POLITECNICO DI TORINO
Repository ISTITUZIONALE

The inNuCE Research Infrastructure and the Neuromorphic MLOps for AIoT prototyping

Original
The inNuCE Research Infrastructure and the Neuromorphic MLOps for AIoT prototyping / Urgese, G., Fra, V., Pignata, A., Fanuli, G., Gomez, W.G., Pignari, R., Barocci, M., Leto, B., Tilocca, S., Cassetta, N., Montuschi, P., Macii, E.. - In: IEEE INTERNET OF THINGS JOURNAL. - ISSN 2327-4662. - 13:9(2026), pp. 19286-19299. [10.1109/jiot.2026.3663902]

Availability:
This version is available at: 11583/3009017 since: 2026-03-20T23:24:26Z

Publisher:
Institute of Electrical and Electronics Engineers

Published
DOI:10.1109/jiot.2026.3663902

Terms of use:

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

Publisher copyright

(Article begins on next page)

14 November 2026

The inNuCE Research Infrastructure and the Neuromorphic MLOps for AIoT Prototyping

Gianvito Urgese, *Senior Member, IEEE*, Vittorio Fra, Andrea Pignata, *Graduate Student Member, IEEE*, Giuseppe Fanuli, Walter Gallego Gomez, Riccardo Pignari, Michelangelo Barocci, *Graduate Student Member, IEEE*, Benedetto Leto, Salvatore Tilocca, Nicola Cassetta, Paolo Montuschi, *Fellow, IEEE*, and Enrico Macii, *Fellow, IEEE*

***Abstract*—Neuromorphic computing promises significant improvements in latency and energy efficiency for machine intelligence at the edge. However, its adoption in the IoT domain is still limited by the heterogeneity of the hardware (HW), the immaturity of the toolchains, and the poor reproducibility of experiments. This article sets out the inNuCE research infrastructure (inNuCE RI), a two-pillar facility composed of a physical laboratory (inNuCE Lab) and a cloud-based heterogeneous prototyping platform (inNuCE HPP). The purpose of the inNuCE RI is to enable developers to prototype, evaluate, and compare neuromorphic and conventional end-to-end digital solutions. From a methodological perspective, we formalize the adaptation of machine learning operations (MLOps) to event-driven sensing and brain-inspired computation as neuromorphic MLOps (NMLOps). We illustrate how inNuCE RI instantiates NMLOps through containerized toolchains orchestrated with Kubernetes and Slurm-managed heterogeneous resources (neuromorphic chips, FPGAs, GPUs, and MCUs). The approach is analyzed on representative Artificial Intelligence of Things (AIoT) use cases, including human activity recognition (HAR), Braille reading, event-based gesture recognition, Hi-Co semantization of memories, navigation tracking, and constraint satisfaction problems. The development of inNuCE RI has been driven by the need to facilitate the transition from prototype (*in nuce*) to engineered AIoT systems for lower entry barriers and enforce reproducibility. This paves the way for future system-of-systems engineering.**

***Index Terms*—Artificial Intelligence of Things (AIoT), brain-inspired computing, event-driven sensors, heterogeneous systems, neuromorphic computing, neuromorphic MLOps (NMLOps), spiking neural networks (SNNs).**

I. Introduction

THE proliferation of Artificial Intelligence of Things (AIoT) applications in manufacturing, robotics, healthcare, logistics, and mobility demands computation that is simultaneously low power, low latency, and scalable. General-purpose CPUs/GPUs deliver high accuracy but often fail to satisfy strict power constraints when operating at the extreme edge. Neuromorphic computing, with its event-driven processing, sparse activation, and massive parallelism, naturally fits sensor-rich, time-critical Intelligence of Things (IoT) scenarios [1], [2].

Neuromorphic platforms, including SpiNNaker 2 [2], Intel Loihi 2 [3], BrainChip Akida [4], BrainDrop [5], and SynSense boards [6], in conjunction with event-driven sensors, such as dynamic vision sensor (DVS) [7] and silicon cochleae [8], demonstrate the capacity of spiking neural network (SNN) to deliver high-performance inference within stringent resource budgets at different scales [9]. By exploiting massive parallelism, sparse activation, and event-driven operation, these systems are well-suited to edge intelligence in AIoT. Nevertheless, despite these advantages, the large-scale adoption of neuromorphic computing in industrial and applied IoT contexts remains limited [10]. Key obstacles include: 1) the difficulty of acquiring, labeling, and curating event-driven, time-varying datasets, which undermines reproducibility; 2) Hardware (HW) heterogeneity across chip families (distinct computational models, programming abstractions, and SDKs); 3) skills gaps in designing and training models that exploit sparsity and temporal coding; 4) fragmented Software (SW) ecosystems due to the absence of standardized tools and APIs; and 5) limited commercialization, which keeps many devices as precommercial prototypes requiring expert deployment.

Machine learning operations (MLOps) is a set of practices that apply DevOps principles to machine learning (ML) [11]. It automates the full model lifecycle, from data preparation and training to deployment, monitoring, and retraining. The aim is to ensure reliable, scalable, and production-ready ML systems, while bridging the gap between data science and operations teams. Within the conventional edge artificial intelligence (edge-AI) ecosystem, several cloud-based development platforms (such as Edge Impulse, SensiML, STM32Cube.AI Dev Cloud, ST AIoT Craft, Arduino Cloud, and BrainChip Akida Cloud) leverage integrated MLOps pipelines to improve accessibility and enable automated deployment. However, these solutions are largely vendor-specific and primarily designed for standard CPU and GPU architectures, offering limited support for neuromorphic workflows. Consequently, a gap

Received 21 October 2025; revised 30 December 2025; accepted 4 February 2026. Date of publication 11 February 2026; date of current version 27 April 2026. This work was supported by Italian National Recovery and Resilience Plan (NRRP), M4C2 through European Union–NextGenerationEU (EBRAINS-Italy) under Project IR0000011 and Project CUP B51E22000150006. *(Corresponding author: Gianvito Urgese.)*

Gianvito Urgese, Vittorio Fra, Giuseppe Fanuli, Benedetto Leto, Salvatore Tilocca, Nicola Cassetta, and Enrico Macii are with the Interuniversity Department of Regional and Urban Studies and Planning (DIST), Politecnico di Torino, 10125 Turin, Italy (e-mail: gianvito.urgese@polito.it).

Andrea Pignata, Walter Gallego Gomez, Riccardo Pignari, Michelangelo Barocci, and Paolo Montuschi are with the Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, 10129 Turin, Italy.

Digital Object Identifier 10.1109/JIOT.2026.3663902

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

remains between the capabilities of neuromorphic HW and its effective integration into industrial AIoT development and deployment processes [12].

To bridge this gap, the present study proposes the definition of the neuromorphic MLOps (NMLOps) as the process of managing the neuromorphic-aware ML lifecycle. This extends reproducibility, automation, and scalability principles introduced by the MLOps architectures to event-driven brain-inspired computing. Specifically, NMLOps is required to as follows.

1) Provides rigorous dataset and model versioning (including spike-encoding methods, parameters, and time bases).
2) Delivers end-to-end pipelines covering data acquisition, preprocessing, training, hyperparameter optimization, simulation, compilation, deployment, and benchmarking.
3) Makes HW abstraction interfaces available to hide heterogeneity while enabling targeting of multiple neuromorphic chips from a unified workflow.
4) Integrates automated ML (AutoML) tools for selecting neuromorphic-specific options (encoding schemes, neuron/time constants, and learning strategies).
5) Supports seamless orchestration across cloud resources and heterogeneous HW pools.

A complementary contribution proposed in this article is the inNuCE research infrastructure (inNuCE RI), a two-pillar infrastructure that instantiates NMLOps in practice for neuromorphic AIoT prototyping. The name inNuCE is derived from the Latin *in nuce* ("in the shell," "in embryo") reflecting the mission to enable developers to create draft prototypes rapidly and then transition them to engineered products once feasibility is established. The first pillar is the Laboratory (inNuCE Lab), a physical facility housing event-based sensors, edge devices, and neuromorphic/digital boards for hands-on experimentation. Pillar two is the heterogeneous prototyping platform (inNuCE HPP), a platform-as-a-service (PaaS) that virtualizes heterogeneous HW and enforces reproducibility via containerized toolchains.

The remainder of this article reviews the related work and current limitations in AIoT prototyping and neuromorphic computing in Section I. Section III provides a detailed description of the NMLOps process that is supported by the inNuCE RI cloud-based prototyping platform, which is described in Section IV. Section V discusses the use cases and evaluation results, and Section VI concludes this article.

II. Related Work and Background

Neuromorphic computing is rooted in a fundamental departure from conventional computing, as it draws inspiration from the human brain to emulate its structure, dynamics, and plasticity [13], [14], [15]. At its core, neuromorphic computing is built on sparse, event-driven processing. Information is encoded in sparse, asynchronous events referred to as spikes, and computation is triggered by their emission rather than being performed at a fixed clock rate. This is analogous to how neurons in the brain communicate, where information is encoded in the timing and frequency of electrical pulses [16]. This approach is particularly well-suited for AIoT ecosystems dominated by battery-powered sensors, whose data are often sparse and intermittent [1]. By only processing data when there is a change in the input, event-driven systems can significantly reduce power consumption and computational overhead, as the system remains in a low-consumption regime when there is no activity [17]. In stark contrast, traditional clock-driven systems consume power continuously, even when no useful work is performed. The event-driven nature of neuromorphic computing makes it an ideal candidate for battery-powered IoT devices, where energy efficiency is a primary concern.

The beneficial impact offered by neuromorphic computing spans the entire AIoT stack. Energy efficiency can improve over standard architectures because only active circuits dissipate power; asynchronous processing can guarantee reaction times within or below the milliseconds regime; the native ability to operate on sparse, temporal data removes dependence on large annotated datasets and slashes communication bandwidth to the cloud [18]. These advantages can take shape through a heterogeneous HW, SW, and sensor ecosystem. The HW landscape covers various design styles: digital, analog, or mixed-signal, in discrete or continuous time, with standard architectures or in-memory computing [12]. The SW support is characterized by the utilization of numerous frameworks, including snnTorch, Nengo, GeNN, Lava, Norse, and SpikingJelly, among others [16]. Concurrently, event-based vision sensors [7], neuromorphic tactile sensors [19], machine olfaction [20], and acoustic front ends inspired by the cochlea facilitate the extraction of sparse spike trains for the implementation of the next-generation use cases, such as keyword detection at microwatt power levels [8].

At the SW implementation level, SNN are the type of artificial neural network (ANN) inspired by the event-driven communication between biological neurons [21]. Unlike traditional ANN, which use continuous-valued signals to represent information, SNN use spikes to encode and transmit information, thus emulating the action potentials that are generated by neurons in the brain. Both timing and frequency of these spikes can be used to represent information, allowing SNN to process temporal data in a more natural and efficient way than traditional neural networks. This makes them particularly well-suited for AIoT applications dealing with time-series data.

Traditional artificial intelligence (AI) models often rely on cloud-based services, implying relevant limitations [22]. First, a constantly available and reliable connection is required; second, data must be protected to avoid privacy issues when transmitting back and forth from the cloud. Neuromorphic systems, by contrast, enable low-power on-device operations even including learning, and this clearly improves the privacy and security of the system, as well as its adaptability and robustness compared to entirely cloud-based AIoT solutions [9].

Neuromorphic computing is hence poised to have a transformative impact on a wide range of AIoT applications, with autonomous systems and smart environments being two of the most promising areas [23]. Looking at autonomous vehicles, drones, and robots, the need for real-time, low-power,

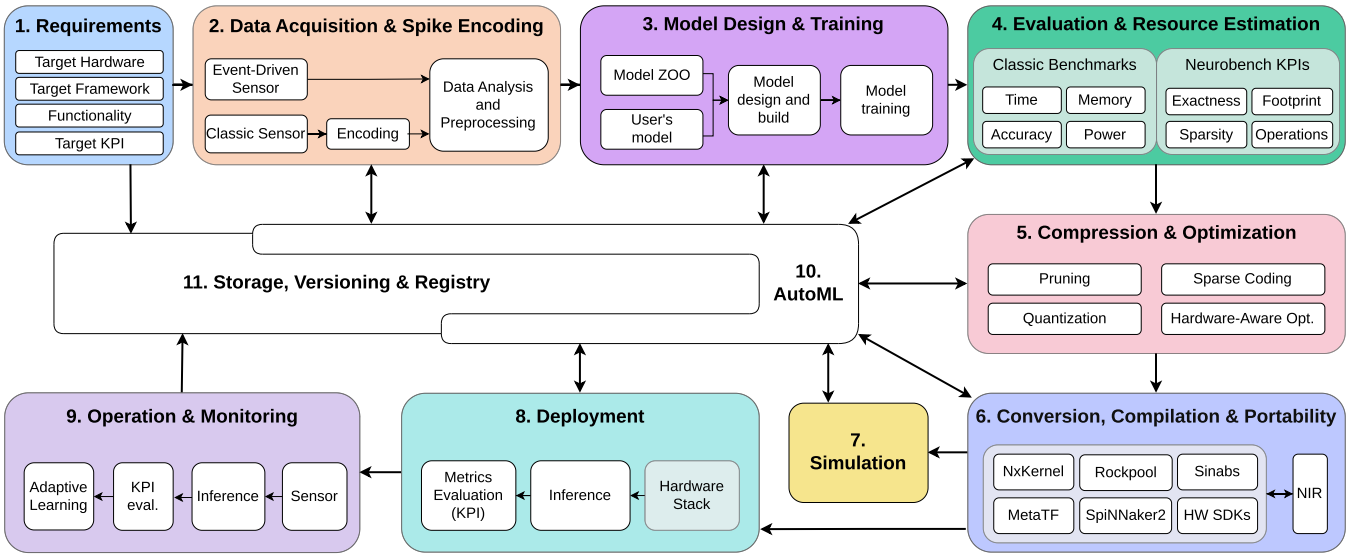

Fig. 1. NMLOps architecture: a MLOps pipeline adapted for neuromorphic systems. The diagram illustrates the adaptation of the MLOps loop with neuromorphic-specific elements, such as event-driven data ingestion, spike encoding, SNN training, and heterogeneous HW deployment. The NMLOps flow goes from 1 to 9, following the arrows. The user can choose to skip unneeded steps, based on the requirements, while other ones can be repeated in an iterative way exploiting AutoML techniques.

and intelligent processing is paramount [24]. Neuromorphic systems are particularly well-suited for these applications due to their ability to process sensory data in a highly efficient and parallel manner [25]. Furthermore, the low power consumption of neuromorphic HW is a key advantage for autonomous systems, as it can extend the operational time of battery-powered devices and reduce the thermal load on the system [26].

III. NMLOps Process

In modern AIoT setups, common MLOps practices can be adopted to support the entire lifecycle of edge AI models in various use cases, such as predictive maintenance, human activity recognition (HAR), visual inspection, autonomous mobile robots, and smart healthcare [27]. Production pipelines typically include dataset versioning and feature stores, containerized training at scale, continuous integration and delivery for both code and data, model registries with governance policies, staged rollouts with A/B and canary testing, telemetry-driven monitoring with automated retraining triggers, and over-the-air updates to fleets of devices [11]. These practices shorten iteration cycles, improve reproducibility and traceability, and reduce operational risk when models are refreshed on distributed devices [28]. Building on this foundation, we specialize the MLOps process for neuromorphic computing and explicitly connect it to AIoT use cases discussed in Section V.

NMLOps is a paradigm that adapts the MLOps lifecycle (data, model, deployment, and monitoring) to event-driven sensing and brain-inspired computation. In contrast to tensor-centric frameworks that target GPUs or tensor processing units (TPUs), neuromorphic pipelines must also manage asynchronous, sparse signals, nondifferentiable activations, and heterogeneous target HW [neuromorphic chips, FPGAs, and embedded microcontroller units (MCUs)], thereby introducing an additional co-design dimension across models, toolchains, and devices.

To ground the discussion, the following paragraphs walk through the phases in Fig. 1 as they are realized in a cloud-based NMLOps implementation built on Kubernetes and Slurm to support application prototyping and deployment. In this environment, Kubernetes serves as the state-of-the-art (SOTA) container orchestration framework, managing user workspaces and services, enforcing resource quotas, and enabling the secure, isolated, and reproducible execution of experiments. Slurm acts as the job scheduling and queue management system, orchestrating user-submitted jobs and experiments across heterogeneous HW resources. The usage of these tools for implementation is explained in Section IV.

A. Requirements

The first phase of the NMLOps process involves the identification of system requirements from developers, encompassing the application, dataset, encoding scheme, model type, target HW, and learning strategy. In addition, developers are tasked with the resolution of constraints and the establishment of objectives. The objective of this process is to ensure that designs are aligned with the intended deployment and that outcomes can be measured through relevant key performance indicators (KPIs).

This step defines six items.

1) HW targets (device class, memory and bandwidth, I/O, on-device learning, mismatch, power, and availability).
2) SW stack (frameworks, training strategy, conversion path, pinned containers and SDKs, and supported operators).
3) Functionality and operating limits (task scope and interfaces, latency and throughput, real-time deadlines, robustness and privacy, and on-device learning needs).

4) Data and encoding plan (modalities, labeling, spike-encoding parameters, calibration, and seeds).
5) KPIs and acceptance thresholds (accuracy sets, latency percentiles and jitter, energy per inference, average power, footprint, and drift/noise tests).
6) Techniques adopted to mitigate the mismatch caused by process variation in analog neuromorphic devices.
7) Deployment and monitoring plan (testbed and telemetry, energy and latency service level objective (SLO), canary strategy, and retraining triggers).

The actuation of this NMLOps plan is tracked in a dedicated registry to ensure all subsequent stages are traceable and thus repeatable across HW and SW configurations.

B. Data Acquisition and Spike Encoding

Event streams from DVS, artificial cochlea, and tactile sensors are characterized by asynchronous operations and sparsity, properties that reduce bandwidth yet introduce challenges in the processes of labeling, synchronization, and ground-truth alignment [29]. In circumstances where frame-based or time-varying sensors are utilized, raw signals undergo conversion to spike trains via delta/level-crossing sampling or rate/temporal coding [30].

C. Model Design and Training

In NMLOps, the model represents the SNN needed to execute inference on the data for different AIoT applications, including classification, regression, and predictions. Such networks need to be sized correctly for the intended application, finding a valid tradeoff that considers the input dimensionality, task complexity, HW limitations, and required output characteristics.

In SNN learning can target several classes of parameters beyond conventional synaptic weights. Synaptic weights regulate the strength of connections and are the most commonly trained parameters. Synaptic and axonal delays can also be trained to model temporal dynamics more accurately by controlling the propagation time of spikes between neurons, which is particularly relevant for time-coded and event-based processing [31]. Neuronal parameters, such as firing thresholds, membrane time constants, and refractory periods, may be optimized to tune neuronal excitability and responsiveness. Additionally, synaptic time constants governing postsynaptic current decay and adaptation parameters related to spike-frequency adaptation can be learned to improve temporal representation and energy efficiency. Training these diverse parameters enables SNNs to exploit both spatial and temporal degrees of freedom, making them well-suited for emergent event-driven computing systems [32]. SNN training can be done offline at design time by exploiting techniques, such as surrogate-gradient backpropagation or ANN-to-SNN transfer learning. Alternatively, also online, on-chip learning is possible, via local plasticity algorithms (e.g., spike-timing-dependent plasticity (STDP), eligibility propagation (e-prop), and homeostatic rules) as well as three-factor learning rules for reward-modulated updates [33].

The implementation of these NMLOps steps can be achieved through the integration of multiple frameworks (e.g., snnTorch, SpikingJelly, Lava, Rockpool, Sinabs, mlGeNN, Nengo, PyNN) within containerized environments. This approach ensures consistency across users and sites. It is evident that HW-awareness is a key consideration in the design process. Designs are developed with a focus on neuron and synapse budgets, memory hierarchies and tiling, numeric precision (fixed-point or quantized), routing constraints, and supported learning rules. Furthermore, when dealing with analog neuromorphic HW, the design phase should also account for device mismatches and carefully adapt the model to minimize its sensitivity to such variations [34]. Regularizers that control spike rate, burstiness, and temporal sparsity are treated as first-class hyperparameters. The conception of a curated *Neuromorphic Model Zoo*, which collects predesigned use cases and provides device-tuned models templates and reference configurations, will accelerate onboarding and reduce the development entry barriers [16].

D. Evaluation and Resource Estimation

Beyond accuracy, NMLOps standardizes latency, jitter, memory footprint, throughput, and energy (e.g., energy-delay product and joules per inference). Benchmarks, such as NeuroBench [35] and board-level profilers, provide comparable measurements across CPU, GPU, and neuromorphic targets. Early resource estimation and fit analysis predict mapping feasibility, on-chip memory use, and routing overhead to prevent late integration failures. Evaluation protocols employ repeated trials to capture nondeterminism and seed sensitivity and include temperature and voltage sweeps on embedded platforms. Golden-set tests verify functional equivalence after conversion and compilation.

E. Compression and Optimization

Spikes are characterized by their inherent sparsity, which can be exploited to reduce both memory footprint and computational cost. NMLOps account for this property and support the implementation of model sparsification through structured or unstructured pruning. This process involves the elimination of redundant synapses and neurons, while ensuring the preservation of model connectivity. The application of quantization to weights, activations, and neuron states has been demonstrated to facilitate low-power inference, thereby preventing degradation in accuracy. Additional transformations include synapse sharing or tying, connection reparameterization, and activity regularization, which trade small accuracy losses for large energy savings [16]. These transformations are integrated into the development pipeline, and their impact on accuracy, latency, and energy is measured against baselines and regression tested across iterations.

F. Conversion, Compilation, and Portability

To facilitate deployment, models are compiled into target-specific formats (e.g., NxKernel, Akida MetaTF, SpiNNaker tools, and SynSense toolchains). NMLOps promote containerized SDKs and intermediate representations (e.g., NIR [12])

for standardized neuromorphic computational primitives. The purpose of this promotion is twofold: first, to hide device-specific differences and second, to support portable releases built with a continuous integration and delivery workflow from a single codebase. The toolchain performs operator lowering, graph partitioning, placement, and routing within HW constraints. It then runs equivalence checks against a golden simulator with explicit numeric tolerance thresholds. Due to the intrinsic differences between neuromorphic HW, it could be possible that some features of an SNN developed for a target are not supported by another platform, undermining portability and integration between different technologies. A fundamental part of NMLOps is overcoming these difficulties, by hand (with considerable effort from a developer) or with the help of proper compiler toolchains (that currently are in the early stage), with the goal of preserving end-to-end functional equivalence. Solutions may include the generation a SW kernel, to be run on a traditional processor; the usage of an alternative operation supported by the target HW and with similar functionality; or the creation of a hybrid partition that offloads the execution of such subgraphs to a third-party digital accelerator.

G. Simulation

This phase provides a fast and controllable environment to validate SNN behavior. In NMLOps, simulation can be implemented either through the target vendor's HW/SW stack (such as event-accurate emulators shipped with the SDK, such as RockPool or Lava) or through generic simulators and reference backends (e.g., Brian [36], SANA-FE [37]). The main value is twofold. First, it enables functional verification: developers can run tests on compiled networks, checking spike statistics, firing rates, latency constraints, energy consumption, accuracy, and other KPIs, but without being affected by board availability or lab scheduling. Second, it supports portability and equivalence: a "golden" simulator becomes a flexible tool for AutoML loops and a common oracle against which further compiler optimizations, graph partitioning, and device-specific mapping can be validated.

H. Deployment

Once the model has been designed, optimized, and compiled, it is possible to deploy it on the chosen target HW through the implementation of custom partitioning and placement strategies to allocate neurons on the HW architecture [38], [39]. In phase, models can be packaged as immutable artifacts (such as SW modules and HW binaries that are made available in a containerized environment) with locked dependencies, recorded provenance, cryptographic hashes, and software bill of materials (SBOMs). Such a package also includes the necessary HW-dependent configuration files, encompassing interconnection settings, timestep definition, mismatch compensation parameters, mapping and routing configurations, and other related specifications. This is possible since all the previous steps have been performed in containerized environments orchestrated by Kubernetes. Then, one-click deployment templates bundle environment variables, configuration files, and drivers with a view to streamlining edge testing (such as on Jetson or MCUs targets). In this phase, the Slurm tool schedules the deploy with fair sharing, enforcing partitions and priority policies. Telemetry agents report latency, throughput, power, and spike-rate statistics to centralized dashboards with synchronized triggers and board identifiers. Rollouts follow development, staging, and production channels. Canary, shadow, or blue-green strategies are used, with automated rollback triggered on KPIs violations.

I. Operation and Monitoring

Due to the nature of neuromorphic HW, based on asynchronous operation, unpredictable spike distribution, and sparse behavior, with the added challenge that different vendors implement these properties in their own unique way, neuromorphic application deployments often exhibit nondeterminism and temporal drift [40]. NMLOps aim at closing the loop with data-drift detection (e.g., distributional tests on event rates and interevent intervals), canary tests, and adaptive retraining triggered by KPIs verifications.

Operations add day-two tasks that keep systems reliable at scale: 1) device and board health monitoring (temperature, voltage, current, and error counters); 2) firmware and SDK lifecycle management with staged rollouts and rollback; 3) configuration and feature-flag management for model variants; 4) fleet management for provisioning, access control, key and certificate rotation, and remote attestation; 5) observability for pipelines and networks (metrics, logs, and traces) with store-and-forward buffers for intermittent links; and 6) incident response runbooks tied to alerts and SLO breaches.

Reproducibility is enforced via artifact registries that version: 1) datasets and encoding parameters; 2) model, topology, and hyperparameters; and 3) HW-dependent compilation outputs and board identifiers. Governance includes signed containers, auditable pipelines, privacy-preserving data handling for multitenant use, and defined retention policies. SLO for energy and latency are monitored with automated remediation and safe rollback, and compliance logs capture model lineage for regulated deployments.

J. AutoML for Neuromorphic Pipelines

In consideration of the design parameters (encoders, time constants, topology, and precision), NMLOps proposes the adoption of AutoML tools [41], operating within explicit device constraints, to achieve concurrent optimization of accuracy, latency, and energy. AutoML is a SW framework that automates and optimizes the entire ML pipeline. This includes the following processes: input preprocessing, feature engineering, encoding strategies, model selection, and hyperparameter tuning. These processes are performed under explicit objectives, such as accuracy, latency, and energy. Additionally, AutoML incorporates device-aware, HW in the loop evaluation for deployment in edge and IoT environments. At this stage of the process, the utilization of real or simulated HW is possible.

Optimizers include multiobjective Bayesian optimization, simulated annealing and evolutionary strategies, and multifidelity schedulers with HW in the loop evaluation wherever

possible. The search space encompasses neuronal firing thresholds, leakage constants, synaptic and neuron state precision, structured and unstructured sparsity, and encoder architectures. The utilization of surrogate models, in conjunction with the concept of early stopping, serves to reduce the computational costs associated with evaluation. The process yields HW-aware models. Each candidate is measured on the intended device class and evaluated against predefined baselines. The final deliverable is a selected model with its encoding configuration, accompanied by deployment-ready artifacts and documented accuracy–latency–energy tradeoffs.

K. Storage, Versioning, and Registry

Different phases of the NMLOps process require repeatability and the ability to reuse previous versions of models or datasets. This is achieved by configuring storage through version control system (VCS) tools, such as Git, which allow tracking and organizing different versions of the files used in application development. In this way, both model engineers and AutoML tools can review the history of optimizations and modifications made throughout the project, identifying the best tradeoffs that satisfy the KPIs defined in the requirements phase. Furthermore, a registry is essential for keeping track of all requirements, development operations, and results obtained at each stage of the NMLOps pipeline.

While NMLOps is model-centric, it provides the practical foundation upon which broader engineering methodologies can be built for system-of-systems solutions that integrate neuromorphic and digital components.

Section IV instantiates this NMLOps blueprint in the inNuCE RI, detailing how Kubernetes and Slurm realize each phase for practical AIoT prototyping.

IV. Cloud-Based inNuCE-RI Architecture

Building on the NMLOps blueprint described in Section III, inNuCE RI instantiates the lifecycle stages in a two-pillar research infrastructure that couples a physical laboratory with a cloud-based heterogeneous prototyping platform. The design draws on virtual prototyping practices from FPGA [42] and HPC and on cloud-native orchestration exemplified by CrownLabs [43], an open-source, Kubernetes-based platform for browser-accessible remote labs. The unified testbed integrates commercial and open-source neuromorphic processors together with FPGAs, GPUs, MCUs, and sensors and exposes them through browser IDEs, templates, and a single job interface. The goal is to lower entry barriers, standardize workflows and interfaces, and accelerate the path from research prototypes to industrial AIoT deployments.

The inNuCE RI combines a physical laboratory with the inNuCE HPP to make neuromorphic AIoT prototyping reproducible and accessible. The architecture of the inNuCE HPP represented in Fig. 2 follows four principles.

1) *Idempotency* is achieved through containerized toolchains and versioned artifacts that allow exact experiment reruns.
2) *Heterogeneity* is embraced by supporting neuromorphic chips, FPGAs, GPUs, MCUs, and a range of sensors within one workflow.
3) *Scalability* is provided by Kubernetes, Slurm, and an optional federation model that accommodates growth in users, devices, and datasets.
4) *Accessibility* is delivered through browser IDEs, curated templates, and a unified job submission interface that reduces the learning curve for newcomers while remaining useful for experts.

The inNuCE HPP is organized in two logical layers that align with NMLOps phases. The *Layer 1*, dedicated to model development, provides the essential microservices required during the development stage. Personal user workspaces host web-accessible *development environments* (VS Code or Jupyter), bootstrapped from templates that provide projects with data pipelines, training scripts, and compilation recipes, together with frameworks, such as snnTorch, Lava, GeNN, NIR tools, NeuroBench, and many others provided as Python packages. Integration with Neural network intelligence (NNI) for AutoML enables experimental tracking, parameter sweeps, and automated runs [44]. Each environment is backed by private, user-specific storage to keep data secure, while shared, cluster-wide storage exposes datasets and a model zoo to further streamline development.

The *Layer 2* executes models on HW. Jobs are placed into a Slurm queue and run on target boards; metrics, artifacts, and logs return to the development layer and appear in dashboards. The configuration of remote execution backends is permitted for external tenants, such as CINECA, and customer clusters. As illustrated in Fig. 2, this logical architecture delineates the control and data paths from the developer workspace to the execution layer and vice versa.

The platform's *Layer 1* is built on Kubernetes, which provides core capabilities, such as authentication, role-based access control, user namespaces and quotas, persistent storage volumes, secrets management, and service routing. Through the *dashboard*, users interact with platform and the Kubernetes API to create *development environments*, work with datasets and model templates, and submit jobs via a consistent, unified interface. Cluster-wide resources (in yellow in Fig. 2) are made available to users to support and streamline development, including shared datasets and model templates.

Datasets are stored in an S3-compatible object repository (MinIO) with full versioning that records encoding parameters, calibration constants, and preprocessing steps. A model registry retains trained models, compiler outputs, and deployment bundles. Each of these items is identified by cryptographic hashes and accompanied by a SBOMs. The system's end-to-end provenance links data versions, code commits, container image digests, and HW identifiers for every run, enabling audit and exact reruns across devices and SW versions.

Security and isolation are enforced through the implementation of namespaces and role-based access control, which serve to compartmentalize users and projects. The protection of datasets and artifacts in storage, as well as the restriction of network traffic between services, is achieved through the use of encryption and network policies. The utilization of signed container images and attestation serves to ensure the integrity of the supply chain from the build phase to the deployment

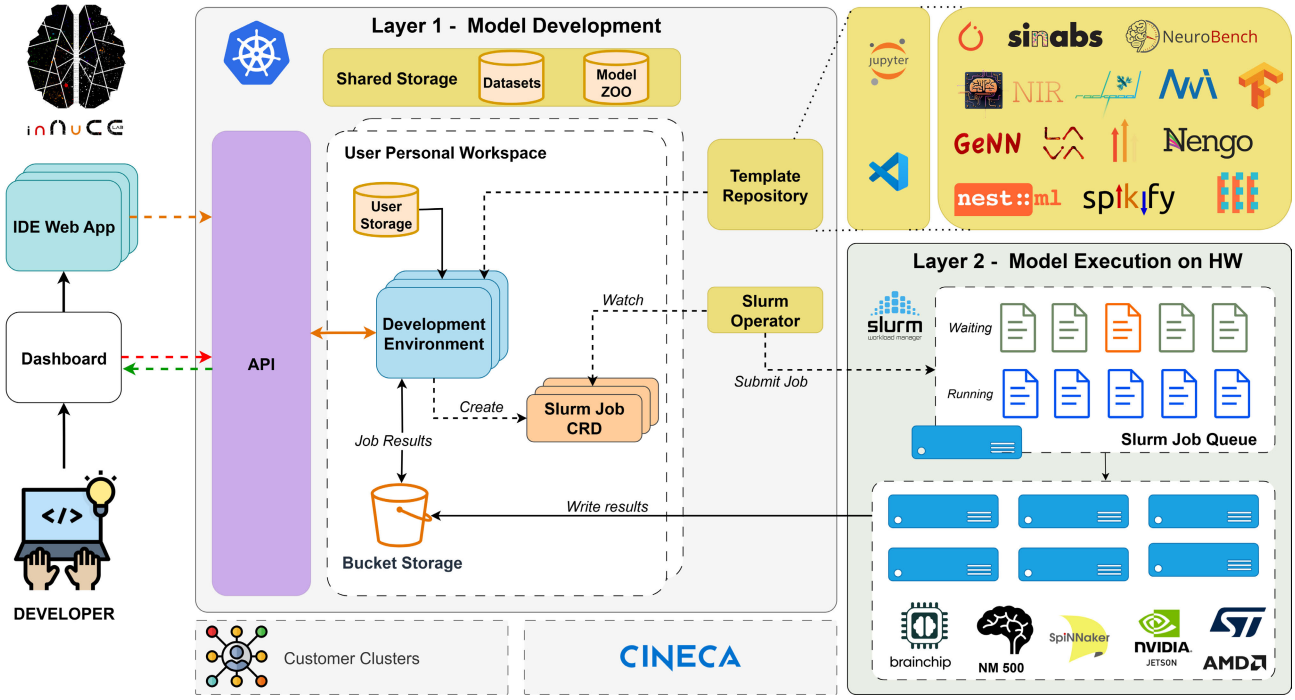

Fig. 2. inNuCE HPP two-layer infrastructure illustrating the service pipeline from development to Slurm-managed HW execution.

phase. The NMLOps *operation and monitoring* procedures are also implemented in this layer.

The platform's *Layer 2*, built on the Slurm framework, is responsible for managing the HW resources and orchestrates via queues the job execution. Slurm exposes GPUs, FPGAs, MCUs, and neuromorphic boards through the implementation of independent queues with partitions that enforce fairness and provide exclusive access when needed. Each job records standardized metrics (latency, throughput, memory, and power) to enable comparisons across boards. The results are archived with the corresponding code, configurations, and compilation artifacts in a personal *bucket storage* to ensure reproducibility. Usage is tracked on a per-user or per-project basis, with quotas and priority policies in place to prevent the undesirable effects of "noisy neighbors" in multitenant settings.

The HW pool currently includes BrainChip Akida-1000, General Vision NM500, SpiNNaker, NVIDIA Jetson devices, ARM-based MCUs, and AMD-Xilinx FPGAs. FPGAs can accommodate open neuromorphic designs (e.g., ReckOn-FPGA [45], Spiker+ [46], and SYNtzulu [47]) and provide a bridge when dedicated neuromorphic chips are scarce, enabling rapid iteration on architecture variants before custom silicon. New neuromorphic HW, such as SpiNNaker 2, Innatera SNP, and Neuronova H1, could be onboarded by registering the Slurm node and its containerized SDKs, which preserves a consistent user experience across targets.

inNuCE HPP offers a PaaS environment that provides containerized workspaces, shared storage, curated model templates, and a single execution layer that schedules jobs across heterogeneous boards. Users develop in browser-based IDEs, such as VS Code and Jupyter, launch pipelines, and submit jobs to Slurm across GPUs, FPGAs, MCUs, and neuromorphic HW. Results, logs, and telemetry flow back to dashboards and registries, enabling traceable and repeatable experiments and one-click packaging for edge pilots. Fig. 3 provides a synopsis of the physical deployment topology, emphasizing the interplay between the cluster and the laboratory.

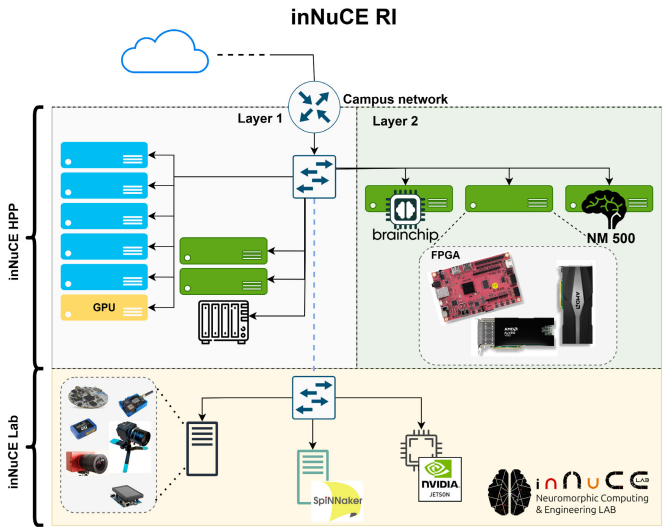

Fig. 3. inNuCE RI. In *Layer 1*, the inNuCE HPP compute cluster is present, comprising CPU, GPU, and storage nodes that are orchestrated with Kubernetes. In *Layer 2*, the neuromorphic boards, FPGAs, and Jetson boards are managed with Slurm. In the inNuCE Lab, the DVS, sensors, and development boards are available for physical access.

The inNuCE Lab is a facility equipped with event-driven sensors, standard sensor development kits, and neuromorphic or digital boards, providing a platform for conducting experiments under realistic conditions. The laboratory is equipped with synchronization tools that are based on trigger generators and power meters, the purpose of which is to measure

the board-level power consumption during the execution of deployed models. The laboratory is a complementary addition to the inNuCE HPP and is accessible to users who require physical interaction with sensors and real boards to implement specific parts of their use cases (e.g., ground-truth acquisition and on-device validation).

A typical session starts when a developer logs into the workspace with the associated *dashboard* that collects the different tools and data. Before any model design, the target platform and KPIs are fixed as part of the predesign requirements. The target board, latency and energy bounds, and the memory budget are specified, and these constraints determine the choice of datasets, spike encoding, and templates. A project template is then cloned, creating the *development environment*, packaged as a container running a browser-accessible IDE web app, where datasets are linked from shared storage, and spike encoding is configured to match the selected device. Training runs either inside the *development environment* for CPU/GPU execution or offloaded to HW available in *Layer 2* nodes by submitting a Slurm job. In the latter case, the user creates a *Slurm Job CRD*, which triggers the platform logic to submit the job to *Layer 2* via the *Slurm Operator*. Results, such as logs and generated files, flow back to a personal bucket storage, accessible through the *dashboard* or the *development environment*, where the user can compare accuracy, latency, memory, and energy against baselines and acceptance thresholds.

The implementation of cloud platforms federation is envisaged as a future capability. A range of strategies for interoperating with external sites (EBRAINS, CINECA, and THOR) will be evaluated and adopted to enable access to and use of remote resources on partner clusters.

Access to inNuCE RI will be offered via: 1) pay-per-use credits redeemable for HW time, expert consulting, or lab days; 2) collaboration agreements under funded projects; 3) federated research-infrastructure programs (e.g., EBRAINS); and 4) teaching allocations for courses taught by inNuCE RI associated members. All users are subject to identity verification, quotas, and fair-scheduling policies.

Compared with cloud tools that target digital accelerators (e.g., Edge Impulse, SensiML, STM32 Dev Cloud, and Arduino Cloud) and with platforms tied to a single neuromorphic ecosystem (e.g., Loihi/vLab, SpiNNaker/EBRAINS, and Akida Cloud), inNuCE RI unifies commercial and open neuromorphic HW alongside FPGAs, GPUs, and MCUs within a single NMLOps-aware workflow. This integration reduces vendor lock-in, enables comparable cross-target evaluation, and shortens the path from prototype to deployable AIoT systems.

V. Use Cases Implemented on the inNuCE RI

This section will build on the inNuCE RI architecture described in Section IV, illustrating how the NMLOps lifecycle is instantiated in practice across multiple applications. The following three user scenarios are presented to illustrate typical entry points and execution paths through the platform, as depicted in Fig. 4. We subsequently provide a concise explanation of how these scenarios correspond to the NMLOps stages. In conclusion, we present the comprehensive set of implemented use cases with harmonized descriptions that accentuate the role of NMLOps in each case.

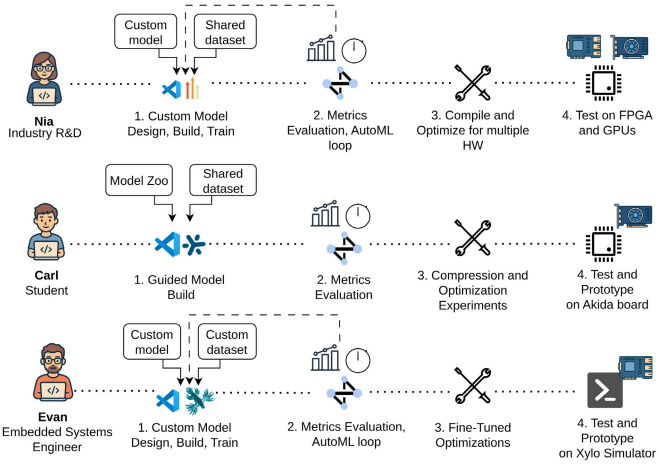

Fig. 4. Three representative user scenarios on inNuCE RI. Top row: industry R&D engineer with a custom model and shared dataset. Middle row: student with a model-zoo template and shared dataset. Bottom row: embedded systems engineer with a custom model and custom dataset. Each scenario follows the NMLOps stages from predesign targets to evaluation, compilation, and prototype execution on HW or simulators.

In Fig. 4, Nia, an industry research and development engineer, commences her research from a custom model and a shared dataset. The metrics are evaluated and a small AutoML loop is launched. The model is then compiled and optimized for multiple HW targets before the prototype is validated on FPGAs and GPUs. Carl, a student, starts his project utilizing a model-zoo template and a shared dataset. Metrics evaluation, compression and optimization experiments, and prototyping on an Akida board are all part of the process. Evan, an embedded systems engineer, starts his research by utilizing both a custom model and a proprietary dataset. The metrics are evaluated using an AutoML loop, fine-tuned optimizations are applied, and prototypes are created on the Xylo simulator. Collectively, these scenarios encompass the prevalent pathways traversed within inNuCE RI, ranging from template-driven experiments to industrial-grade, multitarget optimization.

Each scenario begins with a predesign step that fixes the target platform, resource budgets, and KPIs before any model work. Nia targets multiple devices from the outset, so her workflow emphasizes early conversion and compilation to assess feasibility across boards. Carl follows a guided path: the platform provides a template and a reference dataset, and his effort centers on compression and quantization to meet edge constraints. Evan brings a custom dataset and model, which requires HW-aware training, periodic fit checks against the selected device, and simulator-based validation before moving to a physical board. Across all scenarios, data and encoding are versioned in the object store. Training runs in containerized workspaces. Evaluation reports accuracy, latency, memory, and energy. Conversion and compilation yield device-specific artifacts. Deployment and monitoring are automated by Slurm and Kubernetes. The registry enforces provenance and governance.

The following sections apply this NMLOps pattern to the implemented use cases on inNuCE RI. Each case is organized into *objective*, *NMLOps* instantiation, and *outcome and evaluated KPIs*.

A. HAR With Neuromorphic State-Space Model

Objective—Classify daily activities from inertial measurements on edge devices under tight memory and latency budgets [48], [49]. *NMLOps*—The target device and performance metric, i.e., classification accuracy, are selected during the requirement definition. Raw inertial measurement unit (IMU) streams are processed with spike-encoding parameters. Training is performed in containerized snnTorch notebooks on GPU nodes with tracked hyperparameter sweeps over various neuronal and architectural configurations through an AutoML approach. Compression, conversion, and compilation produce quantized weights and a runtime container stored in the registry. Deployment is performed through one-click templates that emit edge bundles for Raspberry Pi testbeds whose access is scheduled to Slurm. Monitoring agents export inference time and power to dashboards for drift detection. The evaluation reports accuracy, latency, memory footprint, and energy, using the same profilers on both simulation and HW. All data, code, and artifacts are signed and archived for exact reruns.

Outcome and evaluated KPIs—Real-time activity classification on GPUs and embedded targets with reduced memory footprint. Accuracy, on Raspberry Pi, from 91.86% to 95.97% depending on the compression. Inference time, on Raspberry Pi, from 174 to 179 ms depending on the compression. Static and workload metrics evaluated through NeuroBench [35].

B. Braille Letter Reading on Digital and Neuromorphic Boards

Objective—Recognize Braille characters through their spatiotemporal tactile patterns for assistive interfaces at low-energy costs. *NMLOps*—Requirements include low energy consumption, real-time performance, and reproducibility. Pressure signals acquired through an artificial fingertip can be either spike-encoded to fed a neuromorphic platform [50] or sent as raw data to conventional digital HW [51]. In the former case, tactile frames are transformed into event tensors and used in containerized snnTorch and Intel PyTorch2Loihi pipelines to build sparsity-regularized SNNs. For digital platforms, instead, the toolchain produces ONNX models for deployment on edge devices. HW in the loop tests can run on Jetson and Loihi boards scheduled by Slurm, while power meters and GPIO triggers measure energy, latency, and the effects of compression and optimization. Compiling procedure registers device-specific binaries, drivers, and host containers as immutable artifacts. Monitoring schedules periodic checks to detect regressions, and governance tracks dataset licenses and artifact signatures. *Outcome & evaluated KPIs*—Real-time Braille letter reading on low-power devices. Accuracy: 87.78% on microprocessor unit (MPU); from 45.1% to 98.2% on GPU depending on the architecture and the encoding scheme; from 40.1% to 80.9% on Loihi depending on the SNN characteristics and the encoding scheme. Inference time: 264 ms on MPU; from 6.7 to 492.5 ms on GPU; from 2.1 to 5.4 ms on Loihi. Energy per inference: 313 mJ computed from nominal power consumption for MPU; from ~40 to ~1760 mJ on GPU; from ~0.04 to ~0.18 mJ on Loihi.

C. Positioning and Tracking With a Neuromorphic and Sensory Fusion

Objective—Maintain accurate localization under degraded GPS using dead-reckoning fused with intermittent positioning updates [52]. *NMLOps*—HW target and memory budget are defined in the requirements as an embedded GPU with model size under 0.1 MB. Multisensor data from the IMU, wheel encoder, and optical gyroscope are collected and synchronized. The raw signals are then encoded into spike trains through a preprocessing stage implemented by populations of leaky integrate-and-fire (LIF) neurons. A model with a LIF-based Legendre memory unit (LMU) and an extended kalman filter is built. The model defined on snnTorch is trained using AutoML jobs on GPU nodes that search on the model's hyperparameter space. Evaluation measures trajectory error, drift, latency, and energy. Compression generates a pruned version of the model. Conversion and compilation translate the model to the specific target's framework and generate the runtime binaries. The model is deployed on the target and field tests stream position error, runtime, and power to dashboards. Operation and monitoring rules trigger retraining when metrics diverge, and all artifacts and logs are archived for deterministic rollback. Finally, the best model is exported into the model zoo for future usage. *Outcome and evaluated KPIs*—Comparable accuracy to SOTA deep learning methods, with up to one order-of-magnitude reduction in memory (an absolute trajectory error of 1.14 m, with a model size of 0.05 MB).

D. Event-Based Visual Gesture Recognition on Akida-1000 and SynSense Speck

Objective—Perform real-time recognition of rock–scissor–paper gestures using event streams acquired from various DVS cameras, leveraging the asynchronous nature of events for efficient inference (task inspired by [53]). *NMLOps*—The two target HW platforms, Akida-1000 and Speck, and the DVS cameras (DAVIS, Explorer, and Speck) are fixed in the requirements, together with the maximum latency required for real-time operation. Data collection records events from the DVS cameras, that is synchronized and version-controlled in a centralized repository, enabling reproducible experiments and consistent dataset management. During model building and training, a lightweight convolutional neural network and a spiking convolutional neural network are designed for gesture recognition and optimized for event driven computation. Each model is developed in a containerized environment, employing the respective SDKs for each HW platform. Models training, fine-tuning on specific camera inputs, and quantization to adapt weights to the HW requirements are performed using AutoML jobs on GPU nodes. Deployment on AKIDA-1000 is

scheduled through Slurm. Evaluation metrics include latency, power consumption, and energy per inference. Deployment on Speck is performed in the inNuCE Lab, and latency is used as evaluation metric. In both cases, metrics are collected via on-board APIs and systematically logged for traceability across runs. Finally, the best model is rolled out in the model zoo for deployment and reproducibility purposes. *Outcome*—The deployed models achieved 91.5% accuracy on Speck and 93.2% on Akida, with performance comparable to the CPU reference while substantially reducing average power consumption. In particular, the Akida deployment achieves an average latency of 0.29 ms per sample with an energy cost of 0.25 mJ per inference, compared to 0.06 ms per sample on a CPU baseline but at a significantly higher energy cost of 1.22 mJ per inference, while the NVIDIA RTX A4000 achieves a latency per sample of 0.004 ms but with a 0.29 mJ per sample. The classification accuracy remains comparable across platforms.

E. Constraint Satisfaction Problems on SpiNNaker and GeNN

Objective—Solving Sudoku a constraint satisfaction problem via fully spiking networks, minimizing time to solution and spike traffic between SpiNNaker and host computer [54]. *NMLOps*—The pipeline design is explored on the GPUs by using GeNN to optimize the solver capacity of the network. The PyNN containerized architecture and sPyNNaker facilitate the deployment of binaries on the SpiNNaker board through the Slurm orchestration system. The deployment has also been instantiated on the remote SpiNNaker system hosted in the EBRAINS research infrastructure. Telemetry collects spike traffic, convergence time, and success rate. A built-in network validates solutions. Successful configurations are promoted to the model zoo with templates and parameters for one-click re-execution. The SNNs developed for the two target frameworks have been versioned on GitHub and indexed on the EBRAINS knowledge graph. *Outcome & evaluated KPIs*—Significantly reductions in spike traffic up to 99.98% and solution extraction time more than 96% compared with previous SNN solvers.

F. Hi-Co Network on Loihi 2

Objective—To port a biologically inspired hippocampal-cortical (Hi-Co) SNN for semantization of memories to neuromorphic HW and to assess the feasibility of using this network for online learning and inference at the edge. The network was originally implemented in the Brian simulator [55], and the chosen target is the neuromorphic framework Lava and the Loihi 2 neuromorphic chip. *NMLOps*—The target framework Lava and target device Loihi 2 are fixed as requirements. During model building, two containers with fixed dependencies are prepared: one for the original Brian implementation and another for the development of the Lava counterpart. The model is first ported into Lava-CPU and trained using CPU nodes, considering all the architectural details and constraints that arise when writing a model for a specific HW platform and that are not present when writing the model for a simulation. Evaluation checks accuracy and footprint and compares the correctness using the Brian implementation as golden sample. Compression and compilation are performed using Lava-Loihi, taking into account the Loihi 2 HW constraints, such as fixed-point arithmetic, quantized parameters, and assembly-based neuron models. Deployment on Loihi 2 is performed via Intel's vLab service, and evaluation metrics, such as accuracy, latency, and energy consumption, are obtained. All artifacts of the three implementations are stored and versioned for future reruns. *Outcome and evaluated KPIs* The Lava-CPU implementation reproduces the behavior of the original model, reaching an accuracy of 82%, close to the 85% achieved by the Brian implementation.

G. Neuromorphic Pipeline on FPGA

Objective—Evaluate the performance of a digitally implemented SNN on an FPGA with time-varying sensor data using open-source neuromorphic accelerators, and enable rapid co-design of algorithms and HW under tight latency and energy budgets [45]. *NMLOps*—The FPGA targets (PYNQ-Z2 and ZCU102) and neuromorphic accelerator (ReckOn) [56] are fixed in the requirements. Spike encoding is performed with configurable, HW optimized Izhikevich neurons that can be instantiated to perform real-time operation of raw sensor streams. During model building, the ReckOn-based architecture is imported and customized. Compilation is performed using containerized toolchains that generate the customized system bitstream. For deployment, the FPGA processing system (PS) is exposed by Slurm to monitor the evolution of the experiments. Slurm additionally schedules synthesis, place-and-route, and runtime jobs. Training is performed on the HW using the e-prop algorithm supported by ReckOn. Multiple independent ReckOn instances allow parallel runs to accelerate training and parameters exploration. Additional synthesis and implementation strategies can be selected to reach desired timing constraints, resource utilization, or power consumption targets. Golden-set tests verify functional equivalence across revisions. Artifacts, including hardware design language (HDL) sources, constraints, bitstreams, PS images, and host containers, are versioned and archived in a GitLab repository. *Outcome and evaluated KPIs*—FPGA-based neuromorphic pipelines characterized by on-device encoding and parallel experimentation accelerate the prototyping of solutions. These features have been demonstrated to have several benefits, including the potential to shorten design cycles and expand the design space for a given task. Recorded metrics include the overall experiment accuracy and design-specific resource utilization (lookup tables (LUTs), block RAM (BRAM), and digital signal processing (DSP) blocks), board-level power, and compliance to timing constraints, reported in the output logs generated by electronic design automation (EDA) tools.

Taken together, these use cases show how inNuCE RI implements NMLOps end to end. Targets and KPIs are fixed up front, data and models are versioned, compilation produces device-specific artifacts for heterogeneous HW, and deployment and monitoring keep results reproducible and portable across devices. Table I summarizes how the NMLOps steps are implemented for the presented use cases, within the inNuCE RI.

TABLE I

OVERVIEW OF NMLOPS STEPS (ROWS) AND THEIR IMPLEMENTATION FOR SEVEN EXAMPLE USE CASES (COLUMNS), WITH THE CORRESPONDING INNUCE RI LAYER SHOWN ON THE RIGHT. HATCHED CELLS INDICATE STEPS PERFORMED OUTSIDE THE INNUCE RI

			HAR with neuromorphic state space model	Braille reading on digital and neuromorphic HW			Positioning and tracking with neuromorphic sensory fusion		Event-based visual gesture recognition on neuromorphic HW		Constraint Satisfaction Problems on neuromorphic HW		Bio-inspired network on neuromorphic HW	Neuromorphic HW pipeline on FPGA
L1 - Model Development	1. Requirements	HW	Embedded Computer	Loihi	Embedded GPU	MPU	Embedded GPU		Akida 1000	SynSense Speck	SpiNNaker		Loihi 2	ReckOn on FPGA
		FW	snnTorch	PyTorch		snnTorch	snnTorch		MetaTF	Sinabs	sPyNNaker		LAVA	EDA tools
		KPI	Accuracy	Accuracy			Accuracy		Accuracy		Solution rate Time to solution		Accuracy	Accuracy
		Cons	Memory	Real-time			Memory		Real-time		On-device solution		Bio-plausibility	Timing, resources and power
	2. Data	Acqn	WISDM public dataset	Pressure sensor			Wheel encoder, IMU and optical gyroscope		DVS cameras		Sudoku public datasets		MNIST public dataset	Time series datasets
		Pre-Proc	Windowing				Synchronization and windowing		Accumulate events into frame	Windowing			Deskewing Binarization	
		Spike Encoding	Optimized LIF neurons	Sigma-Delta Encoding		Opt. LIF neurons	Optimized LIF neurons						Poisson Encoding	HW-optimized Izhikevich neurons
	3. Model	Design	L^2MU	Fully connected LIF-based SNN			L^2MU	L^2MU with extended Kalman filter	CNN defined in TensorFlow	Spiking CNN defined in Sinabs	SNN based on neuron populations implementing custom operations		Bio-inspired SNN written in Brian2 and ported to LAVA	Recurrent LIF-based SNN
		Training	snnTorch on GPU nodes	PyTorch on GPU nodes		snnTorch on GPU nodes	snnTorch on GPU nodes		TensorFlow on GPU nodes	PyTorch on GPU nodes	Expert-in-the-loop using simulation		LAVA floating-point on CPU nodes	Online with e-prop on ReckOn
	4. Evaluation & Resource Estimation		Accuracy Neurobench KPIs	Accuracy			Accuracy Neurobench KPIs		Accuracy		Solution rate Time to solution Spike traffic		Comparison between Brian2 and LAVA floating-point behaviour	FPGA resource utilization
	5. Compression & Optimization		PyTorch	PyTorch2-Loihi		Neuron's state quant. with snnTorch			QAT with MetaTF-quantize	PyTorch	QAT with sPyNNaker		QAT with LAVA fixed-point	Custom synthesis and implementation optimization strategies
	6. Conversion, Compilation & Portability		TorchScript	PyTorch2-Loihi	PyTorch	ONNX	snnTorch		MetaTF-cnn2snn	Samna	sPyNNaker		LAVA and Loihi 2's BSP	EDA tools
	7. Simulation		snnTorch	PyTorch		snnTorch	snnTorch			Sinabs	GeNN		LAVA fixed-point	
L2 - HW Execution	8. Deployment	Tool	TorchScript	Loihi's BSP	ONNX	PyTorch	PyTorch		AkidaRuntime	Samna	sPyNNaker		Loihi 2's BSP	EDA tools
		Device	Raspberry Pi 4B	Loihi	Jetson Xavier NX	STM32 MP1	Jetson Xavier NX		Akida 1000	Speck	SpiNNaker		Loihi 2	PYNQ-Z2 ZCU102
		Access	L2's Slurm	Intel's Server	L2's Slurm		L2's Slurm		L2's Slurm	inNuCE Lab	L2's Slurm	EBRAINS	Intel's Server	L2's Slurm
L1 - Model Development	9. Operation & Monitoring		Accuracy Latency Energy Memory footprint	Accuracy Latency Energy			Accuracy Latency Energy Memory footprint		Accuracy Latency Energy		Solution rate Time to solution Spike traffic		Accuracy Latency Energy	Accuracy
	10. AutoML		NNI for hyperparameter exploration	NNI for hyperparameter exploration			NNI for hyperparameter exploration				NNI with HW-in-the-loop for QAT exploration			NNI with HW-in-the-loop for hyperparameter exploration
	11. Storage, Versioning & Registry		GitHub	GitHub			L1's GitLab		L1's GitLab		GitHub	EBRAINS Knowledge Graph	L1's GitLab	L1's GitLab

The following discussion summarizes the considerations observed across the platform while implementing these use cases in inNuCE RI. These considerations include scalability and multitenancy, reproducibility and standardization, security,

privacy and compliance, threats to validity, and the evolution from NMLOps toward a broader neuromorphic engineering process (NEP).

1) Scalability and Multitenancy: Kubernetes and Slurm offer elastic scaling and fair scheduling for shared clusters. Registering a Slurm node and supplying a containerized SDK when on-boarding new boards preserves a uniform workflow across targets. Where appropriate, strategies for federating with research infrastructures and HPC sites will be evaluated and, if feasible, implemented to enable the sharing of HW access while enforcing policies relating to identity, quotas, and priorities to ensure fairness in multitenant settings.

2) Reproducibility and Standardization: Reproducibility is achieved through version-pinned containers and an immutable artifact registry which stores data, models, simulation results, AutoML logs, compiler outputs, and deployment bundles alongside cryptographic hashes and SW bills of materials. Standardization efforts focus on portable intermediate representations and common APIs that reduce vendor lock-in and enable cross-target portability. Benchmarking using suites, such as NeuroBench, allows for comparable evaluations across modalities and HW.

3) Security, Privacy, and Compliance: Operating in a multitenant environment introduces risks, such as data leakage, model theft, and side-channel exposure. Mitigation strategies include namespace isolation, role-based access control, per-job secrets, encrypted volumes, signed containers, and auditable pipelines. For sensitive deployments, on-premises execution and confidential compute extensions can strengthen isolation and compliance.

4) Threats to Validity: In neuromorphic sensing, dataset bias and annotation noise can be amplified by event sparsity. Mitigating these risks involves diversifying data sources and, where possible, conducting cross-site evaluations. However, nondeterminism in spiking models can hinder strict reproducibility. Therefore, we log random seeds, encoding parameters, compilation artifacts, and board identifiers, and we use repeated trials and golden-set tests to verify functional equivalence after conversion.

5) From NMLOps to NEP: Although NMLOps governs the lifecycle of neuromorphic models on the platform, the industrial deployment of these models on a large scale requires a broader NEP. The NEP incorporates NMLOps into service-oriented workflows, incorporating stakeholder roles, verification plans, and lifecycle governance across systems of systems [57]. A full treatment of the NEP is beyond the scope of this article and will be addressed in future work.

VI. Conclusion

This article presents the NMLOps process, an evolution of MLOps for the integration of neuromorphic technology in AIoT applications, and the inNuCE RI, a research infrastructure that operationalizes NMLOps on a cloud-native, heterogeneous prototyping platform that is tightly coupled with a physical laboratory. This enables end-to-end, reproducible prototyping and benchmarking across neuromorphic and digital substrates, as well as the validation of representative on-edge AIoT applications, such as HAR, Braille reading, navigation tracking, and constraint satisfaction problems. By consolidating toolchains, orchestrating heterogeneous HW with Kubernetes and Slurm and enforcing rigorous versioning of data, models, and artifacts, inNuCE RI lowers adoption barriers and shortens the path from prototype to engineered system. The utilization of standard storage and versioning systems, in conjunction with the complete accessibility of data and tools within containerized environments, ensures the adherence of the service to the findable, accessible, interoperable, reusable (FAIR) principles. This facilitates the utilization of the platform for scientific studies, where the rigorous management of data is crucial. More broadly, virtual prototyping platforms, such as inNuCE RI, offer a low-cost, low-risk environment in which to explore designs and deployment options before committing to HW. Beyond neuromorphic workflows, the infrastructure also supports AIoT use cases that target standard and emerging digital technologies (CPU/GPU/TPU, MCU/TinyML, FPGAs, and other accelerators) through the same NMLOps procedures. This reduces development costs, shortens time-to-market, and broadens the scope of feasible heterogeneous AIoT use cases.

Our major achievement is turning NMLOps from a conceptual adaptation of MLOps into an operational practice, with browser workspaces, a multiboard execution backend, and harmonized evaluation, making cross-target comparisons and iteration routine. In summary, a cloud-native, NMLOps-driven prototyping infrastructure is an effective catalyst for neuromorphic AIoT, as it preserves reproducibility, reduces risk and cost, and makes heterogeneous HW usable for real applications. Future work will evaluate and, where appropriate, implement federation with other complementary research infrastructures, as well as formalizing a broader NEP that embeds NMLOps into service-oriented workflows for multistakeholder system integration.

References

[1] G. Urgese, A. Rios-Navarro, A. Linares-Barranco, T. C. Stewart, and K. Michmizos, "Editorial: Powering the next-generation IoT applications: New tools and emerging technologies for the development of neuromorphic system of systems," *Frontiers Neurosci.*, vol. 17, May 2023, Art. no. 1197918.

[2] C. Mayr, S. Hoeppner, and S. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning-keynote presentation," in *Communicating Process Architectures*. Amsterdam, The Netherlands: IOS Press, 2019, pp. 277–280.

[3] M. Davies et al., "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proc. IEEE*, vol. 109, no. 5, pp. 911–934, May 2021.

[4] B. Posey, "What is the akida event domain neural processor? 2020," BrainChip, Sydney, NSW, Australia, Tech. Rep. 3, 2022. [Online]. Available: https://brainchip.com/wp-content/uploads/2020/03/BrainChip_tech-brief_What-is-Akida_v3-1.pdf

[5] A. Neckar et al., "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proc. IEEE*, vol. 107, no. 1, pp. 144–164, Jan. 2019.

[6] J. Büchel, D. Zendrikov, S. Solinas, G. Indiveri, and D. R. Muir, "Supervised training of spiking neural networks for robust deployment on mixed-signal neuromorphic processors," *Sci. Rep.*, vol. 11, no. 1, p. 23376, Dec. 2021.

[7] G. Gallego et al., "Event-based vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 154–180, Jan. 2022.

[8] Q. Chen et al., "DeltaKWS: A 65nm 36nJ/decision bio-inspired temporal-sparsity-aware digital keyword spotting IC with 0.6 V near-threshold SRAM," *IEEE Trans. Circuits Syst. for Artif. Intell.*, vol. 2, no. 1, pp. 79–87, Mar. 2025.

[9] D. Kudithipudi et al., "Neuromorphic computing at scale," *Nature*, vol. 637, no. 8047, pp. 801–812, 2025.

[10] D. R. Muir and S. Sheik, "The road to commercial success for neuromorphic technologies," *Nature Commun.*, vol. 16, no. 1, p. 3586, Apr. 2025.

[11] V. J. Reddi et al., "Edge impulse: An mlops platform for tiny machine learning," in *Proc. 6th Conf. Mach. Learn. Syst. (MLSys)*, Miami, FL, USA, vol. 5, Jun. 2023, pp. 254–268.

[12] J. E. Pedersen et al., "Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing," *Nature Commun.*, vol. 15, no. 1, p. 8122, Sep. 2024.

[13] C. Mead, "Neuromorphic electronic systems," *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. [Online]. Available: http://ieeexplore.ieee.org/document/58356/

[14] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[15] W. Zhang, S. Ma, X. Ji, X. Liu, Y. Cong, and L. Shi, "The development of general-purpose brain-inspired computing," *Nature Electron.*, vol. 7, no. 11, pp. 954–965, Nov. 2024.

[16] G. Li et al., "Brain-inspired computing: A systematic survey and future trends," *Proc. IEEE*, vol. 112, no. 6, pp. 544–584, Jun. 2024.

[17] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-based Neuromorphic Systems*. Hoboken, NJ, USA: Wiley, 2014.

[18] S. Höppner et al., "The SpiNNaker 2 processing element architecture for hybrid digital neuromorphic computing," 2021, *arXiv:2103.08392*.

[19] T. Liu, G. Brayshaw, A. Li, X. X. Xu, and B. Ward-Cherrier, "Neuromorphic touch for robotics—A review," *Neuromorphic Comput. Eng.*, vol. 5, no. 3, 2025, Art. no. 032001.

[20] N. Dennler, A. True, A. van Schaik, and M. Schmuker, "Neuromorphic principles for machine olfaction," *Neuromorphic Comput. Eng.*, vol. 5, no. 2, Jun. 2025, Art. no. 023001.

[21] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608097000117

[22] Q. Guo, F. Tang, T. K. Rodrigues, and N. Kato, "Five disruptive technologies in 6G to support digital twin networks," *IEEE Wireless Commun.*, vol. 31, no. 1, pp. 149–155, Feb. 2024.

[23] H. Sabbella et al., "The promise of spiking neural networks for ubiquitous computing: A survey and new perspectives," 2025, *arXiv:2506.01737*.

[24] M. Kato, T. Koketsu Rodrigues, T. Abe, and T. Suganuma, "Exploiting radio frequency characteristics with a support unmanned aerial vehicle to improve wireless sensor location estimation accuracy," *IEEE Internet Things J.*, vol. 11, no. 24, pp. 39570–39578, Dec. 2024.

[25] S. Chakraborty, S. Snyder, M. Abdullah-Al Kaiser, M. Parsa, G. Schwartz, and A. R. Jaiswal, "A retina-inspired pathway to real-time motion prediction inside image sensors for extreme-edge intelligence," *Neuromorphic Comput. Eng.*, vol. 5, no. 3, Sep. 2025, Art. no. 034005.

[26] S. S. Chowdhury, D. Sharma, A. Kosta, and K. Roy, "Neuromorphic computing for robotic vision: Algorithms to hardware advances," *Commun. Eng.*, vol. 4, no. 1, p. 152, Aug. 2025.

[27] L. Faubel, K. Schmid, and H. Eichelberger, "MLOps challenges in industry 4.0," *Social Netw. Comput. Sci.*, vol. 4, no. 6, p. 828, Oct. 2023.

[28] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "Tiny-MLOps: A framework for orchestrating ML applications at the far edge of IoT systems," in *Proc. IEEE Int. Conf. Evolving Adapt. Intell. Syst. (EAIS)*, May 2022, pp. 1–8.

[29] S. Zhong et al., "Recent advances in artificial sensory neurons: Biological fundamentals, devices, applications, and challenges," *Nano-Micro Lett.*, vol. 17, no. 1, p. 61, Dec. 2025.

[30] E. Forno, V. Fra, R. Pignari, E. Macii, and G. Urgese, "Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task," *Frontiers Neurosci.*, vol. 16, Dec. 2022, Art. no. 999029.

[31] E. Grappolini and A. Subramoney, "Beyond weights: Deep learning in spiking neural networks with pure synaptic-delay training," in *Proc. Int. Conf. Neuromorphic Syst.*, Aug. 2023, pp. 1–4.

[32] E. Izhikevich, "Spiking manifesto," 2025, *arXiv:2512.11843*.

[33] J. K. Eshraghian et al., "Training spiking neural networks using lessons from deep learning," *Proc. IEEE*, vol. 111, no. 9, pp. 1016–1054, Sep. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10242251/

[34] C. S. Thakur et al., "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers Neurosci.*, vol. 12, p. 891, Dec. 2018.

[35] J. Yik et al., "The neurobench framework for benchmarking neuromorphic computing algorithms and systems," *Nature Commun.*, vol. 16, no. 1, p. 1545, 2025.

[36] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, p. 47314, Aug. 2019.

[37] J. A. Boyle, M. Plagge, S. G. Cardwell, F. S. Chance, and A. Gerstlauer, "SANA-FE: Simulating advanced neuromorphic architectures for fast exploration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 8, pp. 3165–3178, Aug. 2025.

[38] G. Urgese, F. Barchi, E. Macii, and A. Acquaviva, "Optimizing network traffic for spiking neural network simulations on densely interconnected many-core neuromorphic platforms," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 3, pp. 317–329, Jul. 2018.

[39] M. Kato, T. K. Rodrigues, and N. Kato, "Breakout local search solution to the offloading decision problem in a multi-access edge computing cloud-enabled network," *IEEE Trans. Emerg. Topics Comput.*, vol. 13, no. 3, pp. 1328–1338, Jul. 2025.

[40] J. B. Aimone, "Neuromorphic computing: A theoretical framework for time, space, and energy scaling," 2025, *arXiv:2507.17886*.

[41] M. Baratchi et al., "Automated machine learning: Past, present and future," *Artif. Intell. Rev.*, vol. 57, no. 5, p. 122, Apr. 2024.

[42] S. Werner, A. Lauber, M. Koedam, J. Becker, E. Sax, and K. Goossens, "Cloud-based design and virtual prototyping environment for embedded systems," *Int. J. Online Biomed. Eng. (iJOE)*, vol. 12, no. 9, pp. 52–60, Sep. 2016.

[43] M. Iorio, A. Palesandro, and F. Risso, "CrownLabs—A collaborative environment to deliver remote computing laboratories," *IEEE Access*, vol. 8, pp. 126428–126442, 2020.

[44] V. Fra, "Application-oriented automatic hyperparameter optimization for spiking neural network prototyping," 2025, *arXiv:2502.12172*.

[45] M. Barocci, V. Fra, E. Macii, and G. Urgese, "Heterogeneous SoC integrating an open-source recurrent SNN accelerator for neuromorphic edge computing on FPGA," *Commun. Comput. Inf. Sci.*, pp. 1–16, Sep. 2025.

[46] A. Carpegna, A. Savino, and S. D. Carlo, "Spiker+: A framework for the generation of efficient spiking neural networks FPGA accelerators for inference at the edge," *IEEE Trans. Emerg. Topics Comput.*, vol. 13, no. 3, pp. 784–798, Jul. 2025.

[47] G. Leone, M. A. Scrugli, L. Badas, L. Martis, L. Raffo, and P. Meloni, "SYNtzulu: A tiny RISC-V-controlled SNN processor for real-time sensor data analysis on low-power FPGAs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 72, no. 2, pp. 790–801, Feb. 2025.

[48] V. Fra, E. Forno, R. Pignari, T. C. Stewart, E. Macii, and G. Urgese, "Human activity recognition: Suitability of a neuromorphic approach for on-edge AIoT applications," *Neuromorphic Comput. Eng.*, vol. 2, no. 1, Mar. 2022, Art. no. 014006.

[49] B. Leto, G. Urgese, E. Macii, and V. Fra, "Variable-precision neuromorphic state space model for on-edge activity classification," *Future Gener. Comput. Syst.*, vol. 176, Mar. 2026, Art. no. 108193.

[50] S. F. Müller-Cleve et al., "Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware," *Frontiers Neurosci.*, vol. 16, Nov. 2022, Art. no. 951164.

[51] V. Fra, A. Pignata, R. Pignari, E. Macii, and G. Urgese, "Neu-brauer: A neuromorphic Braille letters audio-reader for commercial edge devices," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, 2023, pp. 51–60.

[52] S. Tilocca, V. Fra, A. Pignata, E. Macii, and G. Urgese, "Positioning and tracking enhancement through a spiking legendre memory unit," *IFAC-PapersOnLine*, vol. 59, no. 26, pp. 235–240, 2025.

[53] X. Xie, S. Zhang, J. Wu, X. Xu, G. Shi, and J. Chen, "A real-time rock-paper-scissor hand gesture recognition system based on FlowNet and event camera," in *Proc. Chin. Conf. Pattern Recognit. Comput. Vis. (PRCV)*, 2019, pp. 98–109.

[54] R. Pignari, V. Fra, E. Macii, and G. Urgese, "Efficient solution validation of constraint satisfaction problems on neuromorphic hardware: The case of sudoku puzzles," *IEEE Trans. Artif. Intell.*, vol. 6, no. 8, pp. 2061–2072, Aug. 2025.

[55] F. D'Alba, N. Kushawaha, L. Fruzzetti, P. S. Paolucci, and E. Falotico, "Semantization of memories in a hippocampal–cortical spiking neural network," *Neurocomputing*, vol. 640, Aug. 2025, Art. no. 130323.

[56] C. Frenkel and G. Indiveri, "ReckOn: A 28nm sub-mm^2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[57] G. Urgese, P. Azzoni, J. van Deventer, J. Delsing, A. Macii, and E. Macii, "A SOA-based engineering process model for the life cycle management of system-of-systems in Industry 4.0," *Appl. Sci.*, vol. 12, no. 15, p. 7730, Aug. 2022.

Gianvito Urgese (Senior Member, IEEE) received the Ph.D. degree in computer and systems engineering from the Politecnico di Torino, Turin, Italy, in 2016.

He is an Associate Professor with the Politecnico di Torino. His main research interests are neuromorphic computing and engineering, parallel and heterogeneous architectures, Artificial Intelligence of Things application development, and algorithm optimization focused on bioinformatics and embedded-systems domains.

Vittorio Fra received the M.Sc. degree in nanotechnologies for ICTs and the Ph.D. degree in physics from the Politecnico di Torino, Turin, Italy, in 2017 and 2021, respectively.

He is a Researcher and an Assistant Professor with the Politecnico di Torino. In the EDA Group, he focuses his activity on neuromorphic and neuro-inspired computing.

Andrea Pignata (Graduate Student Member, IEEE) received the M.Sc. degree in computer engineering focus on embedded systems from the Politecnico di Torino, Turin, Italy, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering (DAUIN).

He is a member of SmartData@Polito Research Center, Politecnico di Torino. His main focus with the EDA Group as a Researcher is on neuromorphic computing and IoT systems.

Giuseppe Fanuli received the M.Sc. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2024.

He is a Research Fellow with the Politecnico di Torino. In the EDA Group, his main research focuses on cloud computing and HW resource sharing.

Walter Gallego Gomez received the M.Sc. degree in computer engineering from Politecnico di Torino, Turin, Italy, in 2018, where he is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering (DAUIN), EDA Group.

His Ph.D. focuses on optimizing neural networks deployment on neuromorphic HW for life-science applications.

Riccardo Pignari received the M.Sc. degree in physics of complex systems from the Politecnico di Torino, Turin, Italy, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering (DAUIN).

He is a member of SmartData@Polito Research Center, Politecnico di Torino. His main focus at the EDA Group is on neuromorphic and neuro-inspired computing.

Michelangelo Barocci (Graduate Student Member, IEEE) received the M.Sc. degree in electronics engineering from the Politecnico di Torino, Turin, Italy, in 2021, where he is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering (DAUIN).

His main focus at the EDA Group is on the development of heterogeneous digital systems on FPGA for prototyping neuromorphic computing solutions.

Benedetto Leto received the M.Sc. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2024.

He is a Research Fellow with the Politecnico di Torino. In the EDA Group, his main research interests are neuromorphic computing and AIoT applications.

Salvatore Tilocca received the M.Sc. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2024.

He is a Research Fellow with the Politecnico di Torino. In the EDA Group, his main research interests are neuromorphic computing and bioinformatics applications.

Nicola Cassetta received the M.Sc. degree in data science from the Università di Padua, Padua, Italy, in 2024.

He is a Research Fellow with the Politecnico di Torino, Turin, Italy. In the EDA Group, his main research interests are neuromorphic computing and computer vision applications.

Paolo Montuschi (Fellow, IEEE) is a Full Professor with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy, and a Member of its Board of Governors. He Co-Founded Italy's First HKN Chapter in 2017. His research spans computer arithmetic/architectures, computer graphics, and electronic publications.

Mr. Montuschi is an IEEE Computer Society Golden Core Member. He received the CS Distinguished Service and Spirit of the Computer Society Awards.

Enrico Macii (Fellow, IEEE) received the Ph.D. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 1995.

He is a Full Professor of computer engineering with the Politecnico di Torino, Turin, Italy. His research interests are in the design of electronic digital circuits and systems, with a particular emphasis on low-power consumption aspects.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement