we have incorporated it as an extension to a commercial platform, Altair AI Studio[6].

The RegEx2NeuroFlinkCEP operator receives as input Extended Regular Expressions describing the pattern based on which a CE would be detected, i.e., this nested operator describes the symbolic part of a NeuroFlinkCEP operator. As shown in Figure 31, each such pattern can be parameterized with time windowing constraints as well as selection strategies and consumption policies supported by FlinkCEP. FlinkCEP supports the following SE selection strategies: (i) Strict Contiguity, (ii) Relaxed Contiguity, and (iii) Non-Deterministic Relaxed Contiguity. For event consumption policies, FlinkCEP provides: NO_SKIP, SKIP_TO_NEXT, SKIP_PAST_LAST_EVENT, SKIP_TO_FIRST[p], and SKIP_TO_LAST[p]. These can be graphically parameterized in a NeuroFlinkCEP operator via the developed extension on Altair AI Studio (Figure 32).

The synapSEflow operator nests the TensorFlow Java API within the NeuroFlinkCEP operator. It receives as input the TensorFlow (.pb) file with a trained neural model. The operator loads the model, composes features from incoming raw streams (e.g., video frames, positional streams), feeds them through the model, and derives labels/symbols. It then (i) directs the simple event outputs to the core of FlinkCEP for pattern matching and (ii) listens to a broadcast stream for model updates via Kafka, installing received models so predictions use the latest version. If no .pb file is specified, the input to a downstream NeuroFlinkCEP operator should be another, upstream NeuroFlinkCEP operator feeding SEs for pattern matching.

The DAG*4CER Optimizer incorporates a state-of-the-art IoT optimization algorithm, DAG* [REF-11], and extends it with FlinkCEP-specific optimizations. DAG* topologically sorts the logical workflow and progressively examines physical instantiations to network devices for the involved operators, computing g(n)=f(n)+h(n) where f(n) is the current partial-plan cost

[6] https://altair.com/altair-ai-studio

and h(n) is an admissible heuristic. Two rules enforce graph (not path) outputs: (i) at each step, only one operator can be examined for instantiation; (ii) no logical operator can be instantiated unless all upstream operators are already in the current partial plan.

DAG*4CER adds CER-oriented rewritings:

- Pattern Decomposition: assigns SE detection across devices instead of placing the entire pattern at one device),
- Early Filtering: prunes irrelevant events with DataStream.filter() before FlinkCEP,
- Reordering: evaluates selective predicates first within FlinkCEP), and
- Pushing Predicates Upstream: moves filtering to source connectors like KafkaSource to save network costs.

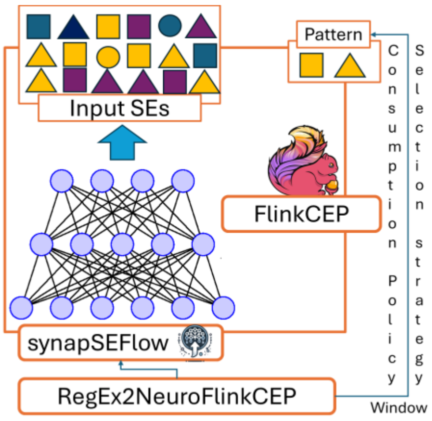
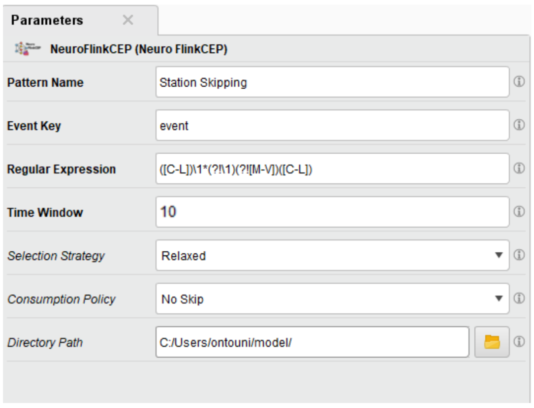


Figure 31: Anatomy of a NeuroFlinkCEP Operator.



Parameters	
NeuroFlinkCEP (Neuro FlinkCEP)	
Pattern Name	Station Skipping
Event Key	event
Regular Expression	([C-L])\1*(?!\1)(?![M-V])([C-L])
Time Window	10
Selection Strategy	Relaxed
Consumption Policy	No Skip
Directory Path	C:/Users/ontouni/model/

Figure 32: NeuroFlinkCEP Operator Parameterization.

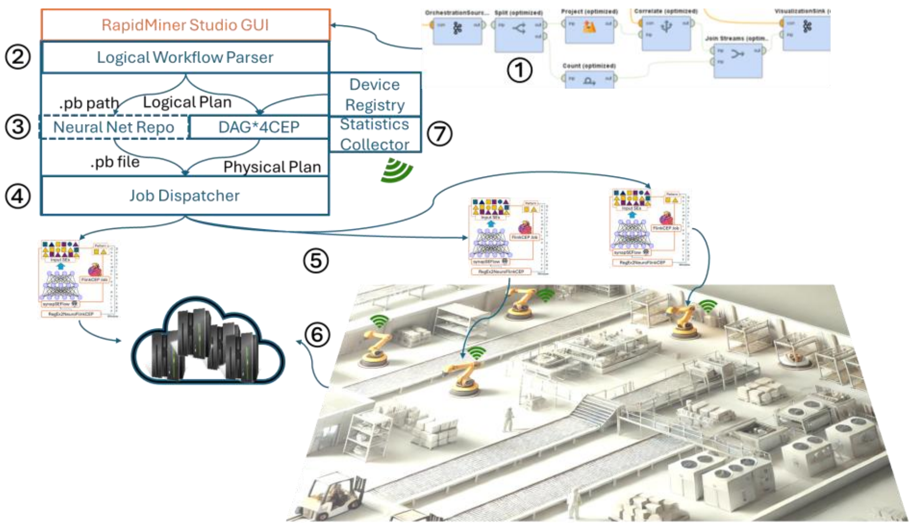


Figure 33: NeuroFlinkCEP Workflow Design, IoT Optimization & Distributed Execution.

4.1.2 From Logical CER workflows to Physical CER over IoT Executions

NeuroFlinkCEP's users and applications are able to perform the numbered tasks in Figure 33. ① The user interacts with the NeuroFlinkCEP GUI in Altair AI Studio to design and parameterize their own logical workflows. The user drags and drops each NeuroFlinkCEP operator on a canvas and connects NeuroFlinkCEP operators and Kafka Source/Sinks to define the data flow as shown on the upper right part of Figure 33. As shown in Figure 32, for each NeuroFLinkCEP operator, the user graphically defines the pattern of interest, selection strategy, consumption policy, time window for the nested RegEx2NeuroFlinkCEP operator. Also, they specify the .pb file path for the nested synapSEflow operator. ② When the user submits the logical workflow, a Logical Workflow Parser checks its validity. It then converts this logical plan to a JSON file that is fed to DAG*4CEP optimizer and to a Neural Net Repo. ③ The DAG*4CER optimizer detects the available network devices via a Device Registry and examines physical assignments to devices for each NeuroFLinkCEP operator, outputting the optimal physical plan. It also projects the optimized physical plan back to the GUI of Altair AI Studio. There are 3 options for the user to interact with the DAG*4CER Optimizer: (a) optimize and deploy: which instructs the Optimizer to directly feed the optimal plan to the Job Dispatcher, (b) only optimize: which instructs the Optimizer to show the suggested physical plan in the GUI for the user to inspect it or change it, before deploying it, (c) only deploy: which will feed the workflow, after (b), to the Dispatcher. ④ In ③(a), ③ (c), the DAG*4CER optimizer feeds the physical plan to the Job Dispatcher, while the Neural Net Repo provides the .pb files for the neural nets engaged in the CER workflow. ⑤The Job Dispatcher submits Flink jobs to the network sites based on the assignment of NeuroFlinkCEP operators by DAG*4CER. ⑥ Detected CEs are continuously visualized in an interactive dashboard (Figure 34). ⑦The deployed plan is monitored and statistics including processing and network latency, throughput and other relevant metrics are collected for future DAG*4CER plan cost estimations.

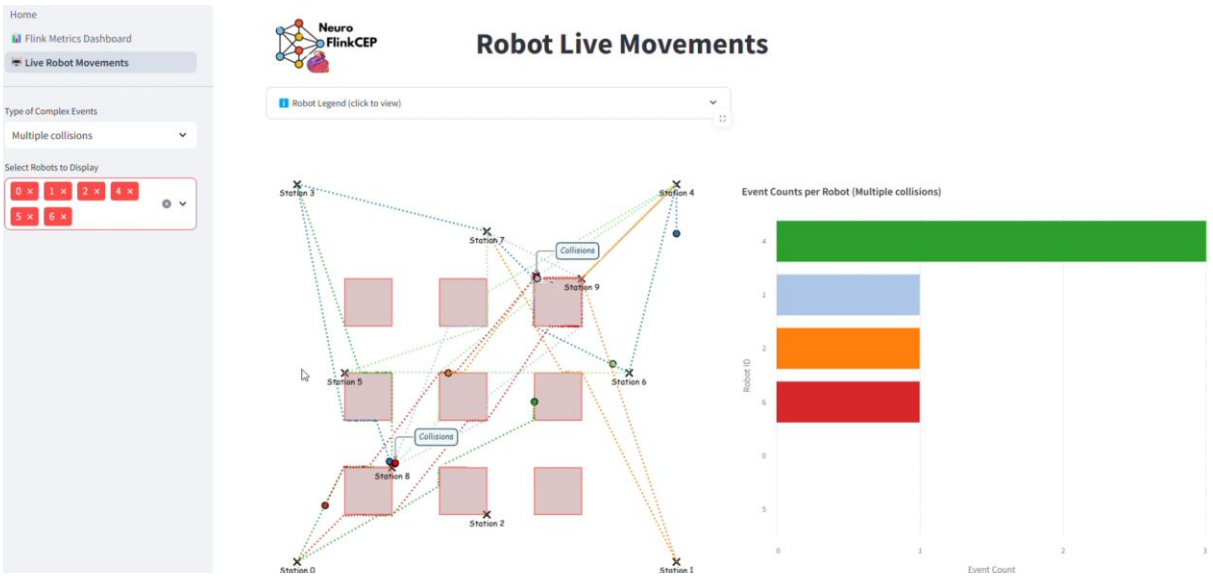
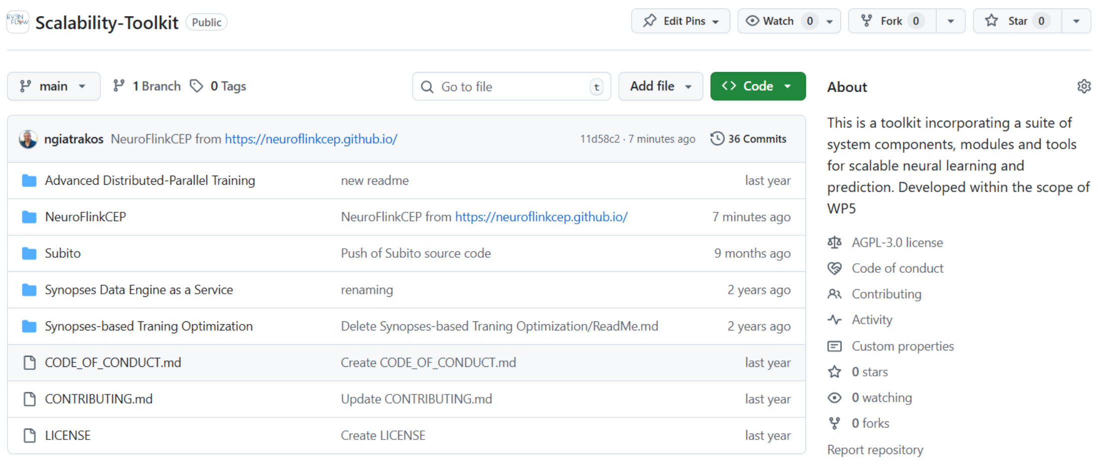


Figure 34: NeuroFlinkCEP dashboard for the Industry 4.0 Use Case.

5 Status of the EVENFLOW Scalability Toolkit

In addition to what was reported in Deliverable D5.1, the EVENFLOW code repository and the Scalability Toolkit in particular, now hosts the 3 main pillars of EVENFLOW scalability. Namely, a) the Synopses-as-a-Service (SDEaaS), (b) SuBiTO and (c) NeuroFlinkCEP. SDE on Dask along with Distribuito SuBiTO on Ray will be provided open source directly after the corresponding systems' papers are published.



ngiatrakos NeuroFlinkCEP from https://neuroflinkcep.github.io/ 11d58c2 · 7 minutes ago 36 Commits

Advanced Distributed-Parallel Training	new readme	last year
NeuroFlinkCEP	NeuroFlinkCEP from https://neuroflinkcep.github.io/	7 minutes ago
Subito	Push of Subito source code	9 months ago
Synopses Data Engine as a Service	renaming	2 years ago
Synopses-based Traning Optimization	Delete Synopses-based Traning Optimization/ReadMe.md	2 years ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	last year
CONTRIBUTING.md	Update CONTRIBUTING.md	last year
LICENSE	Create LICENSE	last year

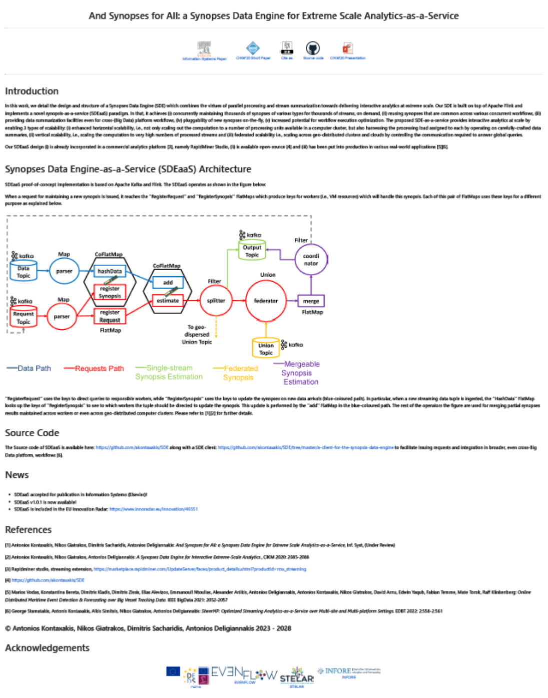
About

This is a toolkit incorporating a suite of system components, modules and tools for scalable neural learning and prediction. Developed within the scope of WP5

AGPL-3.0 license
Code of conduct
Contributing
Activity
Custom properties
0 stars
0 watching
0 forks
Report repository

Figure 35: Status of the Scalability Toolkit at EVENFLOW Repository.

Each of these pillars have also dedicated github.io sites that are continuously in sync with the overall Scalability Toolkit advancements at the EVENFLOW main branch. In particular, SDEaaS has a dedicated site at https://sdeaas.github.io/.



And Synopses for All: a Synopses Data Engine for Extreme Scale Analytics-as-a-Service

Introduction

In this work, we detail the design and structure of a Synopses Data Engine (SDE) which combines the virtues of parallel processing and stream summarization towards delivering interactive analytics at extreme scale. Our SDE is built on top of Apache Flink and implements a novel synopsis-as-a-service (SDEaaS) paradigm. In that, it achieves (i) concurrently maintaining thousands of synopses of various types for thousands of streams, on demand, (ii) reusing synopses that are common across various concurrent workflows, (iii) providing data summarization facilities even for cross-(Big Data) platform workflows, (iv) pluggability of new synopses on-the-fly, (v) increased potential for workflow execution optimization. The proposed SDE-as-a-service provides interactive analytics at scale by enabling 3 types of scalability: (i) enhanced horizontal scalability, i.e., not only scaling out the computation to a number of processing units available in a computer cluster, but also harnessing the processing load assigned to each by operating on carefully-crafted data summaries, (ii) vertical scalability, i.e., scaling the computation to very high numbers of processed streams and (iii) federated scalability i.e., scaling across geo-distributed clusters and clouds by controlling the communication required to answer global queries.

Our SDEaaS design (i) is already incorporated in a commercial analytics platform [3], namely RapidMiner Studio, (ii) is available open-source [4] and (iii) has been put into production in various real-world applications [5][6].

Synopses Data Engine-as-a-Service (SDEaaS) Architecture

SDEaaS proof-of-concept implementation is based on Apache Kafka and Flink. The SDEaaS operates as shown in the figure below:

When a request for maintaining a new synopsis is issued, it reaches the "RegisterRequest" and "RegisterSynopsis" FlatMaps which produce keys for workers (i.e., VM resources) which will handle this synopsis. Each of this pair of FlatMaps uses these keys for a different purpose as explained below.

"RegisterRequest" uses the keys to direct queries to responsible workers, while "RegisterSynopsis" uses the keys to update the synopses on new data arrivals (blue-coloured path). In particular, when a new streaming data tuple is ingested, the "HashData" FlatMap looks up the keys of "RegisterSynopsis" to see to which workers the tuple should be directed to update the synopsis. This update is performed by the "add" FlatMap in the blue-coloured path. The rest of the operators the figure are used for merging partial synopses results maintained across workers or even across geo-distributed computer clusters. Please refer to [1][2] for further details.

Source Code

The Source code of SDEaaS is available here: https://github.com/akontaxakis/SDE along with a SDE client: https://github.com/akontaxakis/SDE/tree/master/a-client-for-the-synopsis-data-engine to facilitate issuing requests and integration in broader, even cross-Big Data platform, workflows [6].

News

- SDEaaS accepted for publication in Information Systems (Elsevier)!
- SDEaaS v1.0.1 is now available!
- SDEaaS is included in the EU Innovation Radar: https://www.innoradar.eu/innovation/46551

References

[1] Antonios Kontaxakis, Nikos Giatrakos, Dimitris Sacharidis, Antonios Deligiannakis: *And Synopses for All: a Synopses Data Engine for Extreme Scale Analytics-as-a-Service*, Inf. Syst. (Under Review)

[2] Antonios Kontaxakis, Nikos Giatrakos, Antonios Deligiannakis: *A Synopses Data Engine for Interactive Extreme-Scale Analytics*, CIKM 2020: 2085-2088

[3] Rapidminer studio, streaming extension, https://marketplace.rapidminer.com/UpdateServer/faces/product_details.xhtml?productId=rmx_streaming

[4] https://github.com/akontaxakis/SDE

[5] Marios Vodas, Konstantina Bereta, Dimitris Kladis, Dimitris Zissis, Elias Alevizos, Emmanouil Ntoulias, Alexander Artikis, Antonios Deligiannakis, Antonios Kontaxakis, Nikos Giatrakos, David Arnu, Edwin Yaqub, Fabian Temme, Mate Torok, Ralf Klinkenberg: *Online Distributed Maritime Event Detection & Forecasting over Big Vessel Tracking Data.* IEEE BigData 2021: 2052-2057

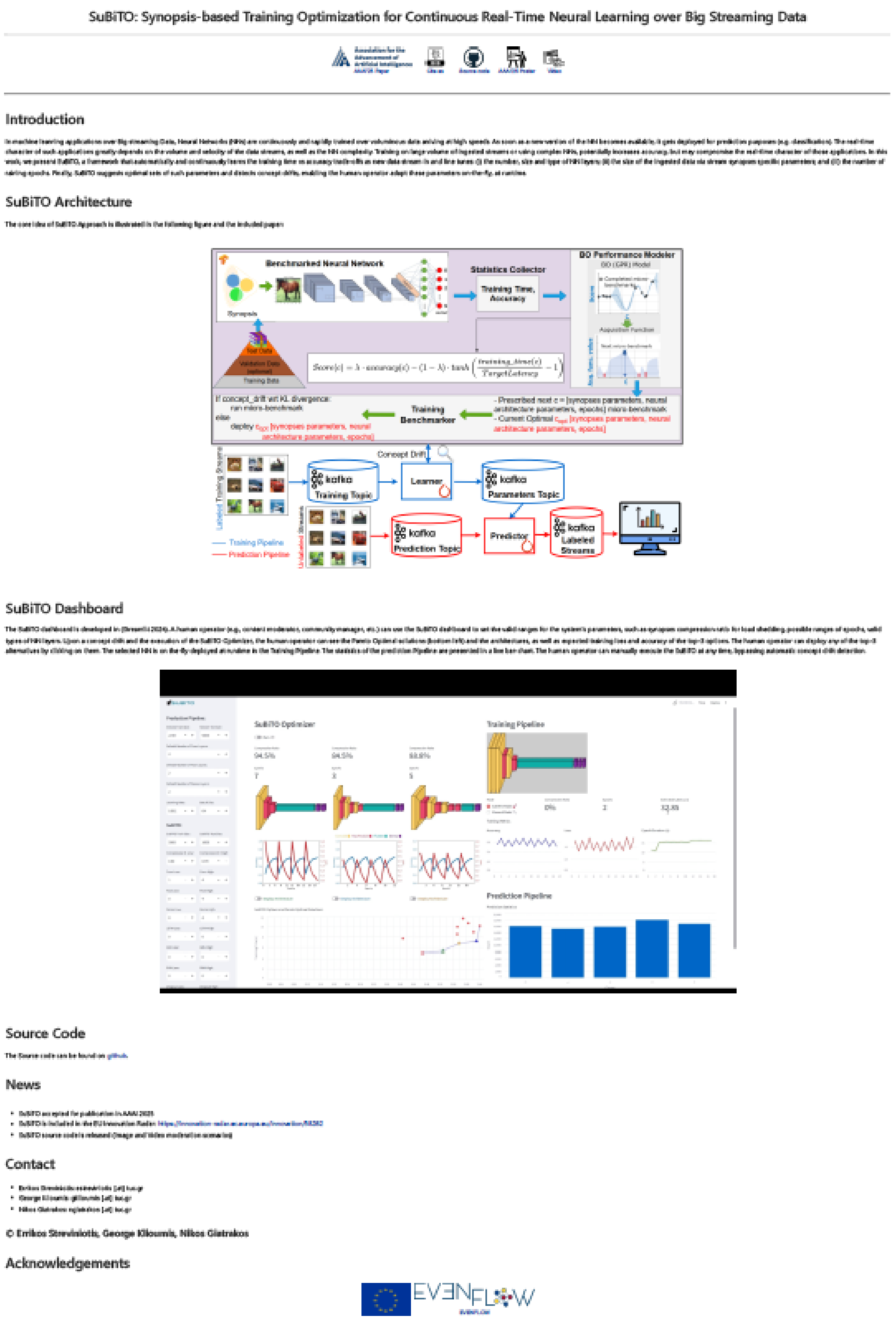
[6] George Stamatakis, Antonis Kontaxakis, Alkis Simitsis, Nikos Giatrakos, Antonios Deligiannakis: *SheerMP: Optimized Streaming Analytics-as-a-Service over Multi-site and Multi-platform Settings.* EDBT 2022: 2:558-2:561

© Antonios Kontaxakis, Nikos Giatrakos, Dimitris Sacharidis, Antonios Deligiannakis 2023 - 2028

Acknowledgements

Figure 36: SDEaaS at https://sdeaas.github.io/.

SuBiTO has a dedicated site available at: https://subito-ai-for-bigdata.github.io/.


SuBiTO: Synopsis-based Training Optimization for Continuous Real-Time Neural Learning over Big Streaming Data

Introduction

In machine learning applications over Big streaming Data, Neural Networks (NNs) are continuously and rapidly trained over voluminous data arriving at high speeds. As soon as a new version of the NN becomes available, it gets deployed for prediction purposes (e.g. classification). The real-time character of such applications greatly depends on the volume and velocity of the data streams, as well as the NN complexity. Training on large volume of ingested streams or using complex NNs, potentially increases accuracy, but may compromise the real-time character of these applications. In this work, we present SuBiTO, a framework that automatically and continuously learns the training time vs accuracy trade-offs as new data stream in and fine tunes: (i) the number, size and type of NN layers; (ii) the size of the ingested data via stream synopses specific parameters; and (iii) the number of training epochs. Finally, SuBiTO suggests optimal sets of such parameters and detects concept drifts, enabling the human operator adapt these parameters on-the-fly, at runtime.

SuBiTO Architecture

The core idea of SuBiTO Approach is illustrated in the following figure and the included paper:

Benchmarked Neural Network
Synopsis
Test Data
Validation Data (optional)
Training Data
Statistics Collector
Training Time, Accuracy
BO Performance Modeler
BO (GPR) Model
Acquisition Function
$Score(c) = \lambda \cdot accuracy(c) - (1-\lambda) \cdot \tanh\left(\frac{training\_time(c)}{TargetLatency} - 1\right)$
If concept_drift wrt KL divergence: run micro-benchmark
else: deploy c_{opt} [synopses parameters, neural architecture parameters, epochs]
Training Benchmarker
- Prescribed next c = [synopses parameters, neural architecture parameters, epochs] micro-benchmark
- Current Optimal c_{opt} [synopses parameters, neural architecture parameters, epochs]
Concept Drift
Labeled Training Streams
kafka Training Topic
Learner
kafka Parameters Topic
Unlabeled Streams
kafka Prediction Topic
Predictor
kafka Labeled Streams
Training Pipeline
Prediction Pipeline

SuBiTO Dashboard

The SuBiTO dashboard is developed in [Streamlit 2024]. A human operator (e.g., content moderator, community manager, etc.) can use the SuBiTO dashboard to set the valid ranges for the system's parameters, such as synopses compression ratio for load shedding, possible ranges of epochs, valid types of NN layers. Upon a concept drift and the execution of the SuBiTO-Optimizer, the human operator can see the Pareto Optimal solutions (bottom left) and the architectures, as well as expected training time and accuracy of the top-3 options. The human operator can deploy any of the top-3 alternatives by clicking on them. The selected NN is on-the-fly deployed at runtime in the Training Pipeline. The statistics of the prediction Pipeline are presented in a live bar chart. The human operator can manually execute the SuBiTO at any time, bypassing automatic concept drift detection.

Source Code

The Source code can be found on github.

News

- SuBiTO accepted for publication in AAAI 2025
- SuBiTO is included in the EU Innovation Radar: https://innovation-radar.ec.europa.eu/innovation/[illegible]
- SuBiTO source code is released (Image and Video moderation scenarios)

Contact

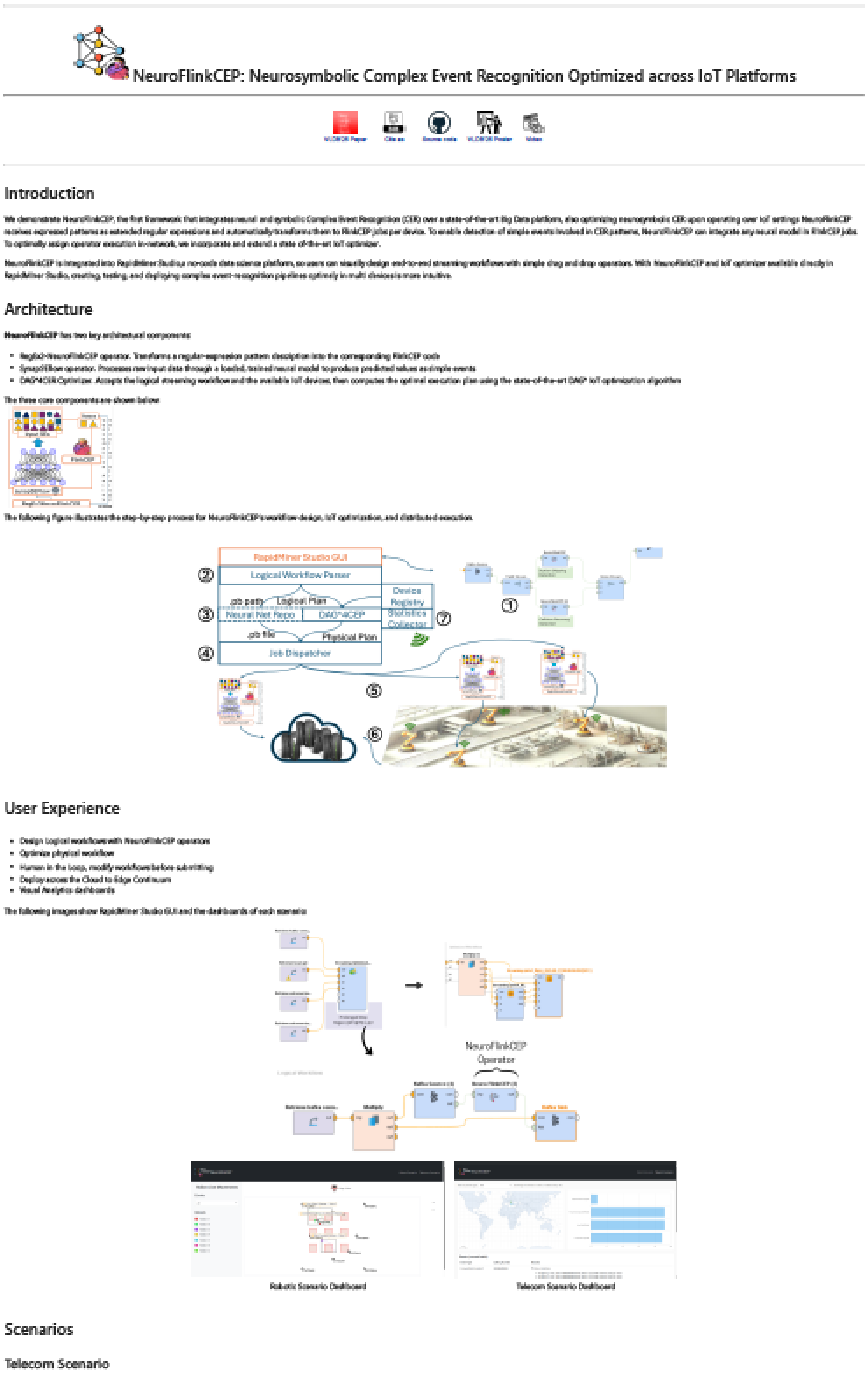
- Errikos Streviniotis: estreviniotis [at] tuc.gr
- George Klioumis: gklioumis [at] tuc.gr
- Nikos Giatrakos: ngiatrakos [at] tuc.gr

© Errikos Streviniotis, George Klioumis, Nikos Giatrakos

Acknowledgements

Figure 37: SuBiTO at https://subito-ai-for-bigdata.github.io/.

NeuroFlinkCEP has a dedicated site available at https://neuroflinkcep.github.io/.



NeuroFlinkCEP: Neurosymbolic Complex Event Recognition Optimized across IoT Platforms

Introduction

We demonstrate NeuroFlinkCEP, the first framework that integrates neural and symbolic Complex Event Recognition (CER) over a state-of-the-art Big Data platform, also optimizing neurosymbolic CER upon operating over IoT settings. NeuroFlinkCEP receives expressed patterns as extended regular expressions and automatically transforms them to FlinkCEP jobs per device. To enable detection of simple events involved in CER patterns, NeuroFlinkCEP can integrate any neural model in FlinkCEP jobs. To optimally assign operator execution in-network, we incorporate and extend a state-of-the-art IoT optimizer.

NeuroFlinkCEP is integrated into RapidMiner Studio, a no-code data science platform, so users can visually design end-to-end streaming workflows with simple drag and drop operators. With NeuroFlinkCEP and IoT optimizer available directly in RapidMiner Studio, creating, testing, and deploying complex event-recognition pipelines optimally in multi devices is more intuitive.

Architecture

NeuroFlinkCEP has two key architectural components:

- Regex2-NeuroFlinkCEP operator. Transforms a regular-expression pattern description into the corresponding FlinkCEP code
- SynapSEflow operator. Processes raw input data through a loaded, trained neural model to produce predicted values as simple events
- DAG*4CER Optimizer. Accepts the logical streaming workflow and the available IoT devices, then computes the optimal execution plan using the state-of-the-art DAG* IoT optimization algorithm

The three core components are shown below:

The following figure illustrates the step-by-step process for NeuroFlinkCEP's workflow design, IoT optimization, and distributed execution.

User Experience

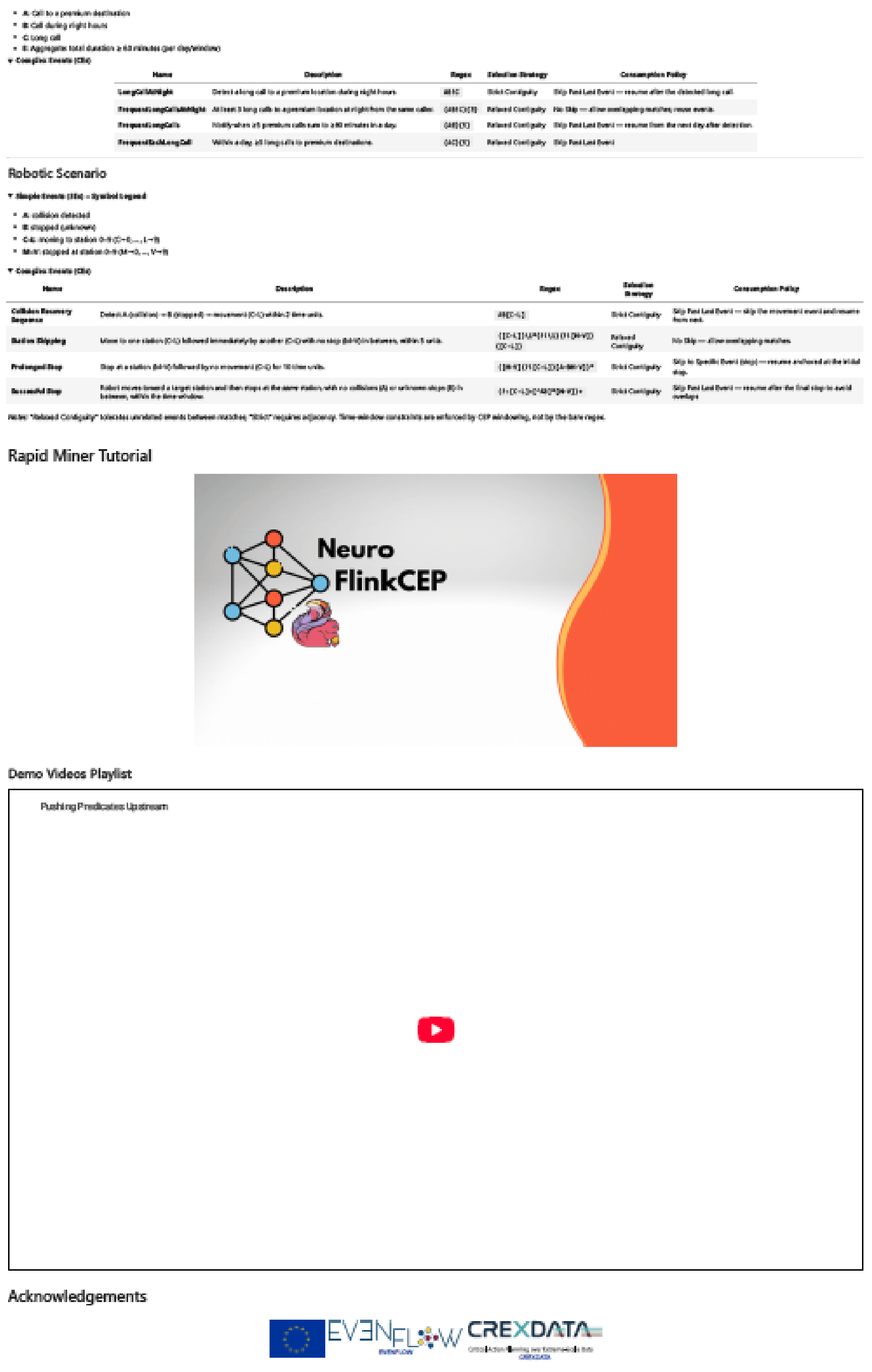
- Design Logical workflows with NeuroFlinkCEP operators
- Optimize physical workflow
- Human in the Loop, modify workflows before submitting
- Deploy across the Cloud to Edge Continuum
- Visual Analytics dashboards

The following images show RapidMiner Studio GUI and the dashboards of each scenario:

Scenarios

Telecom Scenario

Figure 38: NeuroFlinkCEP at https://neuroflinkcep.github.io/ (part 1).



- A: Call to a premium destination
- B: Call during night hours
- C: Long call
- E: Aggregate total duration ≥ 60 minutes (per day/window)

▾ Complex Events (CEs)

Name	Description	Regex	Selection Strategy	Consumption Policy
LongCallAtNight	Detect a long call to a premium location during night hours.	ABC	Strict Contiguity	Skip Past Last Event — resume after the detected long call.
FrequentLongCallsAtNight	At least 3 long calls to a premium location at night from the same caller.	(ABC){3}	Relaxed Contiguity	No Skip — allow overlapping matches; reuse events.
FrequentLongCalls	Notify when ≥5 premium calls sum to ≥60 minutes in a day.	(AE){5}	Relaxed Contiguity	Skip Past Last Event — resume from the next day after detection.
FrequentEachLongCall	Within a day, all long calls to premium destinations.	(AC){3}	Relaxed Contiguity	Skip Past Last Event

Robotic Scenario

▾ Simple Events (SEs) – Symbol Legend

- A: collision detected
- B: stopped (unknown)
- C–L: moving to station 0–9 (C→0, …, L→9)
- M–V: stopped at station 0–9 (M→0, …, V→9)

▾ Complex Events (CEs)

Name	Description	Regex	Selection Strategy	Consumption Policy
Collision Recovery Sequence	Detect A (collision) → B (stopped) → movement (C–L) within 2 time units.	AB[C-L]	Strict Contiguity	Skip Past Last Event — skip the movement event and resume from next.
Station Skipping	Move to one station (C–L) followed immediately by another (C–L) with no stop (M–V) in between, within 5 units.	[illegible]	Relaxed Contiguity	No Skip — allow overlapping matches.
Prolonged Stop	Stop at a station (M–V) followed by no movement (C–L) for 10 time units.	[illegible]	Strict Contiguity	Skip to Specific Event (stop) — resume anchored at the initial stop.
Successful Stop	Robot moves toward a target station and then stops at the same station, with no collisions (A) or unknown stops (B) in between, within the time window.	[illegible]	Strict Contiguity	Skip Past Last Event — resume after the final stop to avoid overlaps.

Notes: "Relaxed Contiguity" tolerates unrelated events between matches; "Strict" requires adjacency. Time-window constraints are enforced by CEP windowing, not by the bare regex.

Rapid Miner Tutorial

Demo Videos Playlist

Acknowledgements

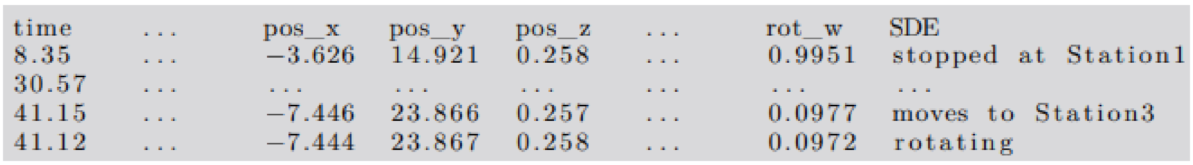
Figure 39: NeuroFlinkCEP at https://neuroflinkcep.github.io/ (Part 2).

Each dedicated site provides additional material including the published papers, videos and images as well as posters and presentation slides of the involved scalability pillars.

6 Scaling EVENFLOW Use Cases

6.1 The SSTRESSED Framework for the Industry 4.0 Use Case

Detecting Simple, Derived Events (SDEs) is the first step towards Complex Event Recognition. In time critical applications, such as safe robot navigation in dynamic smart factory environments of Use Case II, SDE detection should be performed continuously over voluminous streams of movement data arriving at high speeds. In such scenarios, extracting SDEs out of raw streams is a challenging task engaging (a) online neural network training for continuously maintaining an up-to-date model for SDE labelling purposes and (b) semantic-aware trajectory processing for identifying homogeneous movement portions, defining the SDE duration, before using the neural model for labelling it. By definition, output SDEs are simple pieces of information, but the volume and velocity of the original raw streams (Figure 40) in large scale smart factory applications call for scaling out (parallelizing) the computation to a number of machines to ensure real-time processing. Therefore, both (a) and (b) should be set up in state-of-the-art, relevant platforms. To tackle these challenges, we developed SSTRESED, a prototype for scalable SDE detection over streaming movement data. For the first time, SSTRESED establishes a direct connection between semantic trajectory computation and SDE detection in the streaming context. This is in contrast to prior art which uses predetermined, application-defined time windows to a priori restrict eligible SDE durations.



time	...	pos_x	pos_y	pos_z	...	rot_w	SDE
8.35	...	−3.626	14.921	0.258	...	0.9951	stopped at Station1
30.57	...	...	...	...	...	...	...
41.15	...	−7.446	23.866	0.257	...	0.0977	moves to Station3
41.12	...	−7.444	23.867	0.258	...	0.0972	rotating

Figure 40: Example training stream for a single simulated robot. Unlabelled movement streams.

The SSTRESSED Framework [REF-12], illustrated in Figure 41 and described in Deliverable D5.1, composes two connected pipelines distributed across worker machines. In the Industry 4.0 use case of EVENFLOW, truthful, timestamped and labelled movement streams are continuously produced by robotic simulators, as SDEs and their raw features, per robot.

The training pipeline (blue-coloured path) in Figure 41 abides by the one reported in Figure 2. It continuously receives these robot movement time series ingested in Apache Kafka partitions of the Training Topic. The Training Topic is read by parallel PyTorch Learners. Each such learner utilizes an identical neural model (specified by the application) but performs the training process on a separate set of robots. The local models learned at each Learner i (top of Figure 41) are synchronized into a global neural model maintained by a Parameter Server. At a global model update, new weights of the neural network are written to a Weights Topic of Kafka.

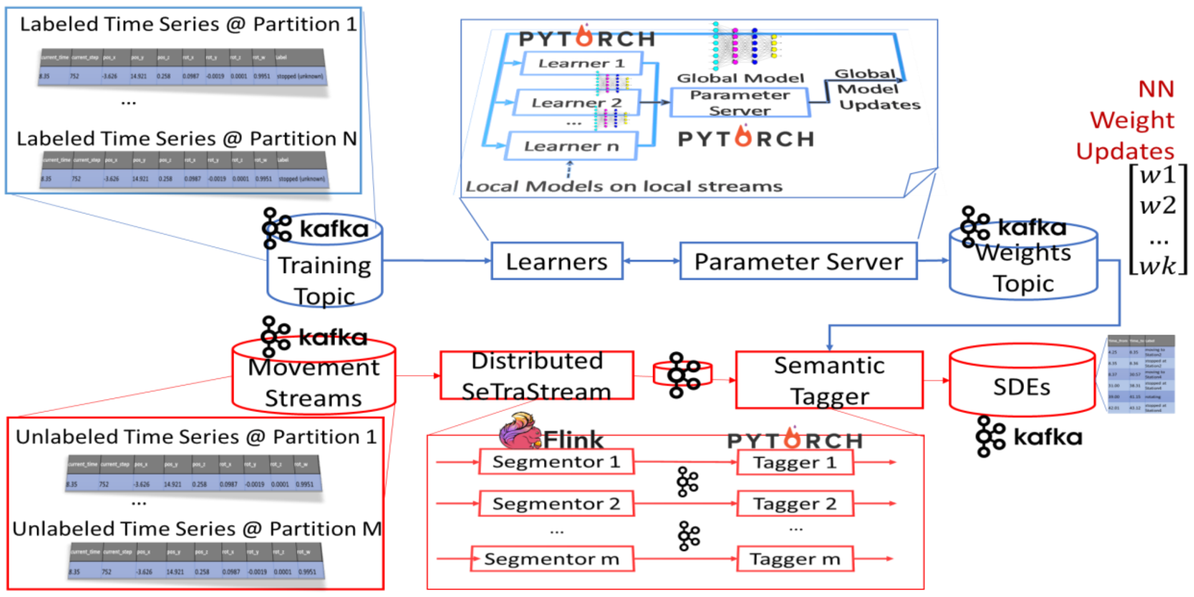

Labeled Time Series @ Partition 1
Labeled Time Series @ Partition N
kafka
Training Topic
Learners
Parameter Server
Weights Topic
NN Weight Updates
$\begin{bmatrix} w1 \\ w2 \\ \dots \\ wk \end{bmatrix}$
PYTORCH
Learner 1
Learner 2
Learner n
Global Model Parameter Server
Global Model Updates
Local Models on local streams
Movement Streams
Distributed SeTraStream
Semantic Tagger
SDEs
Unlabeled Time Series @ Partition 1
Unlabeled Time Series @ Partition M
Flink
Segmentor 1
Segmentor 2
Segmentor m
Tagger 1
Tagger 2
Tagger m

Figure 41: SSTRESED Architecture. Training (blue) and SDE Detection (red) Pipelines [REF-12].

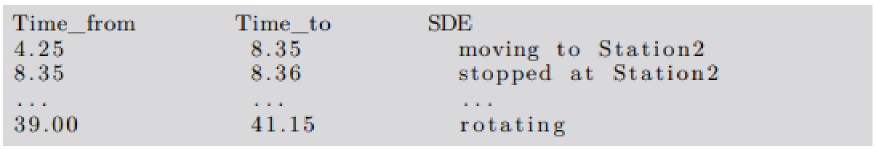
The SDE detection pipeline (red-coloured path in Figure 41) also abides by Figure 2 and extends it with semantic trajectory episode determination via SeTraStream [REF-13]. It receives raw, unlabelled streaming movement data, partitioned in the Movement Streams Kafka Topic. These incoming tuples, ingested directly from the application field, have the same schema as those of the Training Topic, but lack a label/SDE field. Ingested Movement Streams of robots (or, optionally, samples of them) are processed by a distributed version of SeTraStream [REF-13] developed in Apache Flink. Distributed SeTraStream uses each parallel Segmentor i to continuously identify homogeneous movement portions based on the ingested features per robot, thus semantically and temporally segmenting each trajectory. In that, the duration of a SDE is determined, which also bounds the feature tensors that should then be used for labelling the SDE.

Let us assume that there is a buffer containing batches of records of a homogenous movement. The segmentation algorithm of SeTraStream starts by extracting a batch of m records from the streaming data. Once the batch is extracted, its similarity is evaluated against the last batch that was found to be consistent with the previous batches within the current buffer of a specific movement. If similarity is satisfied, the algorithm continues by testing the new batch against the last 2n batches in the buffer, where n starts from 0 and increases until 2n reaches the buffer length. This process continues until the batch is found to be similar to the entire buffer, in which case it is appended to it. When no similarity is detected, the current buffer is assumed to be a separate movement, and the new batch initializes a new buffer corresponding to the next movement. The algorithm then repeats. Similarity is quantified using a metric known as the RV coefficient. The RV-coefficient constitutes a generalization of the correlation coefficient for matrix data. We organize W_l into a d × m matrix, where d is the number of movement features and m represents a number of vectors (at different timestamps) that are the columns of the matrix. Similarly, W_r is organized in a d × m matrix i.e. n columns exist. The RV coefficient is mathematically defined as follows:

$$RV(W_l, W_r) = \frac{Tr(W_l W_l^T W_r W_r^T)}{\sqrt{Tr((W_l W_l^T)^2)\, Tr((W_r W_r^T)^2)}}$$

Where W_l and W_r are the matrices to be tested if similar, W^T_l , W^T_r refer to the transpose matrices, Tr() denotes the trace of a matrix and $0 \leq RV \leq 1$. RV values closer to zero are indicative of uncorrelated movement patterns. Based on a division point threshold σ, matrices Wl, Wr can be either assigned to a pair of different episodes or to a single episode.

Each parallel Segmentor i writes the result of its processing to an intermediate Kafka topic connecting Distributed SeTraStream with a PyTorch Semantic Tagger in the red-coloured path. Each parallel Tagger i (bottom of Figure 41) of the Semantic Tagger, at any given time instance, reads the up-to-date weights from the Weights Topic and uses the updated neural model to label SDEs. The final SSTRESED output goes to the SDEs Kafka topic in the form of tuples as illustrated in Figure 42 (per robot).



```
Time_from          Time_to        SDE
4.25                8.35            moving to Station2
8.35                8.36            stopped at Station2
...                 ...             ...
39.00               41.15           rotating
```

Figure 42: SSTRESED output SDE Stream for the movement of a single robot.

6.1.1 SSTRESED Experimental Evaluation

In Deliverable D5.1 of EVENFLOW we introduced the SSTRESSED architecture [REF-12] and reasoned about its utility in the Industry 4.0 use case. In this section, we experimentally validate the feasibility and scalability of SSTRESSED implementation over robot movement data.

To evaluate the performance of the SeTraStream in accurately classifying robot motion data of homogeneous movement segments, three widely used classification metrics are employed: Precision, Recall, and F1-Score. These measures provide complementary insights into the model's predictive behaviour and are suitable for problems where class imbalance or partial misclassifications may occur.

The training pipeline, in the scheme of streaming data input, employs a distributed learning method so that the system can benefit significantly in terms of time through data allocation. In applications such as robot navigation within a smart factory environment, where robots are prone to intermediate collisions, the distributed approach helps prevent system latencies and ensures a more stable processing flow. The training pipeline was executed for 1, 2, and 4 workers, and the results of the training time per number of workers are presented in the diagram below. As Figure 43 illustrates the training time upon using 2 learners reduces by 1.7x, while it further decreases to 2.7x times in the case of 4 learners, compared to the single learner alternative. The difference between the linear decrease in training time i.e., 2 times for 2 learners and 4 times for 4 learners, is due to the fact that in this experiment we utilized the synchronous, instead of the EVENFLOW, protocol of the parameter server paradigm for parallel training. We did so, in order to stress-test the accuracy of the Prediction Pipeline of

SSTRESED even in the presence of laggy transfer of the most up-to-date model, from the Training to the Prediction Pipeline.

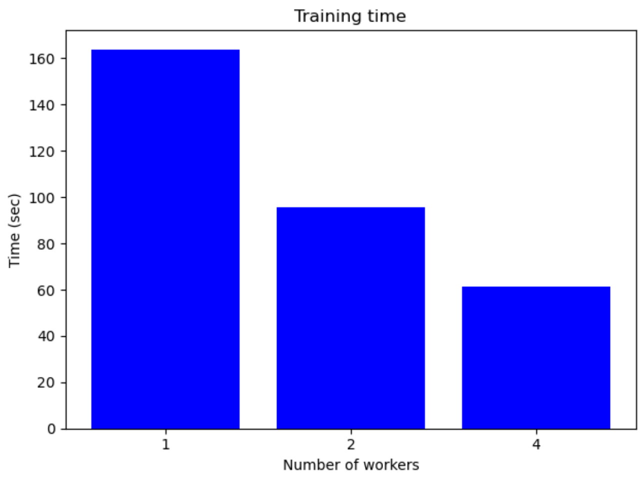


Figure 43: SSTRESED Training Pipeline performance.

Figure 44 presents the SSTRESED throughput, illustrating the number of predictions per second with respect to the number of taggers (predictors) operating in parallel. As shown in the following diagram, the system throughput increases with the number of taggers. In particular, making the transition from 1 to 2 predictors/taggers increases the throughput by 4x, while from 4à8 taggers throughput increases by 3x. This super linear increase in throughput comes from the fact that predictors, contrary to learners, do not need to synchronize and can operate independently provided they have the most up-to-date neural model at any given time.

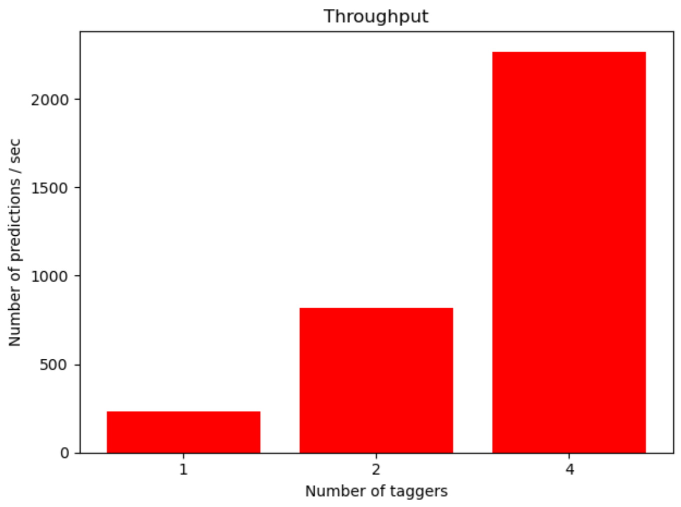


Figure 44: Performance of SSTRESED Prediction Pipeline.

An important question that needs to be answered involves the accuracy of SSTRESED. Instead of evaluating the accuracy of the Training Pipeline and the Prediction Pipeline independently, we choose to directly show the overall accuracy of SSTRESED in the detected SDEs. This is because, by reporting accuracy of the detected SDEs we provide the cumulative accuracy of the framework across (i) the Training Pipeline: not only how accurate is the neural network but also how the small time lap between producing the most up-to-date neural model at each epoch and deploying it across predictors affects the overall accuracy, (ii) the Segmentation part of the Prediction Pipeline: how well robot trajectories are segmented to homogeneous portions of movements (episodes) and (iii) how well the neural model deployed at the Prediction Pipeline, generalizes to unseen data.

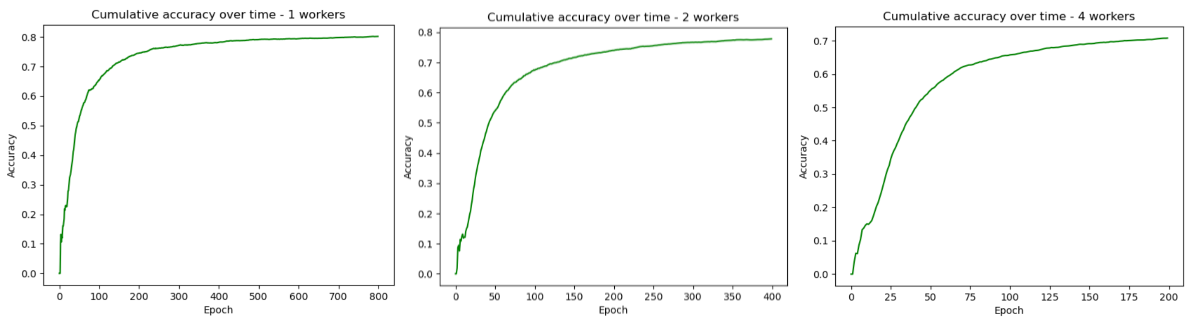


Figure 45: Cumulative SSTRESED accuracy vs Number of Epochs.

Figure 45 shows SSTRESED accuracy across the number of epochs. The number of epochs here denotes the time frame of SSTRESED operation since each new epoch yields a new neural model that is conveyed from the Training to the Prediction Pipeline. This is the reason the number of epochs in the horizontal axis of the figure is halved across the plots, every time we double the number of workers, i.e., the entire dataset is partitioned to more workers. As Figure 45 shows, SSTRESED achieves a cumulative accuracy of 80% with the exception of 4 workers where accuracy reaches 70%, but for early stopping at epoch 200 due to the reason mentioned above. This 70% accuracy at epoch 200 is consistent with the rest of the cases (1 worker, 2 workers) proving that the only reason for the 10% lack is the boundedness of the robot dataset rather than convergence issues of the framework.

Finally, it is important to note that the lower absolute throughput numbers in Figure 44 compared to Figure 29 and Figure 30 comes from the fact that the current Prediction Pipeline includes SeTraStream for segmentation purposes. SeTraStream must not be considered as a performance bottleneck though, since the absence of SeTraStream significantly compromises the accuracy of the framework. This is due to omitting the segmentation step in a setup where SDEs are durative. Therefore, even if a version of SSTRESED without SeTraStream provides much higher throughput, the accuracy of the framework remains significantly low. This is due to the fact that the predictors retain correct tagging of individual motion tuples due to the neural model, but even a single incorrectly tagged tuple splits durative SDEs into non-consecutive time periods and introduced erroneous interleaved SDEs, as well. In other works, if SSTRESED makes predictions on individual tuples, instead of episodes, most of the labels remain correct, but even few wrongly labelled tuples lead to SDEs of invalid duration. Therefore, the overall prediction accuracy is diminished.

To validate our claim, Figure 46 shows the accuracy and throughput of SSTRESED without SeTraStream. As shown in Figure 46, throughput dramatically increases up to 15x for 1 predictor upon omitting SeTraStream from the prediction pipeline, but accuracy receives a maximum value below 0.25

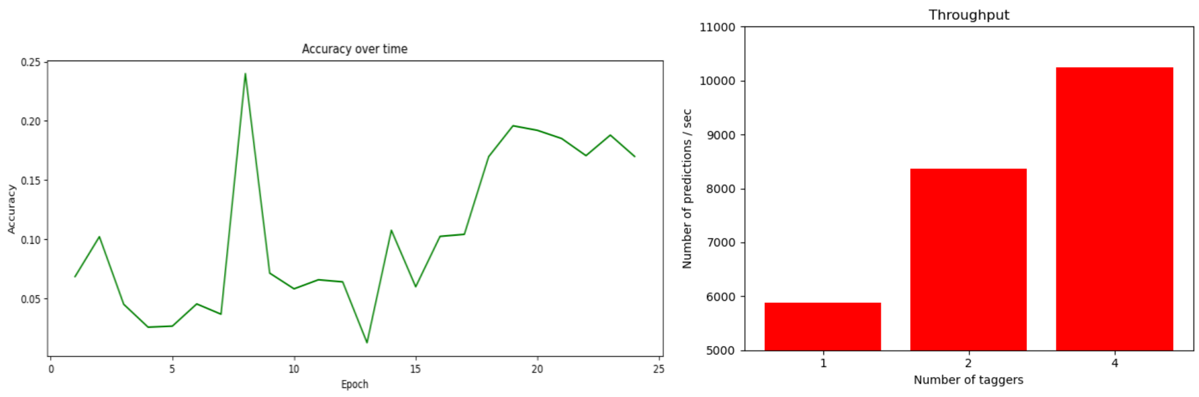


Figure 46: SSTRESED performance without SeTraStream.

6.2 The RATS+ Framework for the Personalized Medicine Use Case

6.2.1 Overview on the RATS Framework

Tumour simulations such as those run on PhysiBoSS 2.0 are expensive, highly parallel, and parameter-rich. Each simulation corresponds to a tumour treatment methodology under examination, described by three TNF parameters—drug administration frequency, duration, and concentration. To cut down time to market for new therapies, each set of simulations needs to be properly scheduled. That is, there is a need to aid life scientists reserve a sufficient number of core hours for their medical study each time, devote the proper number of cores to each simulation so that the study finishes as early as possible and prioritize higher the most promising simulations so that early, useful results can be extracted and prematurely end the rest of the unpromising simulations if needed. In realistic studies, the number of possible TNF combinations is large: in the initial RATS framework [REF-14], described in Deliverable D5.1, our experiments consider 512 TNF triplets and core configurations up to 32 cores per simulation, resulting in 2,560 different {TNF, cores} configurations that could potentially be executed on the MareNostrum 4 supercomputer. Running all these configurations just to learn how performance and treatments behave is clearly infeasible. At the same time, life scientists need answers to three tightly coupled questions: (i) how many cores should be assigned to each simulation so that cores are not under- or over-utilized, (ii) under a global core-capacity cap, which simulations should run at each time step, and with how many cores, to minimize the overall time of the study, and (iii) among all candidate treatments, which ones look most promising and should therefore be prioritized. RATS addresses this full chain of decisions: it learns performance models from a small number of micro-benchmarks and then solves a series of knapsack-style optimization problems to prescribe core allocations and derive a schedule.

The RATS framework [REF-14] starts from a discretized treatment space. TNF Frequency, Duration, and Concentration are discretized into fine-grained ranges via [low, high, step] triplets. All combinations form the set SS of TNF triplets under study. For each simulation S_i in SS, the user may choose a core count k from a set of valid core configurations (e.g., 2, 4, 8, 16, 32). For every pair (simulation, core count), three quantities matter: throughput ($THP_i(k)$), simulation time ($ST_i(k)$), and utility (UTL_i). RATS tackles two optimization problems. First, it chooses, for each simulation, the “sweet-spot” number of cores—the value of k where doubling the cores from k/2 to k still yields a good throughput ratio but avoids diminishing returns. Second, if not all simulations can be started at once due to capacity constraints, it must schedule the simulations in time, subject to a global core capacity cap, so that the total completion time is minimized and high-utility treatments tend to finish earlier. This second problem is formulated as a multiple-choice knapsack: for each simulation, exactly one (simulation, k) choice is picked, and the sum of assigned cores at any time cannot exceed the capacity cap.

To solve these optimization problems, RATS needs good predictors for throughput, simulation time, and utility. It cannot, however, afford to run all 2,560 configurations just for training. Instead, employs Bayesian Optimization (BO) with Gaussian Process (GP) regressors. The RATS Modeler component uses a small simulation budget N (5% or 10% of all 2,560 configurations) and iteratively selects new configurations to benchmark by maximizing an acquisition function over the {TNF, cores} space. Each new micro-benchmark yields observations of throughput, simulation time, and utility; these are used to update three GPs, one per target. Practically, RATS uses a Rational Quadratic kernel (which empirically captured the variability of tumour simulations best), no explicit warm-up set, and only two acquisition functions: an LCB-type function for throughput and an EI-type function for simulation time, with utility modelled passively. This design dramatically reduces the number of required sample simulations without sacrificing GP accuracy, enabling useful models from as little as 5–10% of the configuration space.

Once the regressors are trained, the RATS Solver uses them to make decisions. For each simulation, it queries the throughput model over all allowed core counts and picks the k that maximizes a throughput-ratio-based objective, effectively identifying the optimal core allocation per simulation. This defines both the optimal core hours if all simulations are launched at once and their expected individual completion times. If the total number of cores required exceeds the available capacity, the RATS Solver turns to the second optimization stage. It first queries the utility regressor to obtain an expected utility for each simulation and sorts the simulation queue in descending utility. Then, in rounds, it solves a multiple-choice knapsack: it chooses a subset of simulations and core counts that fit within the capacity cap, favouring high utility and shorter simulated runtimes. The chosen simulations are submitted and, later on, once they complete, capacity is freed, and the solver iterates until all simulations are processed.

Using a ground-truth dataset of 2,560 simulations on MareNostrum 4 as reference, in [REF-14] and Deliverable D5.1, we show that with N = 10% (256 sample simulations), RATS approximates the optimal core hours within 3–9% error, while with N = 5% (128 samples), the error is around 10–18%. Under strict capacity constraints (cap = $5\%|SS|$), RATS matches the “Optimal” scheduling solution: it reduces total simulation time by up to four days in resource constrained environments and core hours by up to 15% to less constrained setups, compared to the best manual baseline (always using a fixed core count), while also achieving higher aggregate utility. These results justify the core design of RATS as a data-efficient, BO-driven resource allocator and scheduler for a single tumour study.

6.2.2 RATS+ Exploiting Transfer Learning

In practice, life scientists rarely run a single, fixed study. As early results come in, they may adjust TNF parameter ranges based on emerging hypotheses. For instance, after studying TNF concentrations in a certain range with a given step, they may suspect that higher doses could be more effective and extend the range to concentrations beyond this interval. The new study's domain therefore overlaps with, or fully contains, the old one. If RATS treated each study independently, it would have to re-run Bayesian Optimization from scratch on the expanded domain, wasting the simulations already executed in the previous study. RATS+ is introduced specifically to avoid this waste. Its goal is to reduce the new-study simulation budget N, reusing the BO models learned in the old study, without sacrificing the quality of the regressors.

The enhanced EVENFLOW contribution involves RATS+ [REF-15], which recasts this scenario using transfer-learning. It defines a source domain D_S: a subset SS' of the original simulation set SS with its own ranges for TNF frequency, duration, and concentration and full core configurations, and a target domain D_T: the new, expanded SS, which partially overlaps or fully contains the source domain. The learning task T is the same in both domains: predict throughput, simulation time, and utility for any {TNF, cores} configuration. RATS in the source domain has already trained GP regressors for these targets using BO. The question is how to transfer these regressors to the target domain to reduce the number of new micro-benchmarks needed there. RATS+ follows a modular approach: it first trains regressors on DS (as in RATS), then replays the BO calls made in D_S as the initial calls in D_T. This effectively seeds the target-domain optimizer with the source domain's GP, while leaving freedom for BO in D_T to explore new parameter regions that did not exist in SS'.

RATS+ operates under the same total budget N as RATS but splits it between source and target domains. Two main configurations are used experimentally: RATS+ (N = 3% + 2%), total N = 5%, where roughly 3% of the 2,560 configurations are sampled in the source domain and 2% in the target, and RATS+ (N = 6% + 4%), total N = 10%, where about 6% are sampled on the source domain and 4% on the target. In both cases, about 60% of the total budget is spent on the source domain and 40% on the new target domain. For comparison, we also consider RATS trained from scratch on the target domain with N = 10%, 5%, 4%, or 2%. A key practical decision is the choice of acquisition functions. For the original RATS Modeler in the source domain, Lower Confidence Bound (LCB) is used for throughput to ensure good mid-range exploration. For the extended RATS+ model in the target domain, Expected Improvement (EI) is preferred. The intuition is that uncertainty is highest in the newly introduced parameter regions; EI naturally directs sampling there, while still exploiting high-mean predictions inherited from the old domain.

6.2.2.1 RATS+ Experimental Highlights

The first RATS+ experiment [REF-15] assesses how well it can estimate the total optimal number of core hours needed to run all simulations in the extended study, compared to the optimal baseline. Six configurations are evaluated: RATS trained from scratch on the target domain with N = 10%, 5%, 4%, and 2%, and RATS+ with transfer using N = (6% + 4%) and N = (3% + 2%). The crucial observation is that RATS+ (6% + 4%) and RATS (10%) converge to the optimal solution with comparable accuracy, and RATS+ (3% + 2%) and RATS (5%) also behave very similarly, both achieving near-optimal performance. However, from the perspective of new simulations in the extended study, RATS (10%) uses 10% of 2,560 configurations (256 simulations) in the new domain, whereas RATS+ (6% + 4%) uses only about 4% (roughly 102 simulations) in the new domain. Similarly, RATS (5%) uses 128 new simulations, while RATS+ (3% + 2%) uses only about 51 new simulations. In both cases, RATS+ cuts the new-study budget by roughly 60% while achieving essentially the same accuracy in total core-hour estimation as its full-budget RATS counterpart.

The second RATS+ experiment embeds transfer learning into the full RATS pipeline, including capacity constraints. Two capacity settings are considered: cap = 5%|SS| (strict) and cap = 25%|SS| (moderate). For each cap, we measure how many simulations have completed over time, the aggregate utility of completed simulations over time, and how close the total execution time is to the optimal schedule. Across both capacities, all RATS and RATS+ configurations closely track the optimal completion time curve. More importantly, the aggregate utility curves show that RATS+ (6% + 4%) is almost indistinguishable from RATS (10%), and RATS+ (3% + 2%) is almost indistinguishable from RATS (5%), in terms of how quickly high-utility simulations are completed and how much utility is accumulated over time. This confirms that transfer learning does not distort scheduling decisions: with fewer new simulations, RATS+ still identifies and prioritizes the same promising treatment methodologies as a freshly trained, full-budget RATS model on the new domain.

We also compare the RATS GP regressors to Tabular Q-Learning and DQN under identical budgets (up to 256 episodes). As shown in [REF-15], Tabular Q-Learning fails to learn meaningful predictions. DQN improves but does not converge. By contrast, the GP models steadily reduce L1 error and drive correlation coefficient R^2 towards 1, supporting the choice of BO/GPR as the model that RATS and RATS+ transfer across studies.

Second, we design a scheduler inspired by Flux [REF-16], which uses a utility-sorted queue with dynamic core assignment and backfilling. Even when this baseline is given oracle knowledge of the true utilities, RATS' knapsack-based solver completes all 512 simulations tens to hundreds of hours earlier and with higher aggregate utility under both strict and moderate capacity caps. These comparisons make clear that the performance gains observed with RATS+ stand on top of an already strong modelling and scheduling foundation.

6.3 Synopses and Smart Sync for the Infrastructure Lifecycle Assessment Use Case

6.3.1 The Reverse Random Hyperplane Projection Scheme

In Wireless Sensor Networks (WSNs), as those deployed in EVENFLOW's Use Case III for Infrastructure Maintenance, the dominant energy cost is communication. Before reaching the point of performing neural or neurosymbolic tasks at a central base station, communication burden is incurred while collecting data from sensors to the base station. Sensors continuously produce streams by sampling quantities of interest from their realm, usually organized over fixed-size windows. Continuously transmitting all raw time series to a base station is expensive in terms of both bandwidth and battery life. Existing data compression approaches either rely on non-reversible random hyperplane projection (RHP) [REF-21][REF-22], which produces compressed bitmaps only suitable for similarity and outlier detection, or on reversible transforms such as DFT, DWT, DCT, and PAA [REF-17][REF-18][REF-19][REF-20], which can provide looser, deterministic error guarantees. In EVENFLOW, we introduce Reverse Random Hyperplane Projection (RRHP) [REF-23], which is designed to inherit the strengths of both lines of existing work on lightweight, reversible summaries. RRHPS uses RHP-style bitmaps for in-network compression, but also provides a principled way to reconstruct approximate real-valued vectors at a base station, together with explicit probabilistic (instead of deterministic) guarantees on the approximation of the original time series. In that, neural learning or broader data mining tasks can be performed on the approximated sensor time series.

The starting point for RRHP is the traditional RHP. Each window of sensor readings is modelled as a vector u ∈ $\mathbb{R}^{\omega}$ (ω denotes the window size). RHP constructs d random unit vectors r_1, …, r_d ∈ $\mathbb{R}^{\omega}$. Each r_k defines a hash function that maps u to one bit according to the sign of the dot product:

$$h_{r_k}(u) = \begin{cases} 1, if\ r_k \cdot u \geq 0 \\ 0, otherwise \end{cases}$$

A bitmap $X_u \in 0{,}1^d$ is formed by concatenating these d bits. For two vectors u and v with angle θ(u, v), the probability that they collide on a random hyperplane is:

$$P[h_r(u) = h_r(v)] = 1 - \frac{\theta(u,v)}{\pi}$$

Equivalently, the normalized Hamming distance between X_u and X_v approximates θ(u, v)/π. As such, RHP offers a one-way mapping $\mathbb{R}^{\omega} \to 0{,}1^d$ useful for similarity estimation, but it does not provide a way to reconstruct u from X_u.

RRHP [REF-23] augments RHP with a reverse mapping. All motes and the base station share a common random matrix $R \in \mathbb{R}^{\omega \times d}$ produced using a different, but equal seed for each time window. The columns of R are the random unit vectors r_1, …, r_d. For each window/vector u, RRHP can conceptually compute the projected vector $\xi = u \cdot R \in \mathbb{R}^d$ and then quantize it to a bitmap X_u via the above sign test. Only X_u (d bits) is transmitted by each sensor.

For reasonably large ω, R has full row rank almost surely, so a right inverse $R^{\dagger} \in \mathbb{R}^{d \times \omega}$ exists. At the base station, RRHP reconstructs an approximate vector:

$$\hat{u} = X_u \cdot R^{\dagger}$$

The key property is that angular similarities are preserved (in expectation) through compression and reconstruction. If θ(u, v) is the angle between original vectors and θ(û, v̂) the angle between their reconstructions, then RRHP shows that [REF-23]:

$$E\left[\frac{\theta(\hat{u},\hat{v})}{\pi}\right] = \frac{\theta(u,v)}{\pi}.$$

Thus, the cosine similarity $cos\big(\theta(u,v)\big)$ is approximately preserved. Since many measures such as Pearson correlation and Euclidean distance on normalized vectors can be expressed via cosine similarity, RRHP can support generic mining tasks on the reconstructed vectors $\hat{u}$. RRHP provides a Chernoff-style bound that relates the bitmap length d with the quality of angle estimation. To approximate the normalized angle between two vectors within an additive error ε with probability at least 1 − δ, it suffices to choose d proportional to log(1/δ)/ε². Thus, RRHP exposes a direct and tuneable trade-off between compression ratio and relative distance /reconstruction accuracy.

6.3.1.1 RRHP Experimental Evaluation

We tested RRHP on two settings. Initially we conduct experiments, examining the reconstruction quality of RRHP on several Machine Learning tasks (including neural learning), with compression ratios of 4 and 8. For stress testing RRHP on a high number of sensors, we use the Intel Lab Dataset[7] for clustering tasks and the Pump Sensor Dataset[8] for regression and classification.

Table 2: RRHP performance on various Machine Learning tasks, under a compression ratio of 8, over the reconstructed sensor time series, varying window sizes.

Window ω	**16**	**32**	**64**	**128**	**Average per Metric**
Feed Forward Neural Net	0.92	0.97	0.98	0.99	Classification Accuracy
DBSCAN Clustering	0.95	0.96	0.97	1.00	Clustering Similarity on Adjusted Rand Index (ARI)
Linear Regression	0.12	0.12	0.12	0.07	Root Mean Square Error
Support Vector Machine	0.92	0.97	0.98	0.99	Classification Accuracy
K-NN	0.83	0.96	0.98	0.99	Classification Accuracy

In our first experiment we showcase the accuracy of RRHP across different window sizes, on various mining tasks, by compressing sensor windows by a factor of 8, i.e., compression ratio is 8. In Table 2, we can see that the performance of RRHP significantly increases when we

[7] https://db.csail.mit.edu/labdata/labdata.html
[8] https://www.kaggle.com/datasets/nphantawee/pump-sensor-data

increase the window from 16 to 32 observations but, in all cited cases, RRHP yields high accuracy with respect to the metrics cited in the rightmost column of the table.

We also provide a comparison of RRHP vs DFT, DCT, DWT and PAA reversible data summaries, indicatively, for clustering sensor time series. This experiment tests RRHP on a clustering task, across sliding windows using the Intel Lab Dataset and reports the Adjusted Rand Index (ARI) score with 95% confidence interval. In Figure 47, we tested compression ratios of 4 and 8 across two rows of plots. As we can see in the figure, RRHP outperforms competition by, on average, 20% on both compression ratios, regarding the ARI score of the clustering task. Note that RRHP achieves also the tightest 95% confidence interval.

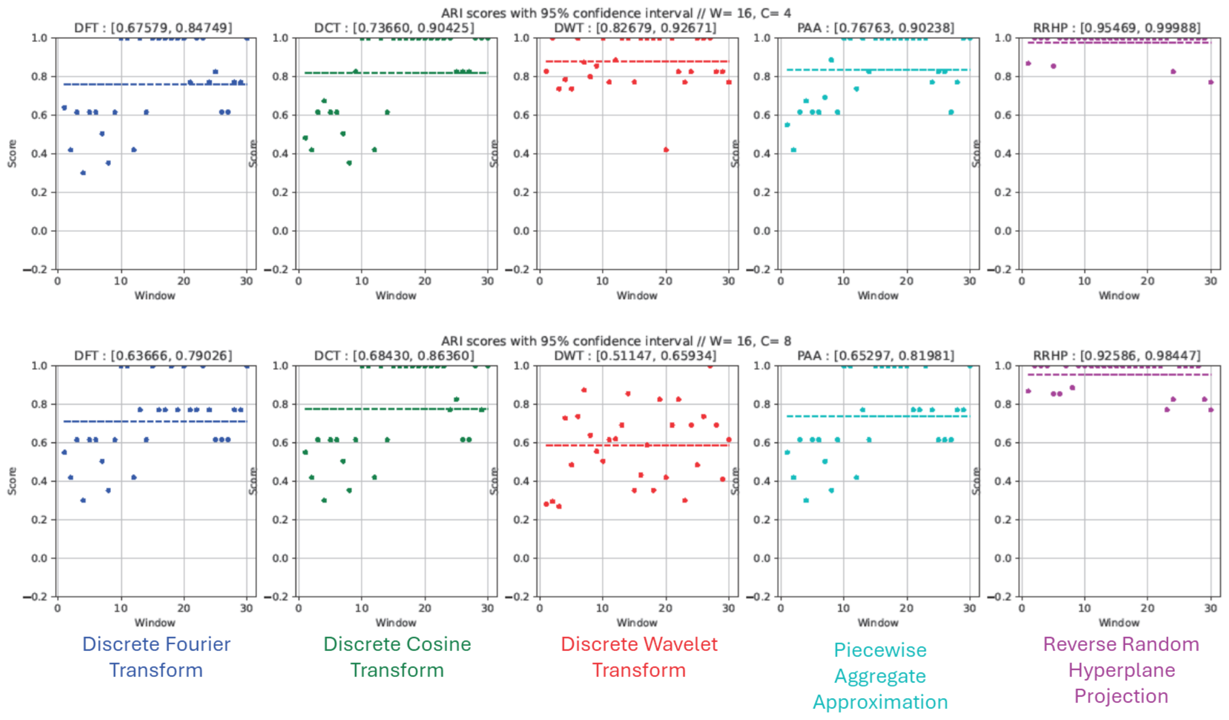


Figure 47: RRHP Performance on Clustering vs other Competitors.

We then conduct a WSN simulation, examining RRHPs ability to prolong the lifetime of sensors in a real scenario, using the TOSSIM simulator [REF-24]. The hierarchical network tested consists of 4 cluster heads with 12 sensors per cluster. The communication and lifetime gains plotted below hold irrespectively of the chosen Machine Learning task, since the task itself takes place at the base station after having collected the compressed sensor data streams and having recovered the RRHP-approximated ones.

In Figure 48, we see the total number of bytes that are transmitted during the simulation, including the ones of retransmissions. In Figure 49, we plot the total energy drain of the setups of Figure 48, essentially interpreting the communication costs to network lifetime. Retransmissions occur when multiple motes transmit their messages simultaneously, therefore resulting in corrupted messages due to collisions. Large message size and frequent communication are the main culprits of signal retransmissions. We observe that the compression ratios of 4 and 8, reduce the total number of bytes transmitted and the energy drain by 5 and 10 times, respectively. This increased reduction, compared to the theoretically expected reduction of 4 and 8 on both cases, is the combined result of the reduced message size of RRHP due to compression, and the reduced number of message collisions and retransmissions.

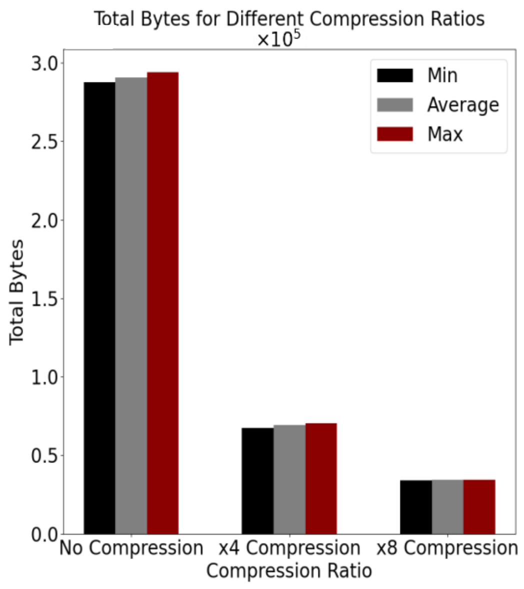


Figure 48: RRHP Comm. Reduction vs Compression Ratio using ω = 16.

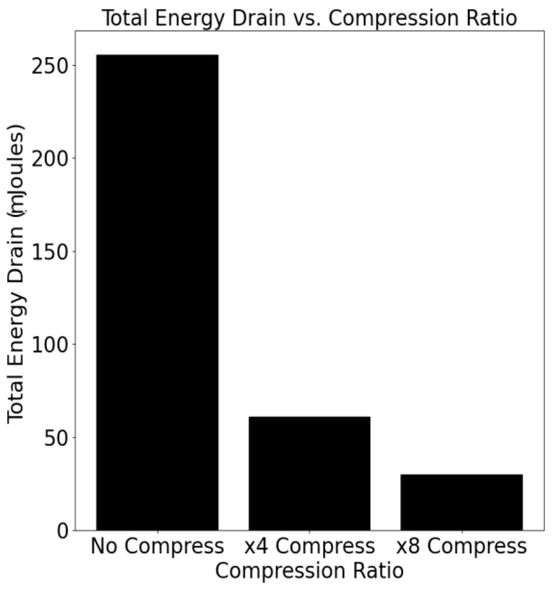


Figure 49: RRHP Network Lifetime vs Compression Ratio using ω = 16.

6.3.2 Uncertainty-aware Synchronization Protocols

In Section 3.5 we presented data-driven synchronization protocols that postpone a global model update until a considerable drift in the neural network weights, defined based on any given thresholded function, may exist. However, in EVENFLOW's Use Case III, the Infrastructure Maintenance Monitoring utilizes sensors which produce uncertain measurements, for instance, due to noise or calibration errors.

This uncertainty passes on to the weights of the neural network of the various distributed learner during the training process. Therefore, instead of monitoring whether a function f parameterized by the global weight vector has crossed the threshold, we instead want to know if f has crossed the threshold with sufficiently high confidence, provided we know the uncertainty distribution of learners' weights.

The Uncertainty-aware Global Monitoring (UGM) method addresses the problem of continuously monitoring a non-linear function over many distributed data streams whose values are uncertain. Each learner N_i produces a local data stream S_i, modelled as a time-varying random variable $x_i(t)$ with its own distribution $p_i(t)$. The Parameter Server (PS) wants to monitor a global function f(y(t)) of all streams and automatically raise an alert when f(y(t)) crosses a threshold T with high statistical confidence, while minimising communication.

To this end, each learner maintains a sliding window of recent observations, from which it estimates the parameters of its local distribution $p_i(t)$ (e.g., empirical mean and covariance). Learners are assigned weights $w_i \geq 0$ with $\sum_i w_i = 1$, and the global random vector at time t is $y(t) = \sum_i w_i x_i(t)$. The monitoring task is to decide, at any time t, whether f(y(t)) $\lesssim$ T holds with confidence at least δ, without continuously transmitting raw data or full local models to the Parameter Server. Since the techniques we describe henceforth are currently under submission, we provide only algorithmic sketches of their functionality.

In a centralized setting, the monitoring condition can be expressed as the global filter Pr[f(y(t)) ⋚ T] ≤ δ, which guarantees that an alert is raised only when the probability of violating the threshold exceeds δ. However, evaluating this global filter centrally would require frequent updates from all learners. UGM decomposes the global filter into local filters that can be evaluated independently at each learner. The PS periodically synchronises with the learners at times t_s and computes the global average vector $e(t_s) = \sum_i w_i o_i(t_s)$, where $o_i(t_s)$ is the last local average vector sent by learner i. Around this point, the coordinator constructs a convex "DONT-region" in the global feature space: a region that contains only "good" points, i.e., points where f(y(t)) is guaranteed to satisfy the monitored inequality with high confidence. As long as y(t) remains inside this DONT-region with sufficient probability, the global filter is satisfied and no alert is needed. The key design is to choose the DONT-region so that (i) it is convex, enabling rigorous decomposition into local conditions, and (ii) it is as large as possible, so that communication is reduced.

We instantiate the DONT-region as a d-dimensional ball $B(M, r) = x \in R^d\colon \left|\left|x - M\right|\right|_2 \leq r$ in the global feature space. The ball is constructed via a greedy sphere augmentation process: starting from a conservative initial ball around the current global mean, we iteratively increase its radius and slightly shift its center, as long as we can still guarantee that all points inside remain "good" (i.e., they satisfy the monitored inequality f(x) ⋚ T). The procedure stops when any further inflation or shift would introduce points that might violate the constraint. The resulting ball is a maximal convex DONT-region that can be safely decomposed into local filters at the individual nodes.

If no non-trivial sphere that contains only good points exists (e.g., when the global mean already lies near or beyond the threshold), the method enters a NO-SPHERE period; in this case, learners temporarily send updates eagerly, without local filtering.

Given the DONT-region, each learner N_i evaluates a local filter that decides whether to raise an alert. The learner maintains a set of drift vectors $D_i(t)$ that describe how its local distribution, translated near the last global mean, might move in the global space. Concretely, each drift vector has the form $d_j^i(t) = e(t_s) - o_i(t) + z_j$, where z_j is a sample drawn from the current local distribution $p_i(t)$. Each learner holds m such drift vectors per time t.

At time t, learner i estimates the probability that its local contribution keeps the global vector inside the DONT-region by counting the fraction of drift vectors that fall inside the sphere: $p_i(t)$ = # {drift vectors in don't-region} / m.

Each learner is associated with a confidence threshold $\alpha_i \in [0,1]$. If $p_i(t)$ falls below α_i, learner i has strong evidence that the global function may have crossed the threshold and sends an alert to the coordinator; otherwise, it remains silent. Initially, all learners share the same α_i, derived from δ and |N|, ensuring that if all local filters hold, then the global filter also holds. When the PS receives an alert from one or more learners, it distinguishes between true positives (TP), where f(y(t)) has indeed crossed the threshold, and false positives (FP), where local filters were overly conservative. Only TP alerts trigger an application-level reaction (e.g., reconfiguration, mitigation).

As the number of learners grows, having a uniform confidence threshold α_i can become overly strict, causing excessive alerts and communication. A Slack Allocation mechanism dynamically redistributes "slack" among learners by adapting their local thresholds α_i after each synchronisation, based on their recent behaviour.

Learners are partitioned into GOOD and BAD sets depending on the last alert:

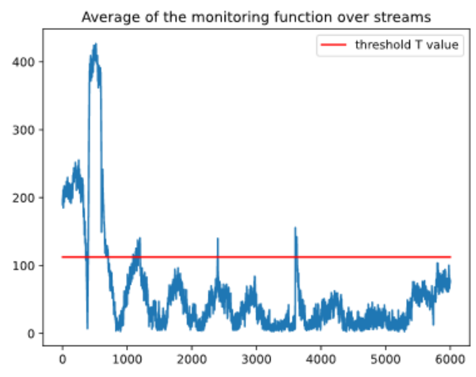
- In the FP case, learners that raised unnecessary alerts are marked as BAD, and learners that remained silent are GOOD. The coordinator computes a slack measure for each GOOD learner, reflecting how comfortably its local probability $p_i(t)$ was above its threshold α_i. It then gradually lowers the thresholds of BAD learners (making them less sensitive) and compensates by slightly raising the thresholds of selected GOOD learners, preserving a global probabilistic guarantee.
- In the TP case, learners that failed to alert in time are BAD, while learners that correctly alerted are GOOD. A symmetric slack definition is used, and thresholds are adjusted in the opposite direction, making BAD learners more sensitive.

This iterative procedure continues until all learners satisfy the global constraint on the joint probability of violation. In practice, Slack Allocation significantly reduces communication by allowing learners whose behaviour is well contained within the DONT-region to take on stricter thresholds, freeing other learners to relax theirs.

6.3.2.1 UGM Experimental Evaluation

For our experiments we utilize a 10-sensor dataset provided by EKSO using 5 computing nodes. We monitor L1, L2 and Variance functions posing the threshold as shown in Figure 50 and Figure 51. In each experiment we present 5 bars with the vertical axis showing the number of transmitted messages, while the horizontal axis varies δ:

- Naïve: representing continuous model updates
- Sphere Max Off, Slack Allocation Off: which is the basic UGM protocol without any optimization
- Sphere Max On, Slack Allocation Off: the protocol that uses the maximal, augmented sphere according to our discussion in Section 6.3.2, but does not perform wise Slack Allocation
- Sphere Max On, Slack Allocation On: the protocol that uses both the maximal augmented sphere and performs wise Slack Allocation



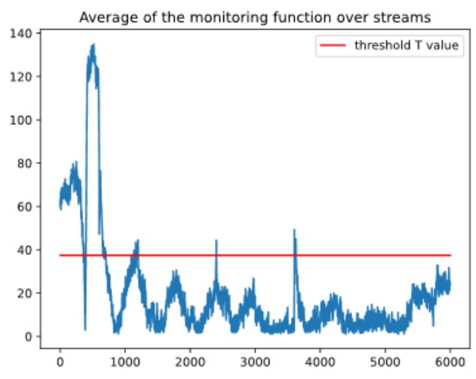


Figure 50: L1 function (left) and L2 function (right) Threshold (red line) vs actual function values as time passes.

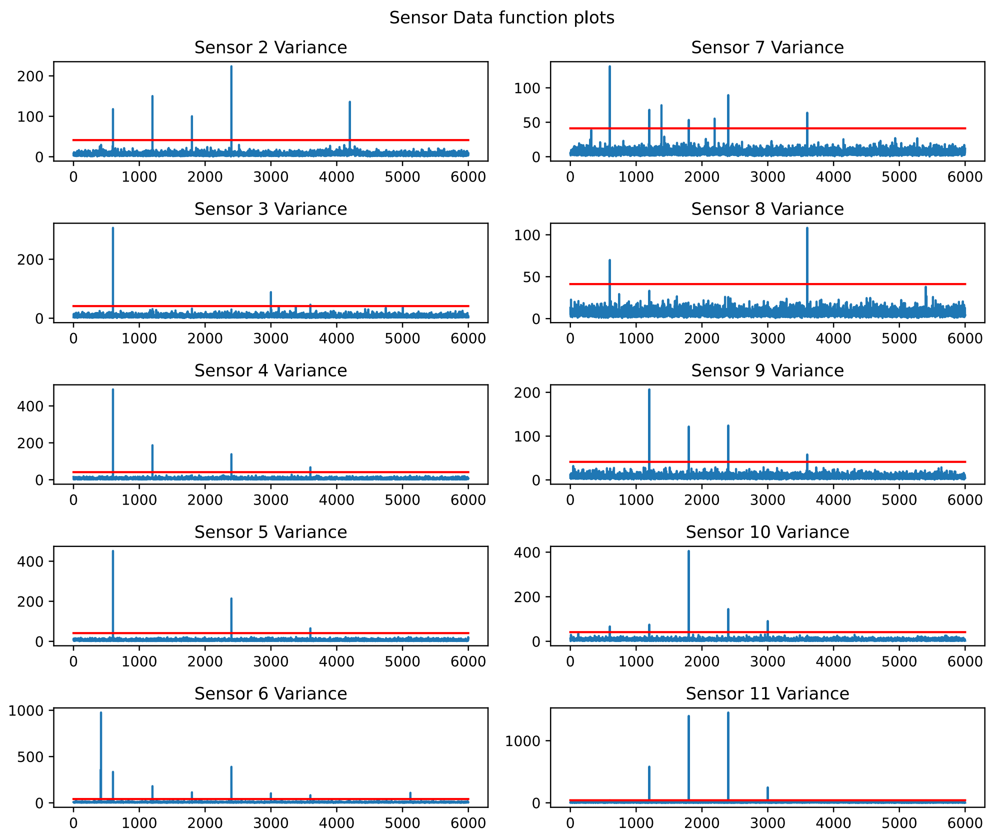


Figure 51: Variance Threshold Selection based on Sensor violations.

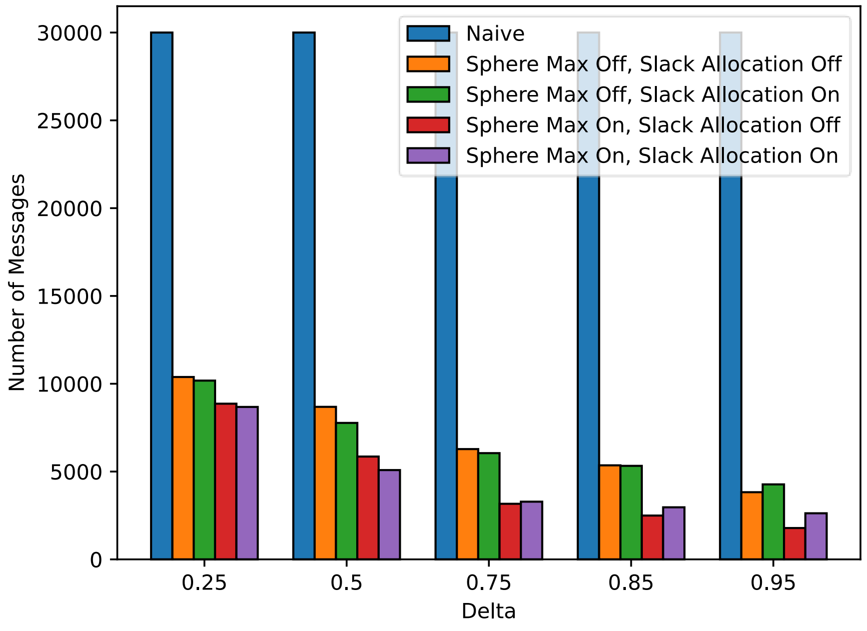


Figure 52: Number of transmitted messages for L1-based monitoring across δ values.

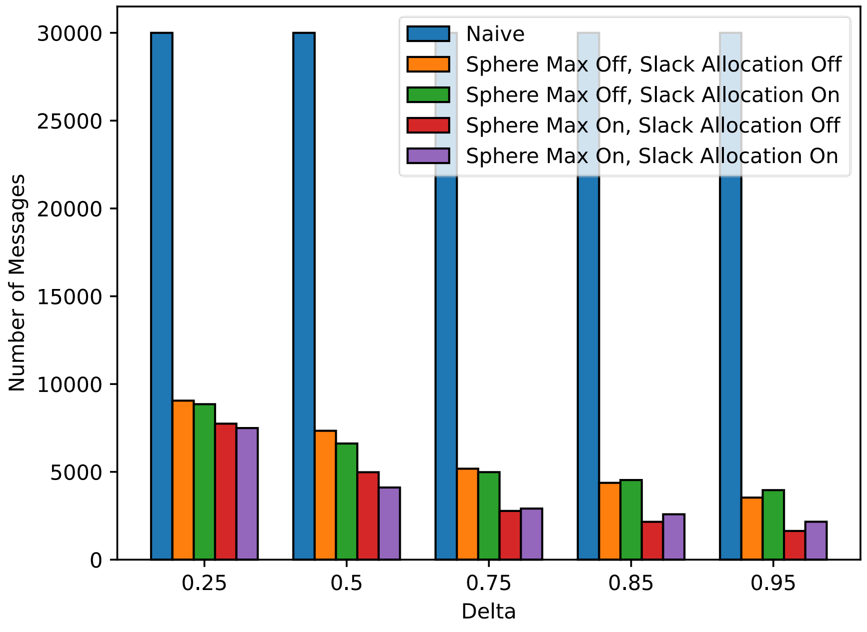


Figure 53: Number of transmitted messages for L2-based monitoring across δ values.

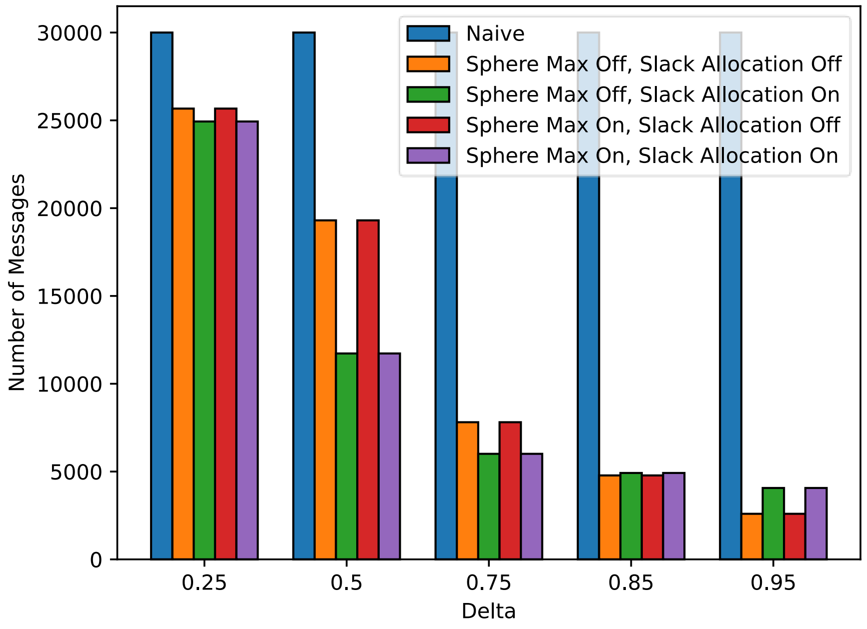


Figure 54: Number of transmitted messages for Variance-based monitoring across δ values.

Across Figure 52, Figure 53 and Figure 54 we observe that UGM “Sphere Max Off, Slack Allocation Off” improves the Naïve approach yielding 17% (Var) to 3x fewer messages for loose δ = 0.25. The communication gains progressively increase for δ=0.5 to δ=0.95, reaching values of reaching 10x communication gains for δ=0.95.

“Sphere Max Off, Slack Allocation On”, “Sphere Max On, Slack Allocation Off” as well as “Sphere Max On, Slack Allocation On” further reduce the amount of communicated messages up to 50% compared to “Sphere Max Off, Slack Allocation Off”. As we can observe in the figures, setting On or Off some optimization may occasionally increase communication although more optimizations are applied. For instance, in Figure 52 for δ=0.5, setting “Sphere Max Off, Slack Allocation On” gives a few more messages compared to “Sphere Max Off, Slack Allocation Off” or “Sphere Max On, Slack Allocation On” gives few more messages compared to “Sphere Max On, Slack Allocation Off”. This is because, when we apply each optimization, the synchronization timepoints differ between experiments. Therefore, a false positive synchronization that happens at a specific time, changes o_i and cascades the timepoints at which future synchronization will take place. As such, a difference series of synchronization timepoints may yield a slightly different behaviour in the monitoring process.

7 EVENFLOW Verification Approach

7.1 Formal Verification of Neural Networks

Deep neural networks have become the dominant paradigm for modelling complex functions across vision, language, control, and decision-making tasks. Their power stems from the universal approximation property [REF-27]: with sufficient capacity, a neural network can approximate virtually any continuous function to arbitrary precision. Yet, this expressive capability comes with a fundamental drawback—the internal workings of neural networks are opaque and highly nonlinear, making it difficult to reason about how they will behave under perturbations, distribution shifts, or intentionally manipulated inputs. Even small amounts of noise, such as adversarial perturbations or sensor uncertainty, can cause unexpected deviations in the network's output.

A neural network represents a mapping $f: \mathbb{R}^n \to \mathbb{R}^m$, taking real-valued inputs and producing real-valued outputs. While this functional view is mathematically elegant, it does not provide direct insight into the internal decision logic learned during training. Formal specifications describe what the network should do for all possible inputs within a given domain. These properties capture the intended safe or acceptable behaviour of the system without requiring explicit interpretability of its internal parameters. A property is considered satisfied when the network is guaranteed to behave safely for every input in a defined set, not just for those seen during testing.

Formal verification aims to provide provable guarantees about network behaviour over all possible perturbations within a defined region (e.g., an $\ell\infty$-ball around an input). Ensuring the robustness of neural network classifiers involves demonstrating that the model's predicted label does not change when its input is perturbed within a small region of radius ϵ [REF-52]. Formal verification methods address this by reasoning over all possible inputs within this perturbation region, effectively providing mathematical certificates that guarantee robustness. For a neural network f, the robustness requirement can be stated as follows: for any input x that the network classifies correctly, and for every perturbed input x'satisfying $\| x - x' \| \leq \epsilon$, the prediction must remain unchanged, i.e., $f(x) = f(x')$.

This verification problem can be reduced to analysing the relationships between the unnormalized outputs (logits) of the network's final layer. Specifically, robustness around $x$is guaranteed if, for every x'in the ϵ-ball, the logit for the true class y_{true}exceeds the logits for all other classes $y_i$by a positive margin:

$$y_{\text{true}} - y_i > 0 \text{ for all } i \neq \text{true class.}$$

Checking this requires computing the *minimum* of these logit differences over all possible x'within the perturbation set. If this minimum remains positive, the classifier is provably robust for that radius ϵ [REF-29]. However, computing this minimum exactly is an NP-hard optimisation problem [REF-30], which makes robustness verification computationally challenging in practice.

7.2 Scalable approach towards Probabilistic Neuro-Symbolic Verification

Neuro-symbolic (NeSy) systems bridge deep neural networks with symbolic reasoning to achieve generalization, interpretability, and structured inference. In probabilistic variants of these systems, neural networks extract latent concepts from raw inputs, and a symbolic reasoning layer performs probabilistic inference using logical constraints. For probabilistic NeSy AI architectures [REF-28], formal verification is even more difficult, as the symbolic component requires evaluating weighted model counts (WMC) over latent concept probabilities.

7.2.1 Probabilistic Neuro-Symbolic Verification

We now formalize the goal of relaxation-based methods within neuro-symbolic (NeSy) reasoning systems. For a given NeSy model, as introduced in Section 7.1, the objective is to determine the following quantities:

$$\min_{x'} \; p(y_i \mid x'), \max_{x'} \; p(y_i \mid x') \text{ for all } x' \text{ such that } \| x' - x \| \leq \epsilon$$

for every output label $y_i$in y. In other words, we seek to compute tight lower and upper bounds on each probabilistic output of the NeSy system when its input is subject to perturbations of radius ϵ. As outlined in Section 7.1, these bounds can then be used to formally evaluate the robustness of a particular input instance.

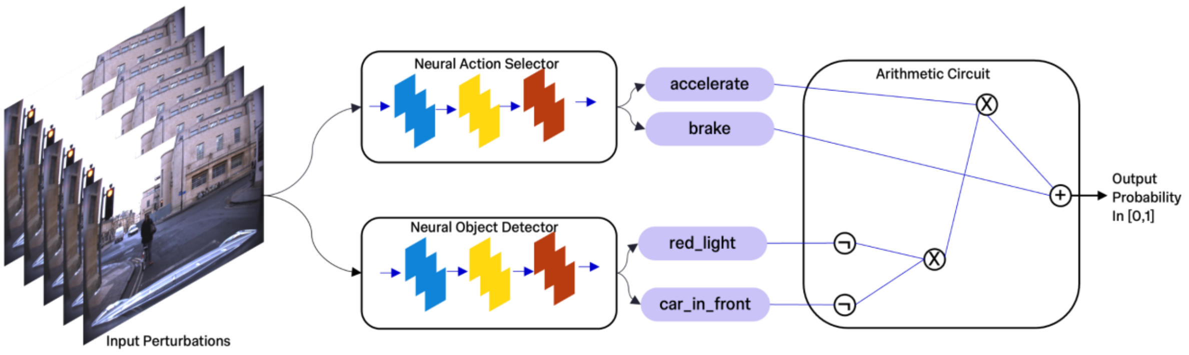


Figure 55: Probabilistic NeSy Verification illustrating the Verification of NeSy system trained on the ROAD-R dataset. The symbolic constraints are encoded as an Arithmetic circuit.

To illustrate, consider the NeSy architecture shown in Figure 55. Its neural components consist of two networks: (1) an object detector that identifies whether a red traffic light or a car is present in front of an autonomous vehicle (AV), and (2) an action-selection network that decides whether the AV should accelerate or brake. The symbolic component encodes a conjunction of two logical constraints, detailed in Appendix A. For an input image x, the system outputs y, representing the probability that these symbolic constraints are satisfied.

An input instance is deemed robust if

$$\min_{x'} p(y \mid x') > T,$$

for some threshold $T \in [0,1]$. This condition states that, across the entire ϵ-ball surrounding x, the probability of satisfying the symbolic constraints never drops below T. For the purposes of the remainder of this work, we focus on the case $T = 0.5$.

7.2.1.1 Relaxation-Based Approach

Because computing exact bounds through the compiled symbolic component is computationally intractable, we turn to relaxation-based methods. These techniques can be naturally extended to neuro-symbolic (NeSy) systems and offer a scalable pathway for solving the equation in Section 7.2.1.

In the NeSy architectures, the neural network outputs serve as inputs to an arithmetic circuit. Owing to this compositional structure and the algebraic nature of the circuit, the entire NeSy model can be treated as a fully differentiable computational graph. This enables us to implement the whole system as a single module within common machine learning frameworks such as PyTorch. The resulting model can then be exported into the Open Neural Network Exchange (ONNX) format [REF-31].

The ONNX format is widely supported by state-of-the-art neural network verification tools, including solver-based systems such as Marabou [REF-32] and relaxation-based verifiers like auto-LiRPA [REF-33] and VeriNet [REF-34]. By exporting a NeSy model to ONNX, we can leverage these tools with minimal additional engineering effort, effectively enabling "plug-and-play'' verification of complex neuro-symbolic pipelines.

Although the framework is compatible with a broad range of verifiers, the focus is on relaxation-based methods to demonstrate scalable probabilistic verification for NeSy systems. These methods allow us to apply input perturbations and compute bounds directly on the system's final outputs—bypassing the need to derive intermediate bounds on individual neural network components.

7.3 Experimental Evaluation

In this section, an empirical evaluation of the proposed verification framework is presented, focusing on both its effectiveness and practical applicability. The scalability of the method is first examined using a synthetic task derived from MNIST addition, a widely adopted benchmark in the neuro-symbolic reasoning literature [REF-35]. To further demonstrate real-world relevance, the approach is applied to an autonomous driving dataset, where a safety-critical driving property is verified over two six-layer convolutional neural networks. This experiment highlights the capability of the proposed technique to manage high-dimensional inputs and larger neural architectures—conditions commonly encountered in operational autonomous systems. All experiments were conducted on a high-performance computing machine equipped with 128 AMD EPYC 7543 32-core processors (3.7 GHz) and 400 GB of RAM.

7.3.1 Multi-Digit MNIST Addition

In this experiment, the primary objective is to examine how well the proposed method scales as the complexity of the symbolic reasoning component increases. As symbolic structures grow larger and more intricate, they naturally introduce greater computational demands;

therefore, understanding how our approximate verification strategy behaves under these conditions is crucial. The experiment is designed to shed light on two key aspects: (1) how the inherent approximations used in our method enable significantly improved scalability, and (2) what trade-offs emerge in the accuracy and tightness of the resulting verification guarantees.

To systematically study these effects, we compare our approach against two distinct verification paradigms:

1. **End-to-End Relaxation-Based Verification (E2E-R):** This approach corresponds to a direct implementation of our proposed method using auto-LiRPA, a leading relaxation-based neural network verification toolkit. In this setting, the entire neuro-symbolic system—including both the neural and symbolic components—is provided as input to auto-LiRPA. The system is internally converted into an ONNX computational graph, enabling uniform processing and bound propagation across the full pipeline. We employ Interval Bound Propagation (IBP), as implemented within auto-LiRPA, to compute end-to-end bounds on the perturbed outputs. This baseline allows us to evaluate how a fully relaxed, approximate strategy behaves when applied to complete NeSy models.
2. **Hybrid Verification (R+SLV):** The second baseline adopts a hybrid strategy that combines relaxation-based analysis for the neural modules with solver-based exact bound computation for the symbolic reasoning layer. More concretely, IBP in auto-LiRPA is used to derive bounds on the outputs of the neural networks, while the symbolic component is processed using exact optimization. Following the transformation of the symbolic circuit into an equivalent polynomial representation, we perform constrained optimization using the Gurobi solver to compute tight bounds. This hybrid method serves as a point of comparison to assess the trade-off between computational scalability and verification precision. Contrasting this approach with E2E-R highlights the relative benefits of exact symbolic reasoning versus the efficiency of fully relaxed, approximate propagation.

7.3.1.1 Dataset and Experimental Setup

To examine scalability, a synthetic dataset is curated that allows precise control over the complexity of the symbolic reasoning component while keeping the neural architecture fixed. Specifically, we construct a modified version of the multi-digit MNIST addition benchmark [REF-35]. In this setting, each data instance is composed of several MNIST digit images and is annotated with the arithmetic sum of the depicted digits. By varying the number of digits included in each sample, we can directly manipulate the size and depth of the symbolic computation required; for example, in a 3-digit addition scenario, a single instance may consist of images such as , , and , with a corresponding label of *18*. The verification dataset is derived from the 10,000 samples of the MNIST test set, ensuring that each image is used exactly once. Consequently, the number of verification instances for a given choice of #digits is 10,000 divided by the number of digits per sample, enabling controlled experimentation across symbolic complexities of increasing scale.

The experimental setup employs a convolutional neural network (CNN) trained to classify individual MNIST digits. Using the full MNIST training set of 60,000 images in a supervised learning setup, the model achieves a test-set accuracy of 98%. The symbolic reasoning component encodes the arithmetic rules governing multi-digit addition. It takes as input the probabilistic digit predictions produced by the CNN and infers a probability distribution over all possible sums. As the number of summands increases, the size and complexity of the reasoning circuit grow accordingly, since more combinatorial pathways exist for generating a given total (for example, there are multiple ways in which 2-digit, and 5-digit combinations can yield the sum of 17).

To evaluate scalability under increasing symbolic complexity and varying levels of input uncertainty, we systematically vary both the number of digits per instance and the magnitude of input perturbation. Specifically, we consider five configurations for the number of digits: {2, 3, 4, 5, 6}, and three perturbation budgets $\epsilon \in \{10^{-2}, 10^{-3}, 10^{-4}\}$, resulting in fifteen distinct experimental settings. Each experiment—defined by a particular (#digits, ϵ) pair—is executed with a timeout threshold of 72 hours. The end-to-end relaxation-based method (E2E-R) is executed on a single computational thread, whereas the hybrid solver-based approach (R+SLV) leverages the Gurobi optimizer, which dynamically allocates up to 1024 threads to accelerate the constrained optimization phase.

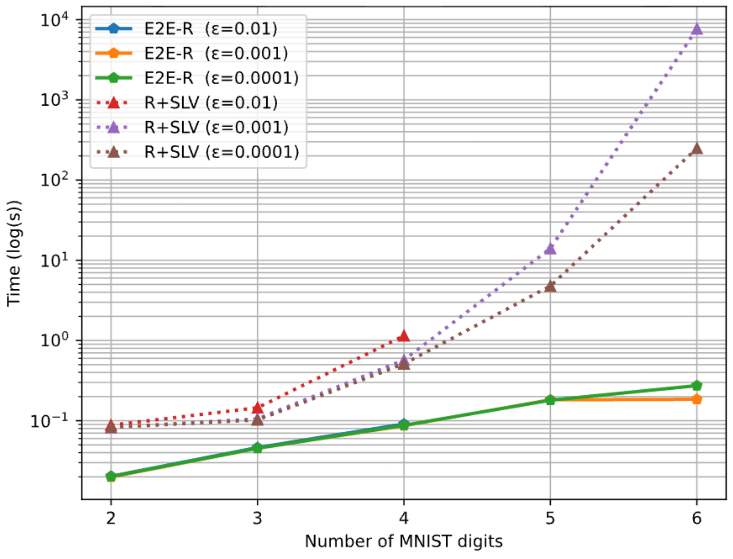


Figure 56: Comparison of verification runtimes for E2E-R and R+SLV. The experiments are evaluated for 3 different ϵ perturbations.

7.3.1.2 Scalability of the approaches and verification results

Figure 56 provides a comparative analysis of the scalability of the two verification methods. The plot reports the average time required to verify the robustness of a single NeSy instance across the test dataset. All experimental configurations complete within the 72-hour timeout, with two notable exceptions for the R+SLV approach. Specifically, for the configurations ⟨$\epsilon = 10^{-2}$, #digits = 5⟩ and ⟨$\epsilon = 10^{-2}$, #digits = 6⟩, R+SLV fails to verify any sample before the timeout, which explains why the corresponding curves for $\epsilon = 10^{-2}$ terminate at four digits. Additionally, for ⟨$\epsilon = 10^{-3}$, #digits = 6⟩, R+SLV can verify fewer than 5% of the inputs; the values shown in Figure 56 therefore reflect the average runtime computed over this small subset.

As clearly illustrated by the log-scale plot, E2E-R exhibits dramatically superior scalability compared to R+SLV. This advantage stems from the inherent computational burden of performing exact bound propagation through the symbolic probabilistic reasoning component. For context, verifying the robustness of the CNN alone using Marabou requires an average of 314 seconds per sample across 100 MNIST test images—demonstrating that solver-based verification of even a single neural module is several orders of magnitude slower than our full end-to-end relaxation-based approach.

These findings are consistent with both theoretical results [REF-36] and recent empirical analyses [REF-37] which highlight the limited scalability of SMT-based verification techniques. The experimental outcomes strongly indicate that, in the NeSy setting—where verification routinely involves multiple neural networks coupled with complex symbolic reasoning—the trade-off favouring approximate but scalable methods is not only acceptable but essential.

We next analyse how increasing the complexity of the symbolic reasoning component influences the quality of the verification outcomes. Table 3 summarises two key metrics across different experimental configurations. First, we report the tightness of the computed output bounds, expressed as the lower–upper interval for the probability assigned to the *correct* sum, averaged over all samples in the test set. This metric provides insight into how precisely each method can characterise the behaviour of the NeSy system under perturbations. Second, we evaluate robustness, defined as the proportion of test instances for which the system remains provably robust, i.e., the number of verified robust samples divided by the total number of samples. Together, these metrics enable us to assess how symbolic complexity impacts both the accuracy and reliability of the verification process.

Table 3: Verification method performance for MNIST digit addition at $\epsilon = 0.001$.

Verification Method	**Metric**	**#MNIST digits**			
		2	**3**	**4**	**5**
R+SLV	Lower/Upper Bound	0.871-0.981	0.815-0.972	0.764-0.962	0.731-0.928
	Robustness (%)	90.60	86.17	81.33	78.31
E2E-R	Lower/Upper Bound	0.871-0.982	0.815-0.974	0.763-0.965	0.716-0.958
	Robustness (%)	90.60	86.11	81.21	76.67

7.3.2 Autonomous driving - ROAD-R

In this experiment, we evaluate our proposed verification framework on a real-world dataset drawn from the autonomous driving domain. The objective is to assess the robustness of a neural autonomous driving system with respect to the safety and commonsense constraints illustrated in Figure 55. Specifically, we aim to determine whether small perturbations to the input can cause the underlying neural components to violate logical constraints that were originally satisfied. This setting enables us to examine how well the system maintains correct and safe behaviour under realistic variations in sensory input, thereby providing a practical test of robustness in a domain where reliability is critical.

7.3.2.1 Dataset and experimental setup

To conduct this evaluation, we make use of the ROad event Awareness Dataset with logical Requirements (ROAD-R) [REF-38]. This dataset contains 22 dashcam videos recorded from the perspective of an autonomous vehicle (AV), with each frame annotated using bounding boxes that identify agents (e.g., pedestrians), the actions they are performing (e.g., approaching the vehicle), and their spatial locations (e.g., on the right pavement).

For the experiments, we restrict attention to the subset of frames that are relevant to the symbolic constraints. Specifically, we select only those frames that satisfy either of the following conditions:

1. the AV is moving forward, and the scene contains neither a red traffic light nor a stopped car in front; or
2. the AV is stationary, and either a red traffic light or a stopped car is present.

Sampling every two seconds across the videos yields a curated dataset of 3,143 instances. Each sample consists of a $3 \times 240 \times 320$ RGB image and four associated binary labels: *red light*, *car in front*, *stop*, and *move forward*.

The neural component of the NeSy system includes two six-layer convolutional neural networks, one performing object detection and the other determining the appropriate driving action. Both models are trained using an 80/20 train–test split over the selected frames. The object detection network attains 97.2% accuracy, while the action selection network reaches 96.3% accuracy on their respective test sets. To evaluate robustness, we perturb the test images using five perturbation magnitudes, $\epsilon \in \{10^{-5}, 5{\times}10^{-5}, 10^{-4}, 5{\times}10^{-4}, 10^{-3}\}$.

Table 4: ROAD-R network verification results, indicating robustness for various epsilon noise.

Metric	**Epsilon**				
	1e-5	**5e-5**	**1e-4**	**5e-4**	**1e-3**
Robustness (%)	96.82%	92.68%	82.64%	6.21%	0.00%
Runtime per Sample (s)	0.091	0.092	0.091	0.092	0.092

Table 4 summarizes the experimental results. We report two key metrics: **robustness**, defined as the proportion of test instances that are provably robust, and the **verification runtime** for the E2E-R method. Because this task involves a relatively small symbolic arithmetic circuit paired with a substantially larger neural component, the computational cost and bound-propagation error are dominated by the neural network rather than the symbolic reasoning module. Consequently, E2E-R and R+SLV—which differ only in how the symbolic component is handled—yield nearly identical outcomes, and we therefore omit the redundant R+SLV results from the table. As anticipated, the verified robust accuracy decreases as the perturbation magnitude increases. With respect to runtime, the findings further support the trends observed in Section 7.3.1.2: the overall verification time for our approach is largely insensitive to changes in the perturbation bound ϵ, underscoring its practical scalability even under varying adversarial budgets.

7.4 Complex Event Verification for Temporal Neuro-Symbolic Models

While Sections 7.2 and 7.3 address the verification of static neuro-symbolic models, EVENFLOW use cases often rely on streaming data, necessitating the verification of temporal neuro-symbolic systems. This introduces an additional axis of complexity: time. In such systems, a neural network processes perception data at each time step t, extracting high-level attributes (symbols) that feed into a symbolic component—specifically a finite state automaton—which maintains state over a sequence.

Verifying properties in this context requires unrolling the system over a time horizon T. This effectively creates a deep computational graph where errors and uncertainties can compound rapidly. A primary challenge identified in temporal verification is that standard Interval Bound Propagation (IBP) becomes insufficiently precise. As the system is unrolled, the coarse over-approximations of IBP accumulate, leading to "exploded" bounds that fail to verify valid properties.

7.4.1 Verification Methodologies

To address these challenges, we move beyond simple IBP end-end verification and evaluate three distinct methodologies for verifying temporal neuro-symbolic systems. These techniques range from purely neural approaches to hybrid optimization techniques:

1. **End-to-End Verification:** The entire temporal neuro-symbolic system (neural network + unrolled automaton) is compiled into a single, end-to-end differentiable computational graph. Standard neural verification tools (e.g., auto_LiRPA) are then applied using IBP or CROWN (a linear bound propagation method).
2. **IBP/CROWN + Step Gurobi:** This is a decomposed approach. The neural network is verified first using IBP or CROWN to obtain bounds on the symbolic inputs. The automaton verification is then handled by solving a local optimization problem at each time step using the Gurobi solver. Because the symbolic interaction is multilinear, the resulting optimization problem is quadratic. We evaluate using both *Exact* and *Relaxed* solving, where the latter employs McCormick relaxations to handle bilinear constraints.
3. **IBP/CROWN + Unrolled Gurobi:** Similar to the stepwise approach, but the verification of the automata is constructed as one large, monolithic optimization problem covering the entire trace. Due to the scale of unrolled traces, this is solved only using relaxed McCormick constraints (Linear Programming).

7.4.2 Evaluation Scenario: Temporal Complex Event Recognition

To validate the proposed verification techniques, we utilize a Temporal Complex Event Recognition task on video sequences. This scenario is designed to stress-test temporal neuro-symbolic reasoning by requiring the system to identify complex patterns over a stream of inputs. As illustrated in Figure 57, the example system is composed of the following stages:

1. **Input Stream:** A sequence of standard MNIST digit images (e.g., 7, 8, 2, 1, 4).
2. **Multi-Attribute Perception:** Unlike simple digit classification, the CNN here acts as a multi-head classifier that predicts specific semantic attributes for each frame:

   a. **Parity:** Classifies the digit as *Odd* or *Even*.
   b. **Magnitude:** Classifies the digit into ranges: $x < 3, 3 < x < 6$, or $x > 6$.
3. **Temporal Reasoning:** The sequence of attribute probabilities is processed by a finite state automaton that monitors for a specific complex event pattern: *"An even digit larger than 6, followed eventually by an odd digit smaller than 6, followed eventually by a digit smaller than 3".*
   a. **State S0 (Start):** Loops until it detects an input satisfying $even \wedge larger_than_6$(e.g., digit 8), transitioning to S1.
   b. **State S1:** Loops until it detects $odd \wedge smaller_than_6$ (e.g., digit 1, 3, 5), transitioning to S2.
   c. **State S2:** Loops until it detects $smaller_than_3$ (e.g., digit 0, 1, 2), transitioning to the accepting state S3.

Robustness in this scenario requires proving that no combination of adversarial perturbations across the input image sequence can alter the final acceptance/rejection decision of the automaton, where S3 is the acceptance state.

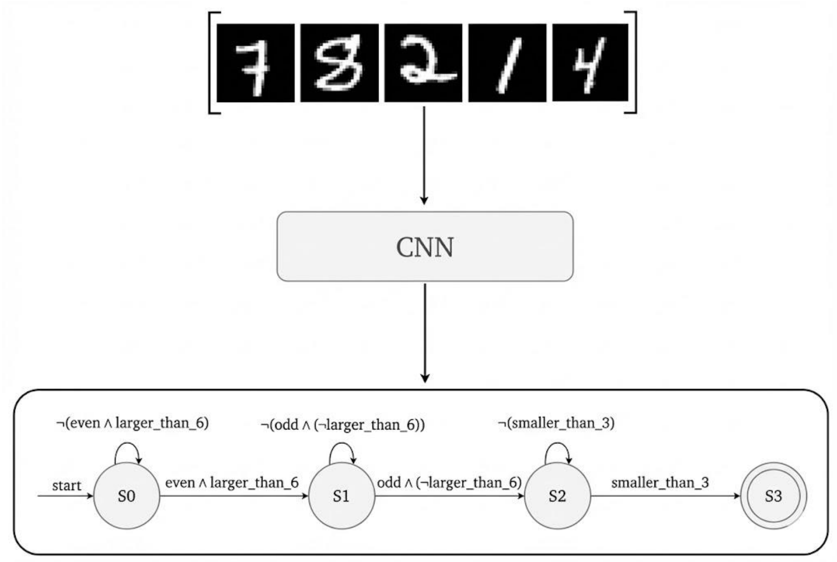


Figure 57: The Temporal Complex Event Recognition evaluation scenario. A sequence of MNIST digits is processed by a multi-head CNN to extract semantic attributes (Parity and Magnitude), which trigger transitions in a finite state automaton.

7.4.3 Experimental Evaluation of Temporal Verification

We evaluated the robustness of these methodologies against adversarial perturbations (ϵ) ranging from 10^{-5} to 10^{-2}. The comprehensive results are detailed in Table 5.

Table 5: Verification Accuracy and Execution Time for Temporal Neuro-Symbolic Systems.

Method Type	**Specific Method**	**Variation**	**ϵ=10−5**	**ϵ=10−4**	**ϵ=10−3**	**ϵ=10−2**
End-to-End	IBP	-	50.20% (09s)	44.22% (09s)	20.35% (09s)	-
	CROWN	-	99.80% (26m)	99.95% (26m)	81.96% (25m)	-
	CROWN-IBP	-	59.05% (12s)	44.22% (12s)	20.45% (10s)	-
Step Gurobi	IBP	Exact	99.69% (2m)	98.21% (2m)	51.74% (5m)	0.00% (2m)
	IBP	Relaxed	99.69% (1m)	98.01% (1m)	51.23% (1m)	0.00% (1m)
	CROWN	Exact	100.0% (3m)	100.0% (4m)	99.28% (4m)	86.96% (5m)
	CROWN	Relaxed	99.95% (2m)	99.95% (3m)	99.34% (3m)	86.81% (3m)
Unrolled Gurobi	IBP	Relaxed	99.69% (1m)	98.06% (1m)	51.28% (1m)	0.00% (1m)
	CROWN	Relaxed	99.95% (2m)	99.95% (3m)	99.34% (3m)	86.86% (3m)
	alpha-CROWN	Relaxed	-	-	-	**88.55% (3h 41m)**

7.4.4 Analysis of Results

The experimental data reveals distinct performance tiers among the verification strategies:

- **Inefficacy of IBP:** Pure IBP-based methods struggle significantly with temporal depth. The End-to-End IBP verification accuracy collapses from ~50% at $\epsilon = 10^{-5}$ to ~18% at $\epsilon = 10^{-3}$. Even when combined with symbolic optimization (IBP + Step/Unrolled Gurobi), IBP bounds are too loose to support verification at higher perturbations ($\epsilon = 10^{-2}$ results in 0.00% accuracy).
- **Robustness of CROWN-based Hybrids:** Using CROWN bounds results in far greater resilience. The *CROWN + Step Gurobi (Relaxed)* and *CROWN + Unrolled Gurobi (Relaxed)* approaches maintain near-perfect accuracy (>99%) up to $\epsilon = 10^{-3}$. At $\epsilon = 10^{-2}$, where IBP methods fail completely, CROWN-based unrolling retains approximately **86.6%** verification accuracy.
- **High-Precision Trade-off:** For the most demanding scenarios ($\epsilon = 10^{-2}$), the specialized **alpha-CROWN + Unrolled Gurobi** method achieved the highest robustness of **88.55%**. However, this precision incurs a massive computational cost, requiring over 3 hours and 41 minutes compared to just a few minutes for the relaxed CROWN approaches. We note that we utilize the computationally intensive *alpha-CROWN* method exclusively for the largest perturbation level ($\epsilon = 10^{-2}$). For smaller perturbations ($\epsilon \leq 10^{-3}$), the standard CROWN-based relaxed methods already

achieve near-perfect verification accuracy ($> 99\%$), rendering the significant computational overhead of alpha-CROWN unnecessary.

These results suggest that for EVENFLOW's real-time constraints, *CROWN + Unrolled Gurobi (Relaxed)* offers the most balanced trade-off, providing high robustness with execution times (approx. 3 minutes) that promises feasibility for scalable deployment.

8 The SCANNV Approach for Parallel Verification

This chapter summarises a Scalable Neural Network Verification (SCANNV) approach, a set of techniques that can reduce the execution time of parallel neural network (NN) verification by optimising (i) how an input property is split into subproblems that can be verified in parallel and (ii) how these subproblems are scheduled for execution.

Most state-of-the-art verifiers expose only a coarse interface. They accept a verification property and internally manage subproblem generation, branching and scheduling. SCANNV treats such a verifier as a black box and adds a thin optimisation layer on top of it. This adds on layer:

- Controls the initial input splitting: it decides how to partition the precondition of the property into a set of subproblems (input polytopes).
- Uses Bayesian Optimisation (BO) to search over possible splitting strategies, using either verification time or internal structural metrics (ReLU stability information) as an optimization objective.
- Optionally performs transfer learning of BO models across related properties on the same neural network, and

As a result, the verifier receives an, expectedly, more "verification-friendly" set of initial subproblems and an execution order that tends to reduce overall wall-clock time.

In our discussion, we henceforth use Venus [REF-25] as the verifier, but the SCANVV framework remains independent of the underlying verification tool.

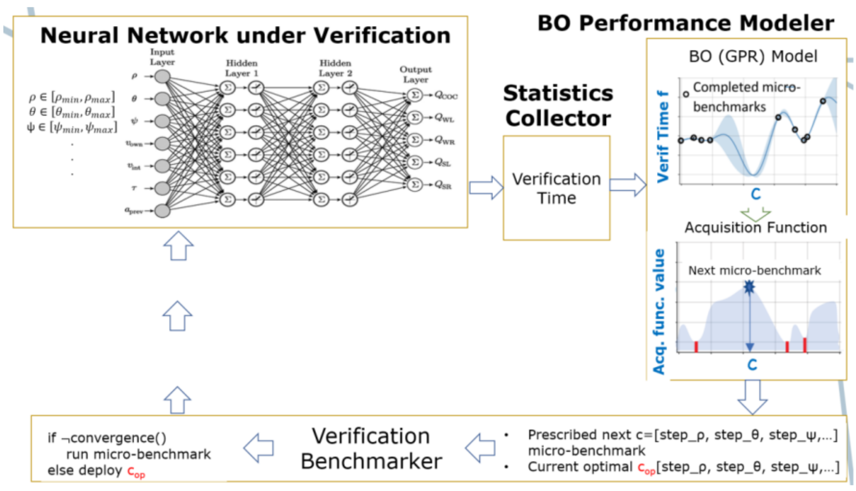


Figure 58: Bayesian Optimization-based Input Splitting Architecture.

The precondition of each property is defined as an axis-aligned box over the input dimensions (e.g., sensor ranges, distances, angles). SCANNV introduces a splitting vector V, where each component V_i specifies into how many equal sub-intervals the i-th input dimension is split.

Given a splitting vector, the Input Splitting Module:

- Splits each input dimension into the required number of **equally sized** sub-intervals.

- Forms the Cartesian product of these sub-intervals across all dimensions.
- Treats each resulting box (polytope) as an initial subproblem and passes the set of subproblems to the verifier.

This procedure preserves the semantics of the original property (the union of all subproblems equals the initial input domain), but exposes a large design space of possible initial splits. Different splitting vectors can lead to very different verification times, even though they are all logically equivalent.

8.1 Input Splitting Black-box Optimization and Transfer Learning

SCANNV models the total verification time corresponding to a given splitting vector as a black-box function. The objective is to find a splitting vector V that minimises this time. To do so, SCANNV applies Bayesian Optimisation with a Gaussian Process (GP) surrogate model, using an RBF kernel and a suitable acquisition function.

For time-based optimisation, it uses a Lower Confidence Bound (LCB) acquisition with a high exploration parameter. This is motivated by the fact that the search space of splitting vectors is discrete and relatively small, but only a limited fraction (e.g., 10%–20%) of candidate splits can be evaluated via micro-benchmarks. Each BO iteration proposes a candidate splitting vector V. SCANNV generates the corresponding set of subproblems, runs a full verification on that set, and measures the overall wall-clock time. The GP is then updated with this pair (split, time), and the acquisition function is maximised again to propose the next split. After a predefined budget of micro-benchmarks is exhausted, the best-performing split observed so far is selected as the optimised initial split. This process is illustrated in Figure 58.

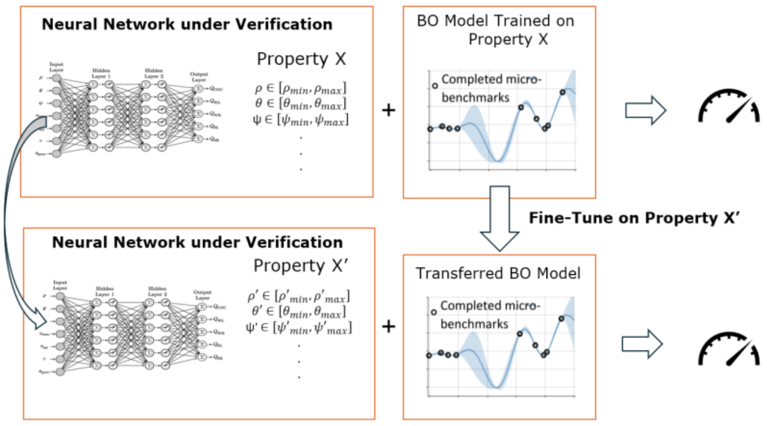

Neural Network under Verification
Property X
$\rho \in [\rho_{min}, \rho_{max}]$
$\theta \in [\theta_{min}, \theta_{max}]$
$\psi \in [\psi_{min}, \psi_{max}]$
BO Model Trained on Property X
Completed micro-benchmarks
Fine-Tune on Property X′
Neural Network under Verification
Property X′
$\rho' \in [\rho'_{min}, \rho'_{max}]$
$\theta' \in [\theta_{min}, \theta_{max}]$
$\psi' \in [\psi'_{min}, \psi'_{max}]$
Transferred BO Model
Completed micro-benchmarks

Figure 59: BO Model Transfer Architecture.

But this approach by itself, does not reduce verification time. This is because we do perform a number of parallel splits and corresponding verification tasks to acquire knowledge about the best-performing split. But, under the same philosophy of RATS+, having acquired knowledge about a property, we can then transfer the GP model to another property and with a minimal number of additional benchmarks we can acquire knowledge about the best-performing input splitting for that new property. If the process continuous for a sufficiently large number of queries/properties, every new property will be served by the developed BO Model, virtually without additional micro-benchmarks.

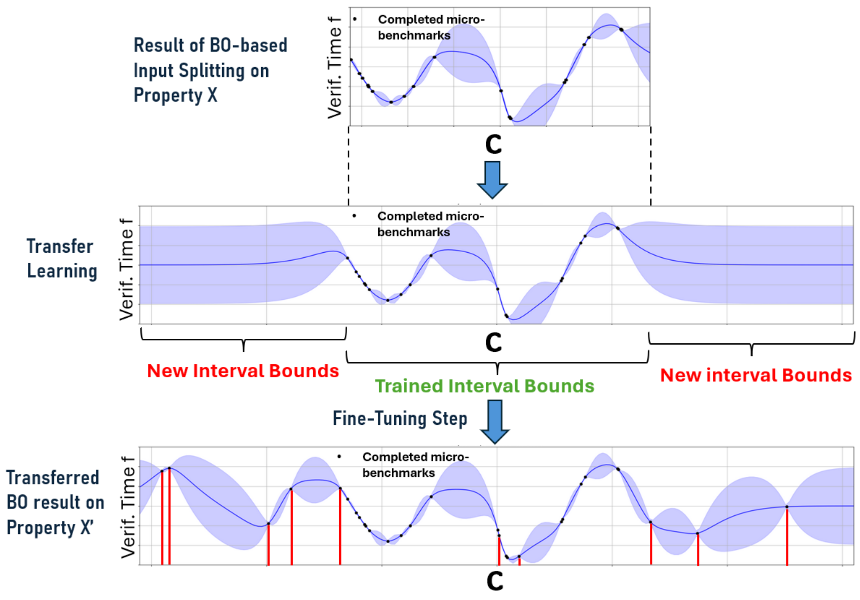

Result of BO-based Input Splitting on Property X
Verif. Time f
Completed micro-benchmarks
C
Transfer Learning
Verif. Time f
Completed micro-benchmarks
New Interval Bounds
C
Trained Interval Bounds
New interval Bounds
Fine-Tuning Step
Transferred BO result on Property X'
Verif. Time f
Completed micro-benchmarks
C

Figure 60: Fine-Tuning of Transferred BO model.

SCANNV therefore extends the above BO process with transfer learning across properties. A BO model trained on a source property is reused as an initial surrogate for a target property, instead of training a new model from scratch (Figure 59). Because the input domains of the two properties may differ, SCANNV performs a fine-tuning phase on the target property with a small number of micro-benchmarks (Figure 60). In this phase, the acquisition function is switched to Expected Improvement (EI) with a large exploration parameter to quickly adapt the surrogate to the new domain. This transfer can substantially reduce the optimisation cost for new properties, especially when the input domains overlap or are subset/superset relations.

8.2 ReLU-Based “Grey-Box” Optimisation

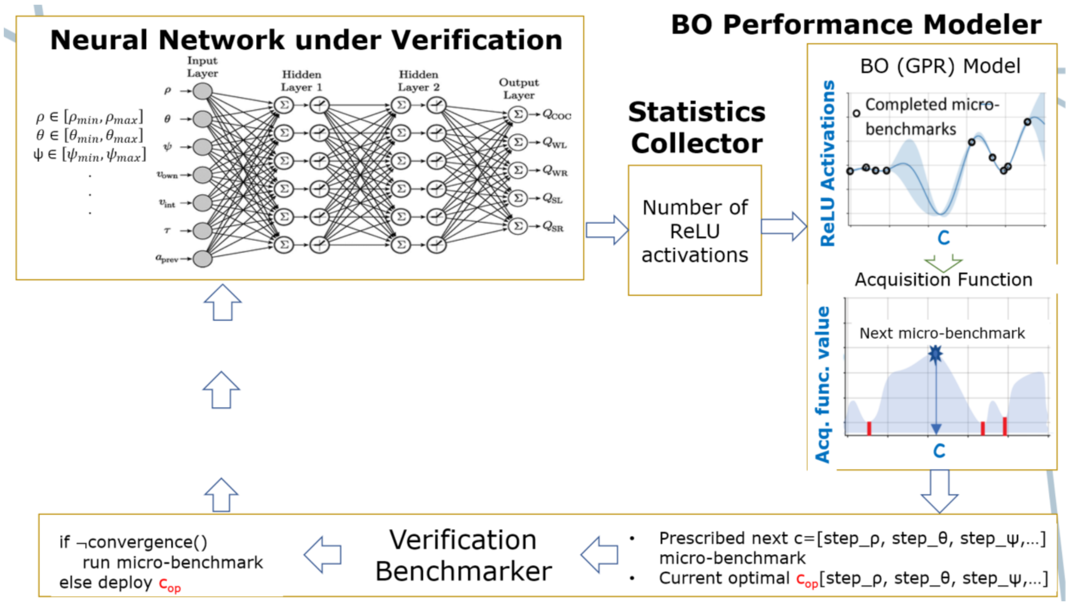


Figure 61: Bayesian Optimization-based ReLU Monitoring Architecture.

Beyond purely black-box optimisation, SCANNV also exploits internal information exposed by the verifier, in particular the stability of ReLU neurons. A neuron is considered stable on a subproblem if its input interval is entirely non-negative or non-positive, i.e., its activation pattern does not depend on the exact input within that subproblem. For each candidate splitting vector, SCANNV computes an average stability ratio across all resulting subproblems, without running full verification, but only inference on V_is. Instead of minimising verification time directly, SCANNV minimises this average stability ratio via BO (using EI as acquisition function). The ReLU-based Grey-Box operation of SCANVV is illustrated in Figure 61. Experimentally, we show that for UNSAT properties, this counter-intuitively leads to lower verification times than the heuristic currently implemented by Venus, which aims to increase stability. The explanation is that less stable initial regions trigger more pruning and faster proof of unsatisfiability in the specific parallel setup.

8.3 SCANVV Experimental Evaluation

SCANVV experiments are conducted on the ACAS Xu benchmark [REF-26], a widely used suite of 45 feed-forward ReLU networks for airborne collision avoidance. Each network has 6 hidden layers and 300 ReLU neurons and replaces a large look-up table-based legacy system. Inputs include relative distances and headings of aircraft, as well as own and intruder speeds. Outputs are discrete advices (e.g., “clear-of-conflict”, “weak left”, “hard right”).

The evaluations focus on a subset of the standard ACAS Xu safety properties, including Properties 2 and 3, resulting in 172 verification queries in total. All experiments use the Venus verifier in a fixed parallel configuration (two splitter processes and four worker processes), with internal heuristics such as the dependency analyser disabled to avoid confounding effects. A maximum of 243 initial subproblems is allowed (e.g., splitting each of 5 input dimensions into up to 3 equal parts), as higher levels of splitting are not consistently manageable within memory and time constraints.

8.3.1 Performance of BO-Based Input Splitting Optimisation

The BO-based input splitting experiments consider two micro-benchmark budgets: (a)10% coverage of the splitting space (e.g., 24 BO evaluations out of 243 possible splits), and (b) 20% coverage (e.g., 48 evaluations). Each BO evaluation runs a full verification of all subproblems generated by the candidate split. Over all properties and networks, the theoretical upper bound of pure verification time is estimated at around three to four months of continuous execution, assuming no timeouts. In practice, due to time limits and early terminations, the effective duration is lower but remains substantial, illustrating the need for sample-efficient BO strategies.

When targeting verification time as objective on property 2 across all associated networks, we compare the BO-selected split against a full grid search over all 243 candidate splits. The quality metric is the rank of the selected by BO split (1 = globally fastest).

With 10% coverage, BO finds a valid split for 22 out of 36 queries. For these:

- All selected splits rank within the top 50;
- 21/22 are within the top 20;
- 18/22 within the top 10; and
- 12/22 within the top 5.

With 20% coverage, BO finds a valid split for 26 out of 36 queries. For these:

- 25/26 are within the top 20;
- 21/26 within the top 10;
- 12/26 within the top 5.

These results show that even with a small fraction of the search space explored, BO identifies near-optimal splits in the majority of cases.

Having acquired good knowledge on a source domain, we proceed to evaluate SCANNV on three transfer scenarios for the BO model:

Subset property (Custom Property 1): whose domain is a subset of Property 2. The transferred model, fine-tuned with a small (e.g., 5%) budget, achieves ranks comparable to or better than those of a newly trained 10% model, and in several networks it even provides strictly better splits.

Table 6: Solution Rank comparison between Transferred Optimizer with 5% domain adaptation vs a 10% Optimizer trained from scratch (Custom Property 1).

ACAS Neural Network ID	Transferred 10% Solution Rank	From scratch 10%
3_6	3	3
4_6	2	3

ACAS Neural Network ID	Transferred 10% Solution Rank	From scratch 10%
5_6	1	1

Superset property (Custom Property 2): whose domain is a superset of Property 2. Again, the transferred + fine-tuning configuration consistently yields splits in the top ranks, often matching or surpassing the from-scratch BO runs.

Table 7: Solution Rank comparison between Transferred Optimizer with 5% domain adaptation vs a 10% Optimizer trained from scratch (Custom property 2).

ACAS Neural Network ID	Transferred 10% Solution Rank	From scratch 10%
3_6	2	6
4_6	1	1
5_6	2	2

Completely Different Property (Property 3): with only partial overlap in input dimensions. Here the domain shift is larger. While the transferred model can perform worse than the newly trained one in some networks, in others it still identifies top-ranked splits.

Table 8: Solution Rank comparison between Transferred Optimizer with 5% domain adaptation vs a 10% Optimizer trained from scratch (Custom property 2)

ACAS Neural Network ID	Transferred 10% Solution Rank	From scratch 10%
3_6	216	5
4_6	2	6
5_6	234	3

Overall, transfer remains beneficial when properties are sufficiently related (subset/superset or substantial overlap). In more distant cases where every dimension of a property is varied, a BO model developed from scratch is preferable.

8.3.2 Impact of ReLU-Based Optimisation and Scheduling

The ReLU-monitoring variant of SCANNV is evaluated by comparing plain Venus against Venus augmented with SCANNV's ReLU-based initial split and schedule.

For UNSAT queries of Property 2, the SCANNV-augmented configuration achieves approximately a 2× reduction in both total and average verification time across the relevant networks. Reducing the total verification time from 619 seconds to 293 seconds

For SAT queries of Property 3, SCANNV achieves equivalent to Venus verification time, although the magnitude of improvement is property dependent.

Overall, ReLU-Based Optimisation and Scheduling seems to favour UNSAT queries, without harming the input splitting performance of SAT ones. Additional validation runs confirm that these improvements are robust and not artifacts of specific random seeds or machine conditions.

Table 9: Comparison between total verification times for standalone Venus, Random Splits + Venus and SCANVV ReLU Monitoring

Venus	**Random Split + Venus**	**SCANVV**
705.86 secs	731.12 secs	319.71 secs

To further validate our claim, we design an experiment where we compare Venus, Random Splits + Venus and SCANVV ReLU-based Optimization, on Property 2. As Table 9 shows, the verification time of standalone Venus is comparable to Venus upon being provided initial random splits. On the contrary, SCANVV reduces the total verification time by more than 2 times.

9 Verification of Spatio-Temporal Systems

Formal robustness guarantees for neural networks have yet to achieve broad practical adoption, primarily due to fundamental scalability limitations. Contemporary verification techniques often fail when applied to high-dimensional input spaces, either because they rely on overly coarse relaxations that weaken the resulting guarantees or because the exact verification procedures become computationally intractable. However, in domains where inputs exhibit inherent spatial or temporal structure—such as consecutive frames in video data or sequential slices in medical imaging—the effective adversarial search space can be substantially reduced. By constraining perturbations to respect realistic spatio-temporal correlations, it becomes possible to narrow the adversary's strength and obtain more meaningful verification outcomes.

In this work, we address the challenge of formally certifying adversarial robustness under such structured perturbation regimes. We introduce **spatio-temporal bound propagation (STBP)**, a new computational technique that begins by solving a mixed-integer linear program (MILP) over the network's initial layer and then propagates the resulting tight bounds through the subsequent layers using a combination of exact and relaxed reasoning. This hybrid design enables the method to capture fine-grained constraints in the early stages of computation while maintaining tractability throughout the network.

Our experiments demonstrate that STBP delivers stronger certified guarantees than standard adversarial training and offers practical advantages over certified training approaches. Notably, STBP achieves more than a 1.7× improvement in robust accuracy for equivalent perturbation budgets, highlighting its potential to make formally verified robustness more achievable in realistic, temporally structured settings.

Despite the clear value of formal certification in safety-critical machine learning, certified robustness techniques have seen limited practical adoption. This gap arises primarily because existing verification methods either (a) rely on coarse approximations that produce guarantees too loose to be actionable, or (b) require computational resources far beyond what is feasible for large-scale models. These issues become even more pronounced as input dimensionality increases. Approximate abstract-interpretation methods, such as interval bound propagation, tend to be overly conservative in high-dimensional spaces and thus fail to produce meaningful robustness guarantees [REF-39]. Conversely, exact verification approaches, such as mixed-integer linear programming (MILP), scale poorly and become prohibitively expensive for modern architectures [[REF-40], [REF-41], [REF-42]]. As a result, current verification efforts are largely restricted to low-dimensional datasets and relatively small neural models [[REF-43], [REF-32]].

To address these limitations, we propose a computational framework that leverages domain knowledge to impose realistic spatio-temporal constraints on input perturbations. Such constraints are highly relevant in domains involving video data [[REF-44], [REF-45]] or volumetric medical imaging [REF-46], where common verification approaches implicitly assume that an adversary is capable of independently perturbing video frames occurring milliseconds apart. In practice, adversarial manipulations—such as placing misleading traffic signs or altering visual artifacts—tend to be externally induced and therefore temporally correlated. Likewise, sensor-induced noise exhibits well-documented spatial correlation patterns [[REF-46], [REF-47], [REF-48]]. Motivated by these observations, we introduce **spatio-temporal shared interval bound propagation (S-IBP)**, a hybrid verification technique that combines the precision of MILP with the scalability of abstract interpretation. Our method first uses MILP to compute exact, tightly bounded perturbations at the network's first layer under the given spatio-temporal constraints and then propagates these bounds through the remaining layers using abstract interpretation. Furthermore, by exploiting the differentiable components of the bound-propagation process, we formulate new training objectives that enable learning neural networks that are *provably robust* to structured spatio-temporal perturbations.

We conduct extensive experiments validating the effectiveness of our approach across diverse datasets, including UCF-101 for action recognition [REF-49], the Udacity self-driving car dataset [REF-51], and several medical imaging benchmarks [REF-50]. To evaluate robustness in a manner grounded in real-world behaviour, we construct novel spatio-temporal verification benchmarks. For autonomous driving, we design perturbation scenarios that identify plausible regions in each frame where adversarial elements—such as spoofed road signs or artificial bumper stickers—could be introduced. For medical imaging, guided by known sensitivity patterns in MRI acquisition, we develop procedures that generate spatio-temporal perturbation constraints aligned with realistic acquisition errors.

Our results demonstrate that, under these structured robustness specifications, S-IBP enables the training of neural networks with **certified robustness an order of magnitude higher** than that provided by standard adversarial training. Moreover, when compared with existing certification-based training approaches calibrated to the same robustness thresholds, our method achieves **over 30% higher accuracy on clean data**, indicating substantial practical utility. Beyond these performance gains, we also conduct a systematic analysis of widely used spatio-temporal neural architectures, characterizing the extent to which they can be made provably robust to realistic perturbation patterns.

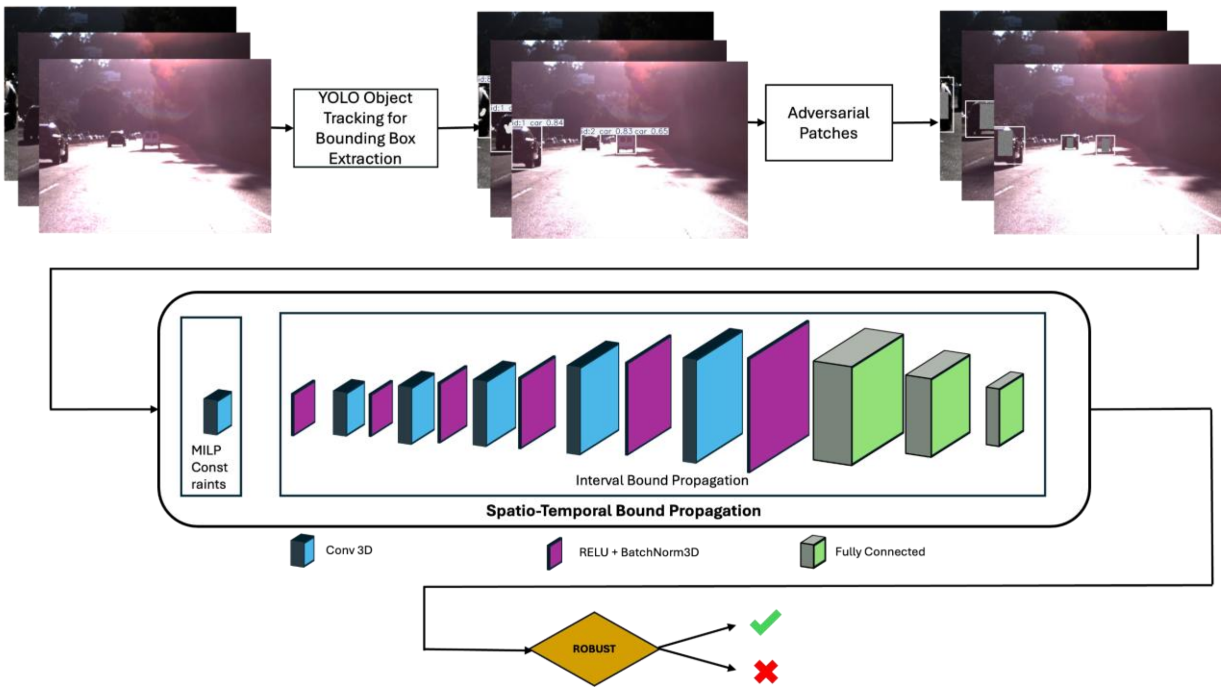

YOLO Object Tracking for Bounding Box Extraction
Adversarial Patches
MILP Constraints
Interval Bound Propagation
Spatio-Temporal Bound Propagation
Conv 3D
RELU + BatchNorm3D
Fully Connected
ROBUST

Figure 62: Overview of Spatio-Temporal Bound propagation.

Figure 62 illustrates the overall architecture of the Spatio-Temporal Bound Propagation (STBP) framework. In this setup, adversarial patches are generated using a YOLO-based pipeline, which provides realistic, structured perturbations aligned with the spatial and temporal characteristics of the input domain. STBP achieves tight robustness guarantees by applying linear programming to the network's first layer, enabling the computation of highly precise bounds at the point where perturbations first enter the model. For the subsequent layers, where exact methods would be computationally prohibitive, the framework transitions to more efficient relaxation techniques—specifically Interval Bound Propagation (IBP)—to propagate the bounds forward through the network. This hybrid strategy balances tightness and scalability, ensuring that STBP captures fine-grained adversarial constraints without incurring the full cost of exact verification across the entire architecture.

9.1 Spatio-Temporal Bound Propagation Method

We consider a supervised learning setting in which a model

$$f: \mathbb{R}^{C\times H\times L} \rightarrow \mathcal{Y}$$

maps an input tensor $\mathbf{x} \in \mathbb{R}^{C\times H\times L}$to an output in a label space $\mathcal{Y}$. Here, Cdenotes the number of channels, $H$the spatial resolution, and $L$the temporal or sequential dimension, such as video frames or stacked imaging slices. The model $f$is typically instantiated as a deep neural network composed of alternating affine transformations and nonlinear activations. We assume a data distribution $\mathcal{D}$over input–label pairs $(\mathbf{x}, y) \in \mathbb{R}^{C\times H\times L} \times \mathcal{Y}$, and our goal is to verify not only the model's behaviour at individual data points, but its behaviour over entire neighbourhoods defined by allowable perturbations.

To formalize this, we define an input specification $T(\mathbf{x}) \subseteq \mathbb{R}^{C\times H\times L}$capturing the set of perturbed inputs considered semantically equivalent to a given $\mathbf{x}$. Robustness verification aims to show that the model's predicted label remains invariant across this set. Specifically, we seek to prove that $f(\mathbf{x}') = f(\mathbf{x})$for all $\mathbf{x}' \in T(\mathbf{x})$.

A certificate of this form constitutes a formal guarantee that the network is robust to all perturbations permitted by the specification. However, verifying such guarantees becomes increasingly challenging in high-dimensional spatio-temporal domains, where existing methods struggle to propagate tight bounds through deep architectures. This motivates the development of new verification techniques that can exploit structured spatial and temporal dependencies in the input.

9.1.1 Modelling Spatio-Temporal Constraints

We next introduce a mixed-integer linear programming (MILP) formulation to compute tight, certified bounds on the activations of the first layer under structured perturbations. Let

$$\mathbf{x} \in \mathbb{R}^{C \times D \times H \times W}$$

be a four-dimensional input tensor and let

$$T(\mathbf{x}) \subseteq \mathbb{R}^{C \times D \times H \times W}$$

denote the set of admissible perturbed inputs constrained to satisfy realistic spatio-temporal coherence. Let $\boldsymbol{\delta} \in \mathbb{R}^n$be the flattened perturbation tensor, with $n = C * D * H * W$.

The perturbed input is $\mathbf{x} + \boldsymbol{\delta}$. We impose structured constraints on $\boldsymbol{\delta}$via the following sets:

- **Bounded perturbations:** for $i \in \mathcal{B} \subseteq \{1, \dots, n\}$,

$$-\epsilon_i \leq \delta_i \leq \epsilon_i.$$

- **Shared perturbations:** for all index pairs $(i, j) \in \mathcal{S}$,

$$\delta_j = \delta_i.$$

- **Fixed (non-perturbed) entries:** for $i \in \mathcal{F} \subseteq \{1, \dots, n\}$,

$$\delta_i = 0.$$

These constraints encode known structural priors such as temporal smoothness or anatomical consistency across slices, thereby reducing adversarial degrees of freedom.

9.1.1.1 Affine layer

Let the network's first layer be affine, with weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$and bias $\mathbf{b} \in \mathbb{R}^m$. The pre-activation response is

$$\mathbf{z} = \mathbf{W}(\mathbf{x} + \boldsymbol{\delta}) + \mathbf{b} \in \mathbb{R}^m.$$

To obtain tight bounds on $z_j$over all valid perturbations, we solve two MILPs per output coordinate $j \in \{1, \dots, m\}$:

9.1.1.2 MILP Formulation

$$\begin{aligned} \max_{\boldsymbol{\delta}} / \min_{\boldsymbol{\delta}} \quad & z_j = \mathbf{w}_j^\top(\mathbf{x} + \boldsymbol{\delta}) + b_j \\ \text{subject to} \quad & \delta_i \in [-\epsilon_i, \epsilon_i] \forall i \in \mathcal{B}, \\ & \delta_i = \delta_k \forall (i,k) \in \mathcal{S}, \\ & \delta_i = 0 \forall i \in \mathcal{F}. \end{aligned}$$

Although each problem is a linear program with bound and equality constraints, the shared perturbation sets introduce nontrivial coupling among variables. Nonetheless, the resulting optimal values give certified, input-dependent intervals for the activations of the first layer, which form the foundation of the hybrid verification pipeline.

9.1.2 Spatio-Temporal Bound Propagation (STBP)

Given the structured input specification $T(\mathbf{x}) \subseteq \mathbb{R}^n$, our goal is to compute sound output bounds for the neural network

$$f = f^{(k)} \circ \cdots \circ f^{(1)}$$

under all admissible perturbations. To accomplish this, we propose **Spatio-Temporal Bound Propagation (STBP)**, a hybrid verification algorithm combining exact MILP-based bounds at the input layer with efficient relaxation-based propagation for deeper layers.

For each neuron $j$in the first layer, we compute exact bounds

$$\ell_j^{(1)} := \min_{\mathbf{x}' \in T(\mathbf{x})} z_j^{(1)}(\mathbf{x}'), u_j^{(1)} := \max_{\mathbf{x}' \in T(\mathbf{x})} z_j^{(1)}(\mathbf{x}'),$$

where $z_j^{(1)}(\mathbf{x}') = \mathbf{w}_j^\top \mathbf{x}' + b_j$. These MILP-certified bounds provide a tight enclosure of the first-layer activations.

For subsequent layers ($i > 1$), we propagate bounds using interval bound propagation (IBP) or similar linear relaxations. Given bounds $\ell^{(i-1)}$and $u^{(i-1)}$, define the center and radius:

$$\mathbf{c}^{(i-1)} = \frac{1}{2}\left(\ell^{(i-1)} + u^{(i-1)}\right), \mathbf{r}^{(i-1)} = \frac{1}{2}(u^{(i-1)} - \ell^{(i-1)}).$$

For an affine transformation with weights $\mathbf{W}$ and bias $\mathbf{b}$:

$$\ell^{(i)} = \mathbf{W}\mathbf{c}^{(i-1)} - |\mathbf{W}| \mathbf{r}^{(i-1)} + \mathbf{b}, \ u^{(i)} = \mathbf{W}\mathbf{c}^{(i-1)} + |\mathbf{W}| \mathbf{r}^{(i-1)} + \mathbf{b},$$

where the absolute value is elementwise.

The key innovation of STBP lies in combining **exact layer-1 MILP bounds**, which encode detailed spatio-temporal structure, with **fast, conservative relaxation methods** for deeper layers. This results in a verification approach that is both scalable and significantly tighter than standard IBP, particularly in domains where spatio-temporal correlations substantially reduce the effective perturbation dimension.

9.2 Datasets and Models

We evaluate the proposed verification methods across four heterogeneous datasets encompassing synthetic, real-world, and medical imaging domains. For **MNIST**, we construct a video-style variant by temporally concatenating individual digit frames, enabling the evaluation of two 3D CNN architectures operating at different spatial resolutions; these models achieve classification accuracies of up to 94.67%. For **UCF-101**, a standard video action-recognition benchmark, we reduce both the spatial resolution and the number of action classes to improve the tractability of verification. Under this simplified setting—restricted to five broad action categories—the resulting model attains a clean accuracy of 74.41%.

For the **Udacity self-driving dataset**, we convert the original steering-angle regression task into a three-way classification problem. To facilitate verification, we downsample the input images and employ a compact CNN, which achieves 84.24% accuracy. Finally, for **MEDMNIST**, we focus on the Synapse-3D subset, where the task is to classify neuron types from 3D voxel representations. Using a model architecturally aligned with the MNIST baseline, we obtain an accuracy of 73.01%.

Across all experiments, the neural networks are deliberately designed to be compact—each containing no more than ten layers—to ensure that spatio-temporal verification remains computationally feasible. These models nonetheless capture the essential complexity of their respective domains, enabling a meaningful evaluation of the scalability and effectiveness of our verification techniques.

9.3 Experiments

We empirically assess the effectiveness of the proposed approach in certifying the adversarial robustness of neural networks across a diverse set of spatio-temporal tasks. Our evaluation demonstrates consistent and significant improvements in certified robustness over standard verification techniques for every dataset and model considered. In particular, we study robustness under perturbation magnitudes $\epsilon \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$and adversarial patch sizes $k \in \{1, \dots, 10\}$, comparing standard Interval Bound Propagation (IBP) against our hybrid Spatio-Temporal Bound Propagation (STBP) method.

Figure 63 presents the certified robust accuracy of the MNIST video model—constructed using 10 frames at 28×28resolution—under increasing perturbation magnitudes (left) and patch sizes (right). Across all settings, STBP consistently outperforms IBP, with particularly pronounced gains under structured perturbations. The patch-based version of STBP achieves near-perfect robustness for perturbations up to $\epsilon = 10^{-4}$and for small patch sizes, underscoring the advantage of integrating domain-specific spatial and temporal structure into the verification pipeline.

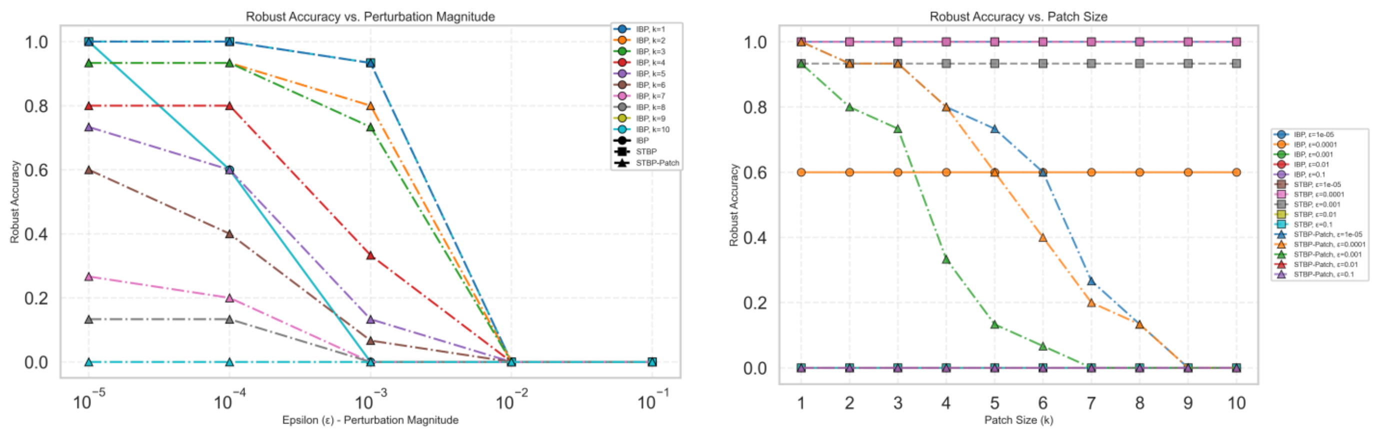


Figure 63: Adversarial Robustness of IBP, STBP, STBP using adversarial patches for MNIST 10 frame 28 x 28 video model; left: against perturbations (ϵ); right: against patch size (k).

A consolidated summary of results is provided in Table 10, reporting both clean and certified robust accuracies across the four benchmark domains: MNIST, UCF-101, Udacity self-driving, and MEDMNIST Synapse3D. STBP yields substantial improvements compared to IBP—for example, on the MNIST video model, STBP with patch constraints increases certified robust accuracy from 49.32% to 77.05% at $\epsilon = 10^{-4}$. For the autonomous driving and medical imaging tasks, STBP achieves over 85% certified robustness under patch perturbations, demonstrating that the method scales effectively to more complex real-world systems.

Although STBP scales well for moderate-sized 3D CNNs, verifying larger architectures—such as ResNet-style backbones—remains challenging even when using shared perturbation models. As expected, the inclusion of a MILP optimization step on the first layer introduces additional computational overhead, but this cost is offset by the substantial improvements in certification quality.

Table 10: Summary of results for MNIST, UCF-101, Udacity self-driving, and MEDMNIST Synapse3D.

Experiment	**Clean Acc.**	**Input Dim**	**Output Dim**	**# Samples**
MNIST Toy Model	93.1%	1×5×8×8	10	10000
MNIST Toy Model	94.67%	1×10×28×28	10	1000
UCF-101	74.41%	3×30×32×32	5	7
Udacity Steering Angle	84.23%	3×30×32×32	3	15
MEDMNIST Synapse3D	73.01%	32×32×32	2	100

Experiment	**IBP Acc.**					**STBP Acc.**				
	ε=0.1	ε=0.01	ε=10^{-3}	ε=10^{-4}	ε=10^{-5}	ε=0.1	ε=0.01	ε=10^{-3}	ε=10^{-4}	ε=10^{-5}
MNIST Toy Model	0.0	0.0	1.2	53.2	89.1	0.0	1.15	**42.0**	**91.2**	**93.10**
MNIST Toy Model	0.0	0.0	0.0	49.32	93.75	0.0	0.0	**58.20**	**77.05**	78.71
UCF-101	0.0	0.0	0.0	23.03	26.36	0.0	**7.3**	**19.36**	**21.9**	**53.81**
Udacity Steering Angle	0.0	0.0	0.0	85.71	85.71	**85.71**	**85.71**	**85.71**	85.71	85.71
MEDMNIST Synapse3D	0.0	0.0	19.0	72.0	72.0	0.0	**72**	**72**	**72**	**72**

10 Probabilistic Verification of Neural Networks via PAC-Interval Estimation

10.1 The Verification Criterion

PAC Interval Estimation defines verification in the context of the **local Lipschitz constant**. Let $B_\epsilon(x_0) = \{x : \| x - x_0 \| \leq \epsilon\}$ be the input region of interest (an ϵ-ball).

Let $M(x_0)$ be the margin (distance) from the output $f(x_0)$ to the nearest decision boundary. A sufficient condition for robustness is given by:

$$L^* \cdot \epsilon < M(x_0)$$

where $L^* = \max_{x \in B_\epsilon(x_0)} \| J_f(x) \|$ is the maximum operator norm of the Jacobian (the local Lipschitz constant) within the ball. If this inequality holds, it is geometrically impossible for the network to misclassify an input within the region.

10.1.1 The Estimation Challenge

Calculating L^* exactly is an NP-hard problem for general neural networks. The optimization landscape of $\| J_f(x) \|$ is highly non-convex, filled with local maxima and sharp peaks.

- **Analytical Bound Propagation (e.g., AutoLIRPA):** These methods propagate error intervals layer-by-layer through the network. While computationally fast, they often suffer from the dependency problem (the "wrapping effect"), resulting in loose upper bounds that are orders of magnitude larger than the true constant. This often leads to a failure to satisfy Eq. (1) even for robust networks.
- **Exact Methods (e.g., LipMIP):** Mixed Integer Programming (MIP) can theoretically find the true global maximum. However, these solvers scale poorly with network depth and width, often timing out before finding a solution.

To overcome these limitations, we relax the requirement for a deterministic maximum in favour of a probabilistic bound that is tight enough to be useful but rigorous enough to provide safety guarantees.

10.2 Theoretical Framework: The PAC Interval

We frame the verification task as a statistical estimation problem. Instead of a single point estimate, we seek a **Probably Approximately Correct (PAC) Interval** $[L_{low}, L_{high}]$ for the true maximum L^*.

10.2.1 Understanding the PAC Framework

The concept of Probably Approximately Correct (PAC) learning originates from computational learning theory. In its standard form, it addresses the question: "How many samples do we need to learn a hypothesis that is accurate with high probability?"

In the context of Lipschitz estimation, we adapt this framework to quantify uncertainty. Since we cannot evaluate the gradient at every single point in the continuous domain $B_\epsilon(x_0)$ (which would require infinite time), we must rely on a finite set of samples. Consequently, there is always a non-zero probability that the true maximum lies in an unobserved region.

The PAC framework allows us to formalize this risk. We seek an upper bound L_{high} such that the probability of the true maximum exceeding this bound is bounded by a small parameter γ. In simple terms, we are making a statistical contract:

> *"We cannot guarantee with 100% certainty that L^* is below L_{high}. However, we have constructed L_{high} such that the chance of us being wrong is less than γ (e.g., 0.1% or 0.001%)."*

Mathematically, our verification guarantee is probabilistic:

$$\Pr(L_{low} \leq L^* \leq L_{high}) \geq 1 - \gamma$$

If $L_{high} \cdot \epsilon < M(x_0)$, the network is verified with confidence $1 - \gamma$. This approach trades the intractable certainty of deterministic methods for the tractable, high-confidence guarantees of statistical methods.

10.2.2 The Cumulative Distribution Function (CDF)

To implement the PAC framework, we analyse the statistical behaviour of the gradient norms. Let X be a random variable representing the input chosen by our sampling strategy and let $Y = g(X) = \| J_f(X) \|$ be the gradient norm at that input.

The central object of our study is the **Cumulative Distribution Function (CDF)**, denoted as $F(y)$. The CDF describes the probability that a random sample will have a gradient norm less than or equal to a value y:

$$F(y) = \Pr(Y \leq y)$$

For example, if $F(50) = 0.95$, it means that 95% of the sampled gradients are less than or equal to 50. The value of y where $F(y) = 1$ is the theoretical maximum L^*. Our goal is to estimate the behavior of $F(y)$ as it approaches 1.

Since we do not know the true CDF $F(y)$, we approximate it using the **Empirical CDF**, $\hat{F}_N(y)$, constructed from N observed samples. The empirical CDF is a step function that jumps by $1/N$ at each observed data point. As $N \to \infty$, the Law of Large Numbers guarantees that $\hat{F}_N(y)$ converges to $F(y)$.

10.2.3 The Dvoretzky-Kiefer-Wolfowitz (DKW) Inequality

While $\hat{F}_N(y)$ converges to the truth, for finite N there is an error. We need to bound this error to maintain our safety guarantee. We utilize the **Dvoretzky-Kiefer-Wolfowitz (DKW) inequality**, a powerful theorem in non-parametric statistics.

10.2.3.1 Concept: The Confidence Tube

The DKW inequality allows us to build a "confidence tube" (or confidence band) around our empirical data. Imagine plotting the empirical CDF step function on a graph. The DKW inequality draws two boundary curves—one above and one below—creating a corridor.

The theorem guarantees that the invisible, true CDF $F(y)$ lies entirely within this corridor with probability at least $1 - \gamma$. It provides a strict, global bound on the distance between the observed data and the ground truth.

10.2.3.2 Mathematical Formulation

Given N independent and identically distributed (i.i.d.) samples, the DKW inequality states:

$$\Pr\left(\sup_y |\hat{F}_N(y) - F(y)| > \alpha\right) \leq 2e^{-2N\alpha^2}$$

By setting the right side to our desired confidence level γ and solving for the bandwidth α, we get:

$$\alpha = \sqrt{\frac{\ln(2/\gamma)}{2N}}$$

We then define the lower and upper bounds of the confidence tube as:

$$F_L(y) = \max(0, \hat{F}_N(y) - \alpha)$$
$$F_U(y) = \min(1, \hat{F}_N(y) + \alpha)$$

This tube serves as the primary constraint for our verification. Any statistical model we propose for the tail of the distribution must effectively "live" within this tube to be considered a plausible explanation of the data.

10.2.4 Parameter Space Search for the Worst-Case Model

A simple statistical approach would be to fit a single "best-fit" curve to the tail data and report its endpoint. However, there might be many slightly different curves that fit the data almost equally well but imply very different maximum values.

To adhere to the safety-critical nature of verification, we employ a **robust optimization procedure**:

1. We define a family of curves (Generalized Pareto Distributions) that are theoretically justified to model the tail.
2. We identify the subset of these curves that are "statistically plausible." A curve is plausible if and only if it lies entirely within the DKW confidence tube ($F_L \leq G \leq F_U$).
3. From this plausible set, we find the single curve that is the most pessimistic—i.e., the one with the largest upper endpoint.

This yields L_{high}, the worst-case Lipschitz constant consistent with our data and our confidence level γ. A practical illustration is shown in Figure 64.

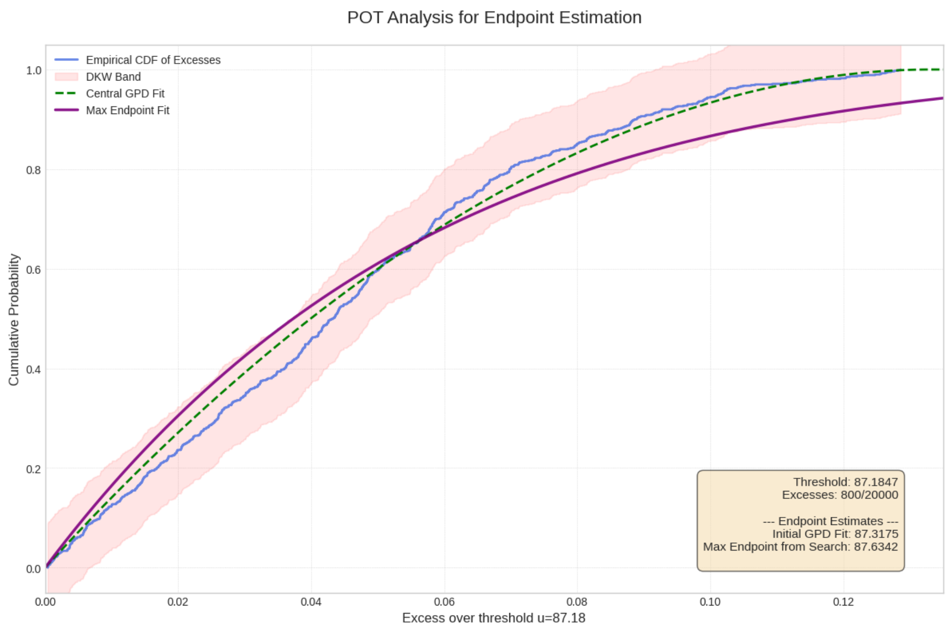


Figure 64: The observed exceedances (blue line), fitted GPD (green line) and maximum plausible fit (purple line). The goal of the algorithm is to find the GPD fit with the largest endpoint that fits entirely in the confidence tube.

10.3 Targeted Sampling: The Adam-Sobol Method

The mathematical guarantees of the PAC framework (specifically the DKW inequality) rely on the assumption that the samples are representative of the underlying distribution. In the context of neural network verification, this is a significant hurdle.

The input space $B_\epsilon(x_0)$ is a high-dimensional hyperball. As the dimension d increases, the volume of the space explodes exponentially (the Curse of Dimensionality). The regions where the gradient norm is maximized—the "peaks" of the landscape—are typically extremely small and sparse relative to the total volume.

If we were to use simple uniform random sampling, the probability of landing near a peak would be vanishingly small. We would mostly observe gradients from the "flat" low-value regions. Consequently, our DKW tube would accurately model the body of the distribution but would contain no information about the tail, leading to a gross underestimation of L^*.

To solve this, we require a sampling strategy that is **biased** towards high values. We developed a specialized, multi-stage strategy named **Adam-Sobol**.

10.3.1 Modularity of the Sampling Engine

It is crucial to note that **this specific sampling method is not a fixed constraint** of the proposed framework. The core contribution is the PAC-Interval analysis (Chapter 2), which acts as a statistical backend agnostic to the data source. We employ Adam-Sobol here because it performs well empirically. However, the framework is modular: if a superior sampling strategy—such as one based on generative models, flow-matching, or more advanced adversarial attacks—can populate the distribution tail more densely, it can replace Adam-Sobol without altering the statistical core of the verification.

10.3.2 Stage 1: Global Peak Discovery (Adam Attack)

The first challenge is to locate the general "basins of attraction" where the gradient norms are high. To do this, we treat the sampling problem as an optimization problem. We want to find input vectors x that maximize the scalar function $g(x) = \| J_f(x) \|$.

We employ an adversarial attack strategy based on the **Adam optimizer**. Adam is a gradient-based optimization algorithm widely used in training neural networks. Here, instead of updating network weights to minimize loss, we update the input vector x to maximize the gradient norm.

- **Initialization:** We define W parallel particles, each initialized randomly within the input ball $B_\epsilon(x_0)$.
- **Ascent:** For each particle, we calculate the gradient of our objective function with respect to the input, $\nabla_x g(x)$. We then move the particle in the direction of this gradient.
- **Restarts:** Optimizers can get stuck in local optima (smaller peaks). By running many restarts in parallel, we increase the coverage of the search space.

This stage results in a set of samples S_{Adam} clustered around the local maxima of the function.

10.3.3 Stage 2: Local Distributional Refinement (K-Box Sobol)

While the Adam attack is excellent at finding single high points, it is not a sampler in the statistical sense; it creates a biased cluster of points at the very tips of the peaks. To perform Extreme Value Theory analysis, we need to understand the *shape* of the peak—how the values fall off as we move away from the maximum. We need to sample the "exceedances" (values above a threshold).

To achieve this, we refine the output of the Adam stage:

1. **Selection:** We take the top K distinct candidates from S_{Adam}. These represent the K most promising regions found.

2. **Bounding (The K-Box):** Around each candidate peak x_k, we define a small hypercube (a "box") B_k constrained within the original ϵ-ball.

3. **Sobol Sampling:** Within each of these small boxes, we generate a high-density set of samples. Crucially, we do not use pseudo-random numbers. Instead, we use **Sobol sequences**.

10.3.3.1 Why Sobol Sequences?

Sobol sequences are a type of Quasi-Monte Carlo (QMC) method. Unlike pseudo-random numbers, which are independent and can randomly clump together leaving gaps in the space, Sobol sequences are deterministic and designed to be "low discrepancy." They are self-avoiding, meaning they spread out as evenly as possible to fill the available space.

By using Sobol sequences within the K-Boxes, we obtain a high-fidelity, uniform scan of the geometry around the peaks. This provides the rich distributional data—the shape of the tail—necessary for rigorous GPD fitting. The final dataset S_{final} is the union of these local scans.

It is important to note that algorithms for generating Sobol sequences are well defined only up to a certain (albeit very large) dimension. In problems where the size of the input dimension is larger than 21201, we have to employ Latin Hypercube Sampling, which is a different method to generate samples that are "not too clustered".

10.4 Estimation Engine: Peaks-over-Threshold (POT)

The theoretical foundation of the POT method is the **Pickands-Balkema-de Haan theorem**. This theorem is to extreme values what the Central Limit Theorem is to averages.

The Central Limit Theorem states that if you sum up many random variables, the result converges to a Gaussian (Normal) distribution, regardless of the original distribution. Similarly, the Pickands-Balkema-de Haan theorem states that if you look at the distribution of values that exceed a sufficiently high threshold u, that conditional distribution converges to the **Generalized Pareto Distribution (GPD)**.

The CDF of the GPD is given by:

$$G_{\xi,\sigma}(y) = 1 - \left(1 + \frac{\xi y}{\sigma}\right)^{-1/\xi}$$

where σ is a scale parameter and ξ is the shape parameter.

10.4.1.1 The Shape Parameter and the Reverse Weibull

The shape parameter ξ determines the behavior of the tail. There are three regimes:

1. **Fréchet ($\xi > 0$):** Heavy-tailed distributions (e.g., wealth distribution). These have no upper limit and decay polynomially.
2. **Gumbel ($\xi = 0$):** Light-tailed distributions (e.g., normal distribution). These have no upper limit but decay exponentially.
3. **Reverse Weibull ($\xi < 0$):** Short-tailed distributions. These have a mathematically **finite upper endpoint**.

For neural network verification, we strictly restrict our search to the **Reverse Weibull** domain ($\xi < 0$). This is not an arbitrary choice but a physical constraint. A neural network with finite weights, operating on a compact input domain $B_\epsilon(x_0)$, defines a continuous function. By the Extreme Value Theorem of calculus, a continuous function on a compact set is bounded. Therefore, the distribution of gradients *must* have a finite endpoint.

Our goal is to estimate this finite endpoint, given by:

$$y_{max} = u + \frac{\sigma}{|\xi|}$$

This y_{max} is our estimator for L_{high}.

10.4.2 Automated Threshold Selection: Balancing Bias and Variance

The application of POT requires selecting a threshold u. This choice is subtle and represents a fundamental **Bias-Variance Trade-off**.

- **Threshold Too Low (High Bias):** If we set u too low, we include data points from the "body" of the distribution. These points do not follow the GPD (which is an asymptotic limit for extremes). Including them violates the assumptions of the theorem, biasing our estimates of ξ and σ.
- **Threshold Too High (High Variance):** If we set u extremely high, we satisfy the theorem’s assumptions perfectly, but we are left with very few data points (only the very tip of the tail). With small sample sizes, statistical estimation becomes unstable, leading to massive variance and wide confidence intervals.

We automate this selection using the **Mean Residual Life Plot (MRLP)**. The "Mean Residual Life" is the expected amount by which a value will exceed the threshold, given that it exceeds it.

$$e(u) = E[Y - u | Y > u]$$

A unique property of the GPD is that the mean residual life is a **linear function** of the threshold u.

Our Algorithm:

1. We compute the empirical mean residual life for a range of candidate thresholds.
2. We fit a Weighted Least Squares (WLS) line to this plot.
3. We perform a sequential hypothesis test (using χ^2 statistics) to find the lowest threshold u_{opt} where the linearity hypothesis cannot be rejected.

This identifies the "sweet spot": the lowest threshold (maximizing sample size) where the distribution is sufficiently "extreme" (minimizing bias). An illustrative example using synthetically generated data is shown in Figure 65, where the algorithmically selected threshold is very close to the true threshold of the synthetic data.

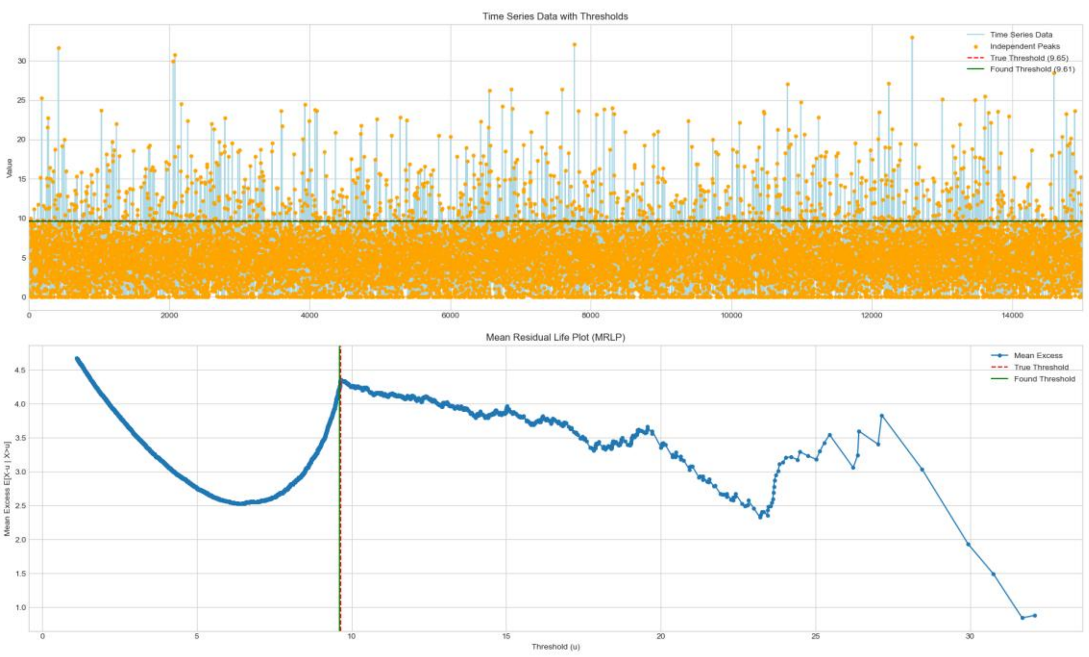


Figure 65: Top: Illustration of synthetic data lying above the detected and true threshold. Bottom: Mean Residual Life (MRL) plot for synthetic data; the algorithm seeks to find the smallest value at which the MRL begins to showcase a linear behaviour. This synthetic data is composed of random uniform sampling and a GPD distribution imposed on the top range of data, hence there is a true threshold which we use as comparison.

10.4.3 Constraint-Based Verification Bound

Once u_{opt} is determined, we construct the final verification bound. As discussed in the PAC section, we do not simply take the maximum likelihood parameters. We search the (ξ, σ) space for the GPD that maximizes y_{max} while remaining inside the DKW confidence tube.

$$L_{high} = \max_{\xi,\sigma} \left(u_{opt} + \frac{\sigma}{|\xi|} \right) \quad \text{s.t.} \quad F_L(y) \leq G_{\xi,\sigma}(y) \leq F_U(y)$$

10.5 Experimental Results

We validated the Adam-Sobol + LipPOT pipeline on the SMALLMNIST5 network. This network, while small enough to be analysed by exact solvers, presents a sufficiently complex landscape to test the efficacy of our statistical approach.

10.5.1 Performance on Small Domain

In this regime, the domain is small enough that the Mixed Integer Programming solver (LipMIP) can converge to the exact global maximum within a reasonable timeframe. This provides a "Ground Truth" against which we can evaluate our method.

Table 11: Comparison on SMALLMNIST5 (Local 0.01 Domain, L_∞ Norm)

Algorithm	Upper Bound	Lower Bound	Time (s)	Outcome
LipPOT (Ours)	**80.92**	80.06	94.18	Tight Interval
auto_LIRPA	1308.26	NaN	0.67	Loose
LipMIP	84.30	NaN	704.18	**Exact Global Max**

10.5.1.1 Analysis of Underestimation

Table 11 reveals a critical insight into the nature of our method. LipMIP finds the exact maximum to be 84.30. Our method, LipPOT, reports a high-confidence upper bound of 80.92.

Crucially, *LipPOT underestimates the true constant*. The upper bound (80.92) is slightly lower than the true maximum (84.30). This discrepancy (approx. 4%) is the "smoking gun" of statistical verification. It indicates that despite our specialized Adam-Sobol sampling, the very highest peak in the landscape was extremely narrow and was missed by the sampler. The EVT model correctly fitted the tail of the *observed* peaks, but it cannot infer the existence of a singular, unobserved spike that behaves differently from the rest of the tail. This confirms that our method provides a probabilistic certificate, not a deterministic guarantee.

In this particular instance, the reason for the underestimation was because the network had a discontinuity contained within the input region. However, compared to the analytical method (auto_LIRPA), which overestimated the bound by over 1500% (1308.26), LipPOT provides a far more distinct and useful signal of the network’s behaviour, albeit with a known statistical risk.

10.5.2 Performance on Large Domain

We tested the largest domain from our experimental set ($\epsilon = 0.13$) to demonstrate robustness under extreme expansion. In this regime, the search space is exponentially larger.

Table 12: Comparison on SMALLMNIST5 (Local 0.13 Domain, L_∞ Norm)

Algorithm	Upper Bound	Lower Bound	Time (s)	Outcome
LipPOT (Ours)	**102.80**	101.98	40.46	Stable
auto_LIRPA	6143.19	NaN	0.80	Exploded
LipMIP	5484.33	NaN	1833.00	**Timeout**

10.5.2.1 Scalability and Stability

Table 12 demonstrates the failure modes of deterministic methods:

- **LipMIP (Timeout):** The exact solver runs for 30 minutes (1833s) and fails to converge. It returns an upper bound of 5484.33, which is likely just a loose bound from the branch-and-bound tree, not a verified tight bound.
- **auto_LIRPA (Explosion):** The analytical bound propagation suffers from the wrapping effect, returning a value (6143.19) that is useless for verification purposes.

- **LipPOT (Stable):** Our method remains computationally efficient (40s) and returns a tight, stable interval $[101.98, 102.80]$.

While the result of 102.80 allows for a verification decision where other methods fail, we must interpret it in light of the underestimation seen in Table 11. It is probable that the true maximum is slightly higher than 102.80, but remarkably unlikely to be anywhere near the 5000+ range reported by the baselines. LipPOT effectively filters out the loose overestimations to reveal the true scale of the Lipschitz constant.

10.5.3 Limitations: Sources of Underestimation

The results highlight that LipPOT provides a **statistical certification**, distinct from deterministic verification. There are two primary theoretical risks where the method may underestimate the true constant:

1. **Sampling Failure (Statistical Risk):** As evidenced by the $\epsilon = 0.01$ experiment, if the sampler misses the global peak entirely, the EVT model will be fit to the local peaks it did observe. The resulting L_{high} will be a valid upper bound for the *observed* distribution, but invalid for the *unobserved* global anomaly. This risk is controlled by the quality of the sampling engine but can never be reduced to zero.

2. **Non-Local Lipschitz Behavior (Geometric Risk):** Our approach assumes that the Lipschitz constant is well-characterized by the maximum local gradient norm $\max \| J_f(x) \|$. For neural networks with ReLU activations, the function is continuous but non-smooth. The global Lipschitz constant is defined as $\sup_{x \neq y} \frac{\|f(x)-f(y)\|}{\|x-y\|}$. In most cases, this equals the maximum gradient norm. However, theoretically, the Lipschitz constant could be defined by a "short-circuit" between two distant points across a decision boundary or a sharp valley, which would not be reflected in any local Jacobian. Since our method relies on gradient norms, it cannot detect non-local Lipschitz violations.

10.6 Conclusion

In this chapter, we have presented a comprehensive statistical framework for the verification of neural networks. By shifting the verification paradigm from intractable deterministic guarantees to high-confidence **PAC Intervals**, we have developed a method that scales to problems where exact solvers fail.

Our contribution is twofold. First, we established the **DKW-Constrained POT** method, a rigorous statistical engine that translates tail samples into a conservative upper bound for the Lipschitz constant. We showed how the Generalized Pareto Distribution, constrained by the Reverse Weibull assumption, provides a physically motivated model for the finite limits of neural network gradients.

Second, we addressed the scarcity of tail data with the **Adam-Sobol** sampling strategy. We demonstrated that simple random sampling is insufficient in high dimensions and that a targeted adversarial attack coupled with low-discrepancy Sobol refinement is necessary to populate the tail of the distribution.

The experimental results validate this approach as a pragmatic trade-off. While we sacrifice the absolute certainty of exact methods—accepting a small, quantifiable risk of underestimation—we gain the ability to generate tight, meaningful bounds in seconds where other methods take hours or fail completely. This suggests that statistical verification, when grounded in rigorous Extreme Value Theory, offers a viable path forward for certifying the robustness of increasingly complex deep learning systems.

11 Verification of EVENFLOW Use Cases

11.1Verification of Industry 4.0 use case

Autonomous robots are poised to become a foundational component of next-generation factory automation, yet their dependable operation in highly dynamic industrial environments remains a major challenge. A critical limitation lies in ensuring that the neural networks governing perception, prediction, and control behave reliably under all possible operating conditions. To address this formal verification techniques are applied to neural systems deployed on autonomous mobile robots (AMRs), with the goal of providing provable safety guarantees.

AMRs use deep neural networks to interpret sensor data, detect obstacles, recognise objects, and make navigation decisions. However, these models are inherently opaque and can behave unpredictably when exposed to noisy, adversarial, or rare environmental conditions. In industrial settings—where robots must operate among workers, other robots, and fast-changing surroundings—such unpredictability may result in unsafe stops, near-misses, or collisions, disrupting production and posing significant safety risks. Traditional testing and empirical evaluation cannot feasibly cover the enormous space of possible inputs generated by high-frequency sensors.

Formal verification provides a principled solution by mathematically analysing neural networks to ensure that their outputs remain safe under all input variations within a specified range. These methods can reason over entire sets of sensor readings, certify robustness to perturbations, and verify that safety-critical decisions (e.g., obstacle detection, emergency braking, or collision avoidance) hold across all relevant scenarios. For factory-deployed AMRs with constrained computational resources, verification must also account for the large volumes of sensor data and the need for lightweight, efficient neural models.

Emerging work combines advanced verification algorithms, symbolic reasoning, and scalable neural approximations to validate AMR behaviours before deployment. By leveraging large datasets, simulations, and domain-specific safety constraints, researchers aim to produce certifiably robust autonomous systems that can anticipate problematic situations while guaranteeing safe responses. Integrating formal verification into the development pipeline promises not only to increase the reliability and safety of AMRs but also to enhance manufacturing productivity and trust in AI-enabled automation.

11.1.1 Neurosymbolic model for robot path planning and collision avoidance

To facilitate tractable formal verification, we employ a deliberately compact yet sufficiently expressive convolutional neural network (CNN) to serve as the perception component of the neurosymbolic collision-avoidance system. The model processes raw visual observations from an autonomous robot platform and predicts whether the robot’s current trajectory is on a collision course with another robot in the environment. The dataset used for this study was provided by the German Research Center for Artificial Intelligence (DFKI). It consists of synchronized left- and right-camera video streams recorded from two mobile robots operating in a shared indoor space. Each image captures the robot’s first-person viewpoint, and potential collision events can be inferred when the second robot appears within the camera's field of view.

The original camera frames have a resolution of 1280×720 pixels. To reduce computational overhead and improve verification scalability, all images are down sampled to 160×90 before being fed into the neural network. The resulting input tensor has shape $3 \times 160 \times 90$, corresponding to RGB channels and reduced spatial dimensions. The output of the neural network is classified into 20 bins representing different spatial configurations and motion patterns associated with the two robots, enabling symbolic reasoning modules to determine whether their trajectories are likely to intersect.

Figure 66 shows the overall neuro-symbolic architecture. The CNN comprises a sequence of convolutional layers with ReLU activations and periodic max-pooling, progressively reducing spatial resolution while increasing feature depth. After ten convolutional blocks, the feature map is flattened into a 384-dimensional vector and passed through a two-layer classifier (384→20→num_classes).

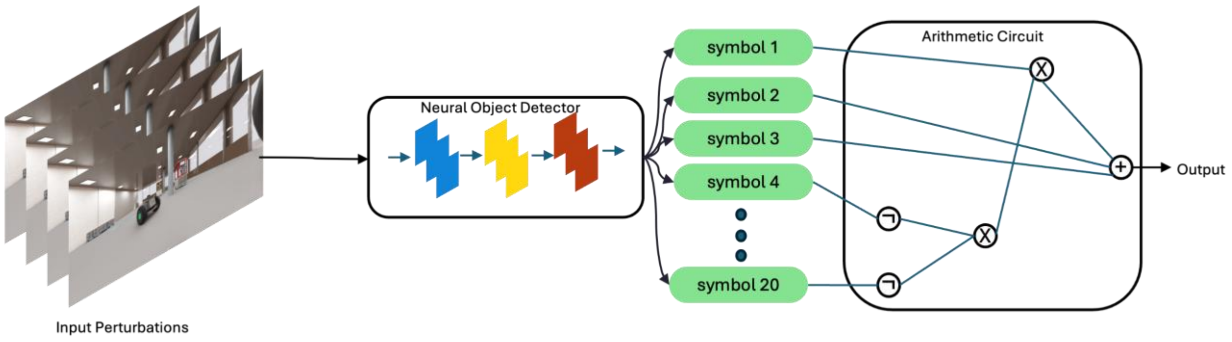


Figure 66: Neurosymbolic model for robot collision avoidance using the DFKI dataset.

11.1.2 Verification of the Neurosymbolic system for Industry 4.0

In this section, we report the formal verification results obtained for the neuro-symbolic collision-avoidance model developed for the DFKI robotic perception use case. The underlying neural architecture was trained using a 5-fold cross-validation strategy to ensure robustness and mitigate overfitting. For each fold, an independently trained model was subjected to verification, enabling us to assess the consistency and reliability of the verification outcomes across five distinct data splits. This evaluation methodology provides a comprehensive view of the model's behaviour under varying training distributions and supports a statistically grounded assessment of its safety properties.

To analyse robustness, we compute certified guarantees under a range of ϵ-bounded perturbations applied to the input images. These perturbations simulate realistic sensor noise and environmental variability that may arise during robot operation. Verification is performed using a custom verification toolkit built on top of the auto_LiRPA framework, which enables sound relaxation-based bound propagation for convolutional neural networks. The toolkit supports the computation of lower and upper bounds on network outputs under adversarially bounded perturbations, thereby allowing us to determine whether safety-critical classification decisions—such as detecting potential robot trajectory collisions—remain invariant within specified perturbation sets. IBP is used to verify the network across various noise perturbations.

The resulting certified robustness metrics for all five cross-validation splits (Split 1 ... 5) are summarized in Table 13 showing the proportion of test samples for which safety predictions remain formally guaranteed across different perturbation magnitudes.

Table 13: Verification results using IBP across 5 splits for various perturbation ranges.

Split	**Epsilons**						
	1e^{-10}	**1e^{-9}**	**1e^{-8}**	**1e^{-7}**	**1e^{-6}**	**1e^{-5}**	**1e^{-10}**
Split 1	75.73%	75.73%	72.52%	24.05%	0.0063%	0.00%	75.73%
Split 2	79.03%	79.01%	68.36%	0.10%	0.00%	0.00%	79.03%
Split 3	76.55%	76.53%	67.58%	0.31%	0.00%	0.00%	76.55%
Split 4	76.21%	76.21%	64.09%	0.49%	0.00%	0.00%	76.21%
Split 5	78.48%	78.47%	72.92%	1.08%	0.00%	0.00%	78.48%

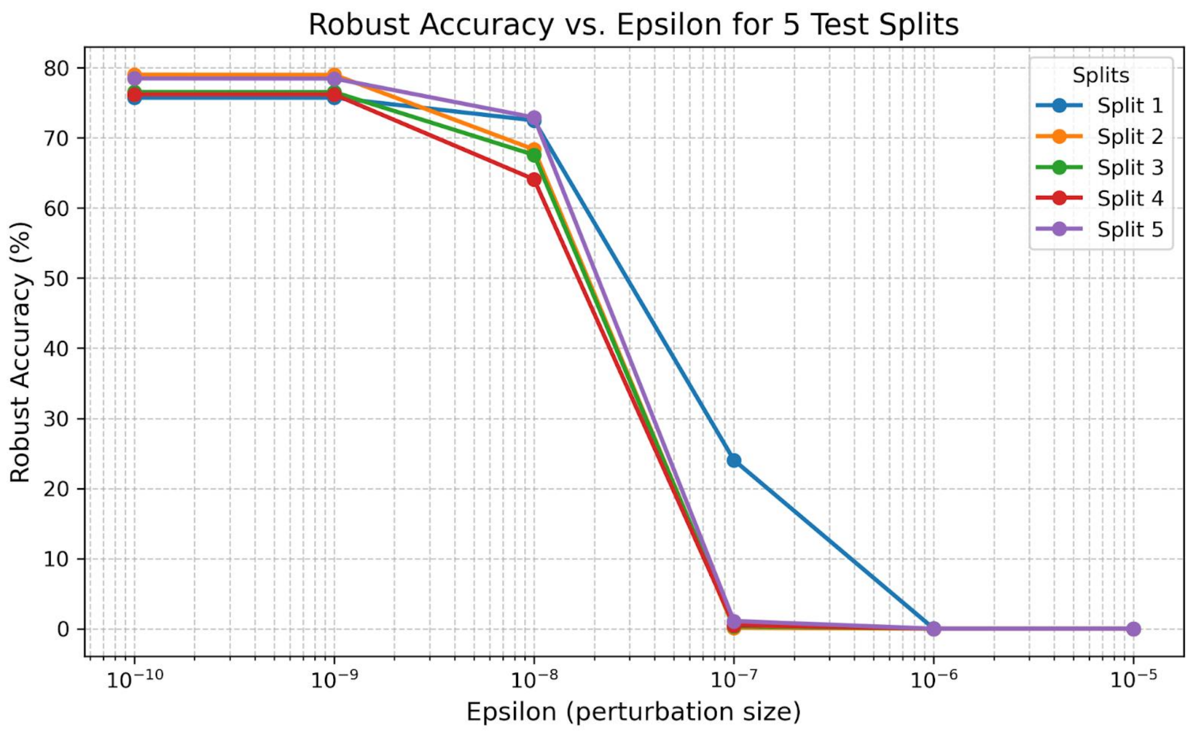


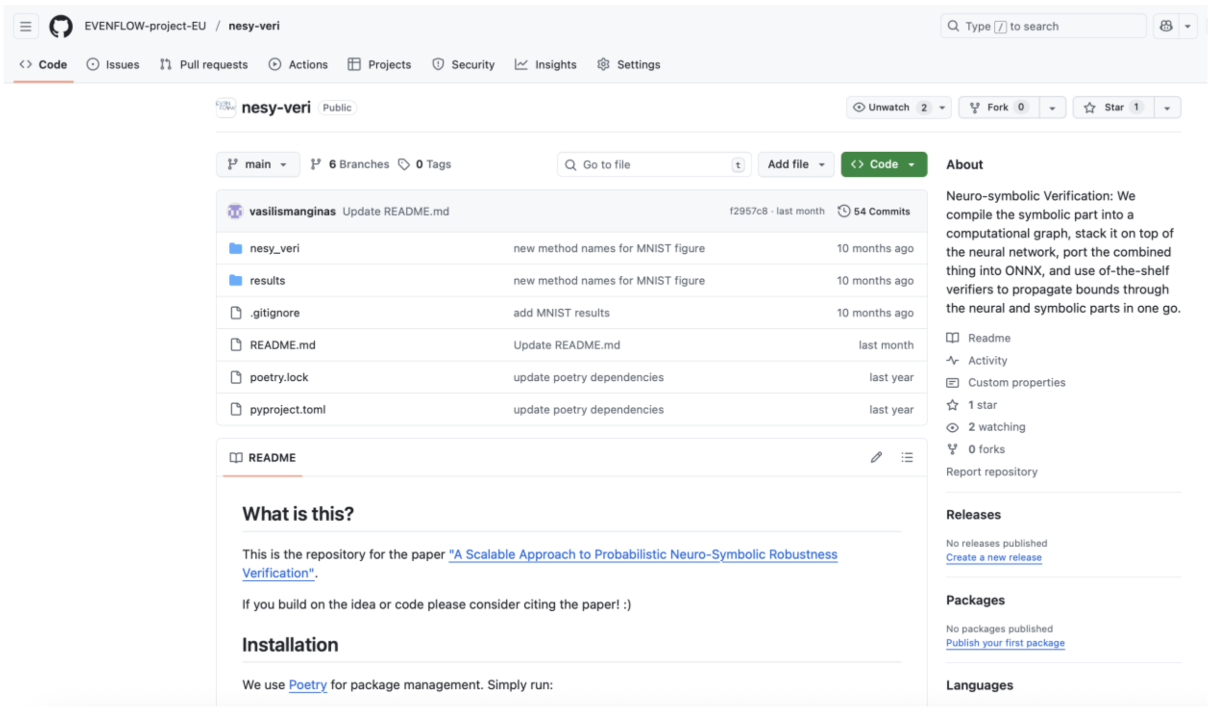
Figure 67: Robust accuracy of the NeSy model at various noise perturbations.

To further enhance the verification performance of the proposed neuro-symbolic model, future work will focus on tightening the verification bounds through the integration of more advanced and expressive certification techniques. In particular, we plan to employ hybrid bound-propagation methods such as CROWN+IBP, which combine linear relaxation with interval reasoning to obtain significantly tighter output bounds while maintaining computational tractability. Such methods have been shown to reduce over-approximation error and are expected to yield stronger certified robustness guarantees for high-dimensional perceptual models.

In parallel, we intend to investigate training strategies that explicitly promote robustness during model optimization. This includes *adversarial training*, where the model is exposed to worst-case perturbations during training to improve empirical robustness, as well as *certified training*, which incorporates verification-aware objectives that encourage the network to produce representations more amenable to formal certification. Together, these approaches aim to systematically strengthen the model's resilience to input perturbations, ultimately improving both its empirical performance and its provable safety properties.

12 Status of the EVENFLOW Verification Toolkit

The EVENFLOW verification toolkit is available as a public repository at - https://github.com/EVENFLOW-project-EU/nesy-veri.



nesy-veri Public

6 Branches 0 Tags

vasilismanginas Update README.md f2957c8 · last month 54 Commits

nesy_veri	new method names for MNIST figure	10 months ago
results	new method names for MNIST figure	10 months ago
.gitignore	add MNIST results	10 months ago
README.md	Update README.md	last month
poetry.lock	update poetry dependencies	last year
pyproject.toml	update poetry dependencies	last year

README

What is this?

This is the repository for the paper "A Scalable Approach to Probabilistic Neuro-Symbolic Robustness Verification".

If you build on the idea or code please consider citing the paper! :)

Installation

We use Poetry for package management. Simply run:

About

Neuro-symbolic Verification: We compile the symbolic part into a computational graph, stack it on top of the neural network, port the combined thing into ONNX, and use of-the-shelf verifiers to propagate bounds through the neural and symbolic parts in one go.

Readme
Activity
Custom properties
1 star
2 watching
0 forks
Report repository

Releases

No releases published
Create a new release

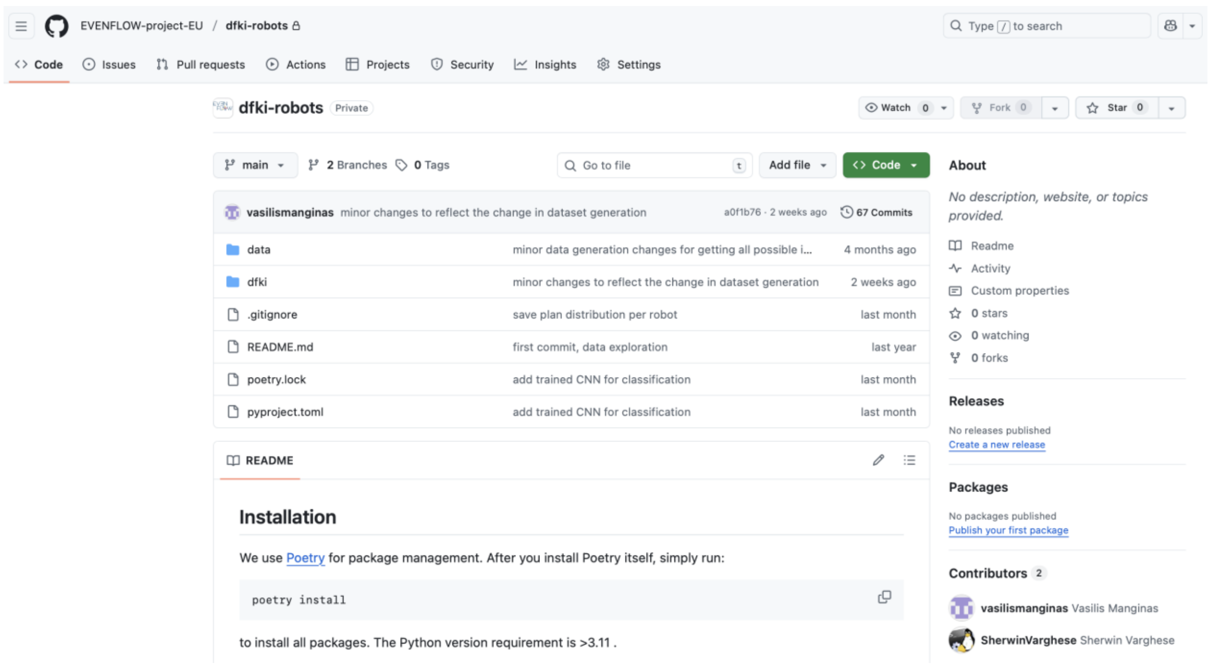
Packages

No packages published
Publish your first package

Languages

Figure 68: The verification toolkit at GitHub.

The verification toolkit for the Industry 4.0 robot navigation use case with Neuro-Symbolic models built on the DFKI dataset is available at - https://github.com/EVENFLOW-project-EU/dfki-robots.



dfki-robots Private

2 Branches 0 Tags

vasilismanginas minor changes to reflect the change in dataset generation a0f1b76 · 2 weeks ago 67 Commits

data	minor data generation changes for getting all possible i...	4 months ago
dfki	minor changes to reflect the change in dataset generation	2 weeks ago
.gitignore	save plan distribution per robot	last month
README.md	first commit, data exploration	last year
poetry.lock	add trained CNN for classification	last month
pyproject.toml	add trained CNN for classification	last month

README

Installation

We use Poetry for package management. After you install Poetry itself, simply run:

```
poetry install
```

to install all packages. The Python version requirement is >3.11 .

About

No description, website, or topics provided.

Readme
Activity
Custom properties
0 stars
0 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2

vasilismanginas Vasilis Manginas
SherwinVarghese Sherwin Varghese

Figure 69: The verification toolkit for the Industry 4.0 robot navigation use case at GitHub.

13 References

[REF-01]	George Kloumis, Nikos Giatrakos: Data-driven Synchronization Protocols for Data-parallel Neural Learning over Streaming Data. IEEE Big Data 2024: 988-997
[REF-02]	Antonios Kontaxakis, Nikos Giatrakos, Dimitris Sacharidis, Antonios Deligiannakis: And synopses for all: A synopses data engine for extreme scale analytics-as-a-service. Inf. Syst. 116: 102221 (2023)
[REF-03]	Errikos Streviniotis, George Klioumis, Nikos Giatrakos: SuBiTO: Synopsis-based Training Optimization for Continuous Real-Time Neural Learning over Big Streaming Data. AAAI 2025: 29697-29699
[REF-04]	Eric Brochu, Vlad M. Cora, and Nando de Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. arXiv:1012.2599 [cs.LG]
[REF-05]	Peter I. Frazier. 2018. A Tutorial on Bayesian Optimization. arXiv:1807.02811 [stat.ML]
[REF-06]	Reddy, K.; and Shah, M. 2013. Recognizing 50 human action categories of web videos. Machine Vision and Applications, 24
[REF-07]	Nick Duffield & Carsten Lund & Mikkel Thorup. “Priority sampling for estimation of arbitrary subset sums”. In: Journal of the ACM (JACM) 54 (6 2007), pp. 1–39.
[REF-08]	Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, Bor-Yiing Su: Scaling Distributed Machine Learning with the Parameter Server. OSDI 2014: 583-598
[REF-09]	C. Chen, W. Wang, and B. Li, “Round-robin synchronization: Mitigating communication bottlenecks in parameter servers,” in IEEE INFOCOM 2019
[REF-10]	Ourania Ntouni, Dimitrios Banelas, Nikos Giatrakos: NeuroFlinkCEP: Neurosymbolic Complex Event Recognition Optimized across IoT Platforms. Proc. VLDB Endow. 18(12): 5355-5358 (2025)
[REF-11]	Errikos Streviniotis, Dimitrios Banelas, Nikos Giatrakos, Antonios Deligiannakis: DAG*: A Novel A*-Alike Algorithm for Optimal Workflow Execution Across IoT Platforms. ICDE 2025: 807-820
[REF-12]	Nikos Giatrakos: SSTRESED: Scalable Semantic Trajectory Extraction for Simple Event Detection over Streaming Movement Data (Extended Abstract). TIME 2023: 15:1-15:4
[REF-13]	Zhixian Yan, Nikos Giatrakos, Vangelis Katsikaros, Nikos Pelekis, Yannis Theodoridis: SeTraStream: Semantic-Aware Trajectory Construction over Streaming Movement Data. SSTD 2011: 367-385
[REF-14]	Errikos Streviniotis, Nikos Giatrakos, Yannis Kotidis, Thaleia Ntiniakou, Miguel Ponce de Leon: Optimizing Resource Allocation for Tumor Simulations over HPC Infrastructures. DSAA 2023: 1-10
[REF-15]	Errikos Streviniotis, Nikos Giatrakos, Yannis Kotidis, Thaleia Ntiniakou, Miguel Ponce de Leon: RATS: A resource allocator for optimizing the execution of tumor simulations over HPC infrastructures. Inf. Syst. 132: 102538 (2025)
[REF-16]	J. Koning, T. Patki, T.R. Scogland, B. Springmeyer, M. Taufer, Flux: Overcoming scheduling challenges for exascale workflows, Future Gener. Comput. Syst. 110

	(2020) 202–213
[REF-17]	D. Rafiei and A. Mendelzon. 1997. Similarity-based queries for time series data. In Proceedings of the 1997 ACM SIGMOD international conference on Management of data (SIGMOD '97). https://doi.org/10.1145/253260.253264
[REF-18]	I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," IEEE Trans. Inf. Theory, vol. 36, no. 5, pp. 961–1005, Sep. 1990, doi: 10.1109/18.57199.
[REF-19]	N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," IEEE Trans. Comput., vol. C–23, no. 1, pp. 90–93, Jan. 1974, doi: 10.1109/T-C.1974.223784.
[REF-20]	E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases," Knowl. Inf. Syst., vol. 3, no. 3, pp. 263–286, Aug. 2001, doi: 10.1007/PL00011669
[REF-21]	Nikos Giatrakos, Yannis Kotidis, Antonios Deligiannakis, Vasilis Vassalos, Yannis Theodoridis: TACO: tunable approximate computation of outliers in wireless sensor networks. SIGMOD Conference 2010: 279-290
[REF-22]	Nikos Giatrakos, Antonios Deligiannakis, Minos N. Garofalakis, Yannis Kotidis: Omnibus outlier detection in sensor networks using windowed locality sensitive hashing. Future Gener. Comput. Syst. 110: 587-609 (2020)
[REF-23]	Antonios Skevis, George Klioumis, Nikos Giatrakos: A Novel Reverse Random Hyperplane Projection Scheme and Its Effect on Mining Sensor Streams. IEEE Big Data 2024: 793-798
[REF-24]	P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in SenSys, 2003.
[REF-25]	Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A. and Misener, R. (2020) "Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis", Proceedings of the AAAI Conference on Artificial Intelligence, 34(04), pp. 3291-3299. doi: 10.1609/aaai.v34i04.5729.
[REF-26]	K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems", in 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), IEEE, 2016, pp. 1–10.
[REF-27]	Hornik, K., Stinchcombe, M. and White, H., 1989. Multilayer feedforward networks are universal approximators. Neural networks, 2(5), pp.359-366.
[REF-28]	Emanuele Marconato, Samuele Bortolotti, Emile van Krieken, Antonio Vergari, Andrea Passerini, and Stefano Teso. Bears make neuro-symbolic models aware of their reasoning shortcuts. arXiv preprint arXiv:2402.12240, 2024.
[REF-29]	Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. CoRR, abs/1810.12715, 2018. URL http://arxiv.org/abs/1810.12715.
[REF-30]	Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kuncak, editors, Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, volume 10426 of Lecture Notes in Computer Science, pages 97–117. Springer, 2017. doi: 10.1007/ 978-3-319-63387-9\5. URL https://doi.org/10.1007/978-3-319-63387-9_5.

[REF-31]	Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results, 2024. URL https://arxiv.org/abs/2412.19985.
[REF-32]	Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zelji´c, et al. The marabou framework for verification and analysis of deep neural networks. In Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31, pages 443–452. Springer, 2019.
[REF-33]	Kaidi Xu, Zhouxing Shi, Huan Zhang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis on general computational graphs. CoRR, abs/2002.12920, 2020. URL https://arxiv.org/abs/2002.12920.
[REF-34]	Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In ECAI 2020, pages 2513–2520. IOS Press, 2020.
[REF-35]	Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. Advances in neural information processing systems, 31, 2018.
[REF-36]	Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions, 2018. URL https://arxiv.org/abs/1811.00866.
[REF-37]	Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results, 2024. URL https://arxiv.org/abs/2412.19985.
[REF-38]	Eleonora Giunchiglia, Mihaela C˘at˘alina Stoian, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. Road-r: the autonomous driving dataset with logical requirements. Machine Learning, 112(9):3261–3291, 2023.
[REF-39]	Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovi´c, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. ArXiv, abs/1810.12715, 2018. URL https://api.semanticscholar.org/CorpusID:53112003.
[REF-40]	Jerry Lambert, Amedeo Ceruti, and Hartmut Spliethoff. Benchmark of mixed-integer linear programming formulations for district heating network design. Energy, 308:132885, 2024
[REF-41]	In Gyu Lee, Qianqian Zhang, Sang Won Yoon, and Daehan Won. A mixed integer linear programming support vector machine for cost-effective feature selection. Knowledge-based systems, 203:106145, 2020.
[REF-42]	Yun Hui Lin, Yuan Wang, Dongdong He, and Loo Hay Lee. Last-mile delivery: Optimal locker location under multinomial logit choice model. Transportation Research Part E: Logistics and Transportation Review, 142:102059, 2020.
[REF-43]	Debangshu Banerjee and Gagandeep Singh. Relational dnn verification with cross executional bound refinement. In Proceedings of the 41st International Conference on Machine Learning, ICML'24. JMLR.org, 2024.
[REF-44]	Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-

	to-end 3d detection, tracking and motion forecasting with a single convolutional net. In 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 3569–3577. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00376. URL http://openaccess.thecvf.com/content_cvpr_2018/ html/Luo_Fast_and_Furious_CVPR_2018_paper.html.
[REF-45]	Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.
[REF-46]	Akshay Agarwal, Mayank Vatsa, Richa Singh, and Nalini K. Ratha. Noise is inside me! generating adversarial perturbations with noise derived from natural filters. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2020.
[REF-47]	Nabeel Hingun, Chawin Sitawarin, Jerry Li, and David Wagner. Reap: A large-scale realistic adversarial patch benchmark. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 4640–4651, October 2023.
[REF-48]	Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/ file/5d4ae76f053f8f2516ad12961ef7fe97-Paper.pdf.
[REF-49]	Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. CoRR, abs/1212.0402, 2012. URL http://arxiv.org/abs/1212.0402.
[REF-50]	Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. CoRR, abs/2110.14795, 2021. URL https://arxiv.org/abs/2110.14795.
[REF-51]	AR Udacity. Udacity self-driving car dataset, 2017.
[REF-52]	Scaling provable adversarial defenses. In Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS18), 2018.