Università di Genova

PhD Program in Science and Technology for Electronic and Telecommunication Engineering

Enhancing the Measurify IoT data management platform with high performance, edge-optimized deep learning models for time series processing

Matteo Fresta

Thesis submitted for the degree of *Doctor of Philosophy* (38° cycle)

May 2026

Prof. Riccardo Berta — Tutor
Prof. Francesco Bellotti — Tutor
Prof. Maurizio Valle — Head of the PhD program

Università di Genova | DITEN DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

ENHANCING THE MEASURIFY IOT DATA MANAGEMENT PLATFORM WITH HIGH PERFORMANCE, EDGE-OPTIMIZED DEEP LEARNING MODELS FOR TIME SERIES PROCESSING

Matteo Fresta

Reviewers

Paolo MOTTO ROS, Assistant Professor, University of Turin, Italy
Federica ZONZINI, Assistant Professor, University of Bologna, Italy

Examiners

Mauro OLIVIERI, Full Professor, University of Rome, Italy
Davide BRUNELLI, Full Professor, University of Bologna, Italy
Edoardo RAGUSA, Assistant Professor, University of Genoa, Italy

---

University of Genoa

Matteo Fresta

ENHANCING THE MEASURIFY IOT DATA MANAGEMENT PLATFORM WITH HIGH PERFORMANCE, EDGE-OPTIMIZED DEEP LEARNING MODELS FOR TIME SERIES PROCESSING

xix+257 pages.

To my family

Funding acknowledgment

The Hi-Drive project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101006664. The sole responsibility of this publication lies with the authors. Neither the European Commission nor CINEA - in its capacity of Granting Authority - can be made responsible for any use that may be made of the information this document contains.

Abstract

This thesis presents an end-to-end workflow for the management, analysis, and deployment of time series-based machine learning systems in IoT and Edge AI applications.

Modern systems continuously generate large volumes of time series data describing the temporal evolution of physical processes.

To manage this time series data effectively, workflows must cover data acquisition, storage, model training, and deployment on embedded hardware. In literature, these steps are often treated separately, resulting in fragmented solutions, limiting reproducibility, scalability, and real-world applicability.

This thesis addresses this fragmentation by proposing a unified workflow that spans the entire lifecycle of time series data, from acquisition on edge devices to real-time inference on resource-constrained platforms. Rather than focusing on isolated algorithms or individual application domains, the work adopts a system-level perspective, integrating data management, machine learning, and deployment-oriented evaluation into a coherent and reusable methodology.

The proposed workflow is built around Measurify, an open-source, measurement-oriented IoT data management framework. Throughout the thesis, Measurify is extended to support large-scale time-series ingestion through robust CSV-based workflows, semantic data modeling, experiment metadata management, and a Model Registry for datasets, algorithms, and trained models. These extensions enable reproducible experimentation and consistent dataset handling across heterogeneous application domains.

Building on this infrastructure, the thesis investigates a range of lightweight machine learning models specifically tailored for time series processing under realistic embedded constraints. The proposed models include compact convolutional and recurrent architectures (such as 1D-CNN and LSTM), efficient feature-based methods (such as MiniRocket), and highly compressed neural networks (such as binary neural networks), evaluated not only in terms of predictive accuracy but also with respect to latency, memory footprint, energy

consumption, and robustness. This evaluation highlights practical trade-offs that are often overlooked in purely algorithmic studies.

A central aspect of the proposed workflow is the explicit consideration of heterogeneous target hardware platforms. Throughout the thesis, models are designed and evaluated across a spectrum of embedded devices, ranging from low-end microcontroller-class platforms such as Arduino and STM-based systems, to single-board computers including Raspberry Pi and NVIDIA Jetson Nano, and up to Field-Programmable Gate Arrays (FPGAs) for hardware-accelerated deployment.

The methodology is applied and validated across multiple application domains, including human and sport activity recognition, structural health monitoring, and driver distraction detection. For each domain, the existing literature is analyzed and the proposed solutions advance the state-of-the-art by achieving competitive or improved performance under more realistic evaluation settings, while explicitly accounting for deployability constraints.

Overall, this thesis demonstrates that combining time series data management, lightweight machine learning, and embedded deployment within a unified workflow enables both scientific progress and practical applicability, bridging the gap between machine learning research and real-world Edge AI systems.

Keywords: Time Series Analysis; Edge AI; Internet of Things; Lightweight Machine Learning; Embedded Deployment; Data Management Workflow; TinyML; FPGA.

Abstract (ITA)

Questa tesi presenta un workflow end-to-end per la gestione, l'analisi e il deployment di sistemi di machine learning basati su serie temporali in applicazioni IoT ed Edge AI.

I sistemi moderni generano continuamente grandi volumi di dati temporali che descrivono l'evoluzione nel tempo di processi fisici.

Per gestire efficacemente questi dati, è necessario disporre di soluzioni che coprano l'intero ciclo di vita dei dati, dall'acquisizione alla memorizzazione, dall'addestramento dei modelli, fino al deployment su sistemi hardware. Tuttavia, nella letteratura e nella pratica applicativa, queste fasi sono spesso affrontate in modo disgiunto, dando origine a soluzioni frammentate che limitano la riproducibilità, la scalabilità e l'effettiva applicabilità nel mondo reale.

La tesi affronta questa frammentazione proponendo un workflow unificato che integra gestione dei dati, machine learning e valutazione orientata al deployment, coprendo l'intero ciclo di vita delle serie temporali: dall'acquisizione su dispositivi lato edge fino all'esecuzione in tempo reale su piattaforme a risorse limitate.

Piuttosto che concentrarsi su singoli algoritmi o su casi di studio isolati, il lavoro adotta una prospettiva a livello di sistema, in cui i modelli vengono progettati e valutati come componenti di sistemi embedded soggetti a vincoli reali di latenza, memoria ed energia.

All'interno di questo contesto, Measurify viene adottato come infrastruttura di riferimento per la gestione dei dati. Measurify è un framework open-source e orientato alle misure per applicazioni IoT, che nel corso della tesi è stato esteso per supportare l'ingestione di serie temporali su larga scala tramite workflow robusti basati su CSV, modellazione semantica dei dati, gestione dei metadati sperimentali e un Model Registry per dataset, algoritmi e modelli addestrati. Tali estensioni consentono sperimentazioni riproducibili e una gestione coerente dei dati in domini applicativi eterogenei.

A partire da questa infrastruttura, la tesi analizza e sviluppa modelli di machine

learning leggeri, specificamente progettati per l'elaborazione di serie temporali in contesti embedded. I modelli considerati includono architetture convoluzionali e ricorrenti (come 1D-CNN e LSTM), metodi efficienti basati su feature (come MiniRocket) e reti neurali fortemente compresse (come le Reti Neurali Binarie). Le prestazioni vengono valutate non solo in termini di accuratezza, ma anche considerando latenza, occupazione di memoria, consumo energetico e robustezza, mettendo in evidenza compromessi progettuali spesso trascurati negli studi puramente algoritmici.

Un elemento centrale del lavoro è la valutazione su piattaforme hardware eterogenee. I modelli sono testati su un ampio spettro di dispositivi embedded, che include microcontrollori a bassissime risorse come Arduino e sistemi basati su STM, single-board computer come Raspberry Pi e NVIDIA Jetson Nano, fino a FPGA per il deployment con accelerazione hardware.

La metodologia proposta è validata in diversi domini applicativi, tra cui il riconoscimento di attività umane e sportive, il monitoraggio della salute strutturale e il rilevamento della distrazione del conducente. In ciascun dominio, le soluzioni proposte avanzano lo stato dell'arte ottenendo prestazioni competitive o migliorative in condizioni di valutazione più realistiche, tenendo esplicitamente conto dei vincoli di applicabilità.

Nel complesso, questa tesi dimostra che l'integrazione di gestione delle serie temporali, machine learning e deployment embedded all'interno di un workflow coerente consente di colmare il divario tra la ricerca sul machine learning e lo sviluppo di sistemi Edge AI realmente utilizzabili in contesti industriali e IoT.

Parole chiave: Analisi delle Serie Temporali; Edge AI; Internet delle Cose; Machine Learning Leggero; Deployment su Sistemi Embedded; Workflow di Gestione dei Dati; TinyML; FPGA.

Contents

List of Figures

List of Tables

Acronyms

1D-CNN	One-Dimensional Convolutional Neural Network
ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
API	Application Programming Interface
BNN	Binary Neural Network
BMS	Battery Management System
BRAM	Block Random Access Memory
CAN	Controller Area Network
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
DDD	Driver Distraction Detection
DL	Deep Learning
DSP	Digital Signal Processor
ECG	Electrocardiogram
EDA	Electrodermal Activity
Edge AI	Artificial Intelligence on Edge Devices
FF	Flip-Flop
FPGA	Field-Programmable Gate Array
GRU	Gated Recurrent Unit
HAR	Human Activity Recognition
HDF5	Hierarchical Data Format version 5
HLS	High-Level Synthesis
HR	Heart Rate
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
LSTM	Long Short-Term Memory
MAC	Multiply–Accumulate Operation
MCU	Microcontroller Unit
ML	Machine Learning
MLP	Multilayer Perceptron
NoSQL	Not Only SQL
OA	Overall Accuracy
OCSVM	One-Class Support Vector Machine
OOD	Out-Of-Distribution
OSM	OpenStreetMap
PCA	Principal Component Analysis
PPV	Proportion of Positive Values
RAM	Random Access Memory

ReLU	Rectified Linear Unit
REST	Representational State Transfer
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SHAP	SHapley Additive exPlanations
SHM	Structural Health Monitoring
SoC	System on Chip
STM	STMicroelectronics Microcontroller
SVM	Support Vector Machine
TFLite	TensorFlow Lite
TinyML	Tiny Machine Learning
ToF	Time of Flight
TOR	Take-Over Request
XML	Extensible Markup Langue

CHAPTER 1

1. Introduction

In recent years, the rapid growth of the Internet of Things (IoT) has significantly changed the way data is collected, processed, and used in many application domains [1]. Sensors embedded in wearable devices [2], vehicles [3], industrial machines [3], buildings [4], and urban infrastructures [5] continuously generate measurements that describe how systems behave over time. These measurements often take the form of time series, which capture the temporal evolution of physical quantities such as acceleration, voltage, steering angle, vibration, or physiological signals. Time series are therefore a central element in modern monitoring and decision-making systems, and their relevance continues to grow as IoT deployments scale in size and heterogeneity [6].

However, the widespread availability of sensor data also introduces practical challenges [7]. IoT devices are usually designed to be inexpensive, compact, and energy-efficient, meaning that they have limited computational resources. For this reason, it is often not feasible to send all raw data to the cloud for processing. Instead, it is increasingly desirable to analyze data directly on-site, following the edge computing paradigm. This reduces communication overhead, protects user privacy, and allows systems to react more quickly to events. At the same time, edge devices must operate under strict memory, latency, and energy constraints, and this makes the deployment of machine learning (ML) models particularly challenging.

To fully exploit the value of time series in IoT applications, it is therefore necessary to design complete workflows that cover all steps of the data lifecycle: from acquisition on the device, to storage and organization in the cloud, to training and deployment of ML models, and finally to real-time inference on embedded devices. Unfortunately, existing solutions are often fragmented. Data

management tools may not support time series in a flexible way, ML workflows may rely on ad-hoc scripts that are difficult to reproduce, and embedded deployments may require significant manual optimization. This fragmentation creates barriers not only for researchers but also for companies and institutions that want to build reliable, scalable IoT systems.

The work presented in this thesis aims to address these challenges by developing a unified methodology that links data ingestion, management, ML, and embedded deployment into a coherent and reusable workflow. The central idea is that modern IoT applications require more than isolated algorithms or hardware components, but a coherent experimental setting in which data, models, and devices can be studied together under realistic constraints. In this thesis, Measurify [8], [9], an open-source, IoT cloud-based framework, is enhanced as a data management backbone to support this setting, providing consistent dataset organization, metadata handling, and experiment reproducibility. Its extension to time series data provides the experimental foundation on which the thesis advances the state-of-the-art in multiple application domains, by enabling consistent evaluation of ML models and deployment strategies under realistic conditions.

A first contribution of the thesis concerns the development of a robust workflow for loading and managing CSV datasets [10]. Although CSV files are among the most common formats for sharing time series, they often come with inconsistent headers, missing metadata, or ambiguous column structures. To overcome these issues, a dedicated ingestion mechanism was designed, based on a lightweight JSON descriptor that explains how each column should be interpreted. This makes it possible to map heterogeneous CSV files into Measurify's internal data model in a systematic and reproducible way. The workflow was tested on hundreds of public datasets and also adopted in an industrial context within the European Hi-Drive project [11], where Measurify was used to manage experiment metadata and compute performance indicators for large-scale automated driving trials.

Once a consistent data management layer is available, ML becomes the next crucial step for extracting value from time series data. In recent years,

increasingly complex deep learning models have shown remarkable performance in capturing temporal dependencies and nonlinear patterns. However, the majority of these approaches are developed and evaluated in unconstrained, cloud-based settings, where computational resources and energy consumption are not limiting factors. As a result, model accuracy is often prioritized over deployability, leaving critical aspects such as memory footprint, latency, and power efficiency largely unaddressed.

This thesis address this gap by explicitly treating energy efficiency, resource optimization, and real-time execution as primary design constraints. From this perspective, ML models are not considered alone, but as components of embedded systems that must operate reliably under strict hardware and energy budgets. Consequently, this work aimed to advance the state-of-the-art by studying time series models that balance predictive performance with practical feasibility on real edge devices.

Unlike cloud-based environments, edge devices must often perform inference in real time, with limited hardware resources and, in many cases, battery-powered operation. For these reasons, the thesis explores a range of lightweight model architectures, including compact 1-dimension convolutional neural networks (1D-CNN), dilated models for long temporal dependencies, efficient feature extractors such as Minimally random convolutional kernel transform (MiniRocket), and even binary neural networks, which reduce both storage and computation by using 1-bit weights and activations. The choice of model depends strongly on the target hardware platform, and throughout the thesis several real devices are considered, ranging from Arduino Nano and STM-based microcontrollers to Raspberry Pi boards, NVIDIA Jetson modules, and Field-Programmable Gate Arrays (FPGA) platforms. Figure 1.1 summarizes the end-to-end workflow of the thesis, from data ingestion and management to model training and edge deployment across the considered application domains.

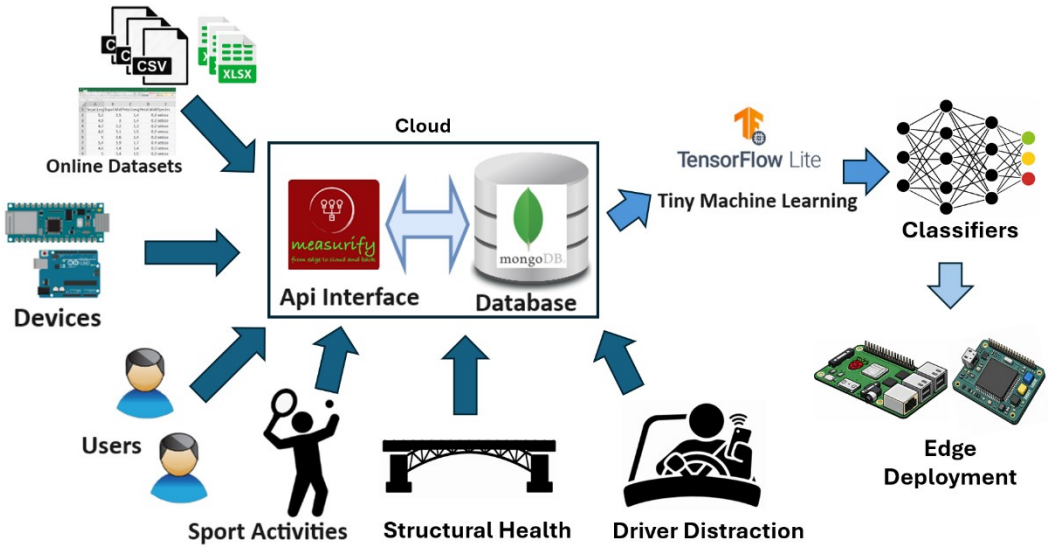


Figure 1.1: Overview of the end-to-end workflow addressed in this thesis.

These methodological developments are validated across multiple application domains, each one presenting different challenges and demonstrating the flexibility of the proposed workflow (Figure 1.2).

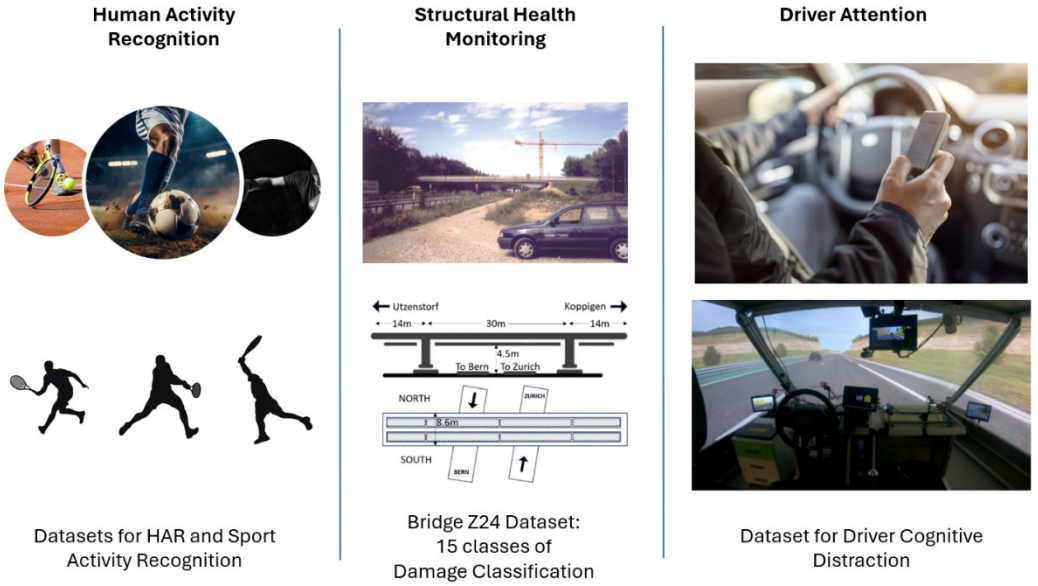


Figure 1.2: Application domains addressed in this thesis

The first set of experiments focuses on human and sports activity recognition [12]. Wearable devices equipped with inertial sensors are used to collect labelled time series during sports such as tennis, skiing, fit walking, and boxing. The data are stored in Measurify and used to train ML models, that are finally deployed for real-time classification on extremely resource-constrained microcontrollers, such

as the Arduino Nano 33 BLE Sense, achieving state-of-the-art performance under severe hardware limitations. These case studies highlight the importance of efficient data collection pipelines, the impact of sensor placement, and the trade-offs involved in deploying models on low-power platforms.

The thesis then considers a more complex domain: Structural Health Monitoring (SHM) [13]. In this context, vibration signals recorded from bridges are analyzed to detect and classify different structural conditions. SHM datasets are challenging for time series analysis, because they are typically very long, noisy, and influenced by environmental factors, often requiring extensive data cleaning, preprocessing and feature engineering.

However, this thesis demonstrates that models such as WaveNet and MiniRocket can achieve state-of-the-art performance when also trained directly on raw signals, without relying on handcrafted features or complex preprocessing pipelines.

Furthermore, the deployability of these models on embedded platforms is evaluated to minimize latency, memory footprint, and power consumption. A feature analysis based on SHAP values provides insights into which temporal patterns are most relevant for the classification task.

A further application addressed in the thesis is Driver Distraction Detection (DDD), a topic of increasing relevance for road safety and semi-automated driving [14]. In this work, multiple sensor modalities are combined, including vehicular data, eye tracking, and physiological signals. The goal is to detect whether the driver is engaged in a cognitively distracting task. This scenario is challenging because sensor alignment is not perfect, physiological signals may contain noise or missing values, and different users exhibit different behavioral patterns. In the literature, these challenges are often addressed by increasingly complex and large models that rely on a high number of input features and long temporal windows to capture sufficient context. While this approach can improve accuracy, it significantly increases computational cost and latency, making real-time deployment on embedded automotive platforms difficult.

This thesis advances the state-of-the-art by demonstrating that deep learning models specifically designed for time series processing can generalize across

unseen drivers and achieve robust performance even when operating on short time windows, thereby enabling real-time inference under the computational, memory, and energy constraints of on-board embedded platforms. Finally, the thesis explores the use of FPGA boards for edge deployment [15]. FPGAs offer the possibility of highly parallel computation with low latency, but require dedicated development workflows and often impose strict limitations on model architectures. Through a set of prototype implementations, including GRU-based battery state-of-charge estimation and hybrid anomaly detection models, the thesis examines both the potential and the practical constraints of FPGA-based AI for time series.

Across all these domains, the contributions of the thesis are unified by a common vision: studying time-series intelligence as an end-to-end problem that spans data management, model design, and embedded deployment. By relying on Measurify as backbone and deployment-aware experimentation, the work shows that meaningful advances in IoT and Edge AI arise from integrated, experimentally validated approaches rather than isolated domain-specific pipelines. The final goal is not only to produce accurate models for specific tasks, but to provide a general, reusable methodology that supports future research and applications across a wide range of IoT scenarios.

1.1 Research questions

The broad goal of this thesis is to design a unified and practical workflow for managing and analyzing time series in IoT applications, from data collection to edge deployment. To guide this objective, several research questions were formulated. These questions reflect the main challenges identified throughout the introduction: flexible data ingestion, scalable management of time series, the design of lightweight deep learning models, and their deployment on resource-constrained devices across different domains.

Table 1.1: Research questions

Research Question	**Addressed Chapter**
RQ1. Can online CSV datasets be uploaded with optimal performance into a database automatically, without requiring user-side modifications or specific expertise in data management?	Chapter 3
RQ2. Can a state-of-the-art open-source IoT data management tool provide a generalized and scalable framework to support different time series ML developments across domains?	Chapter 3
RQ3. Is it possible to develop an end-to-end versatile system architecture for efficiently and easily collecting sport activity dataset?	Chapter 4
RQ4. Can devices with extreme limited resources and low-cost, such as an Arduino board, provide reliable results in ultra-low-power and low-energy contexts?	Chapter 4
RQ5. Is an end-to-end ML approach suited, or is extensive signal preprocessing needed for proper feature extraction?	Chapter 5
RQ6. Are MiniRocket and WaveNet models able to achieve state-of-the-art classification accuracy, even with memory footprint and energy consumption levels suitable for field deployment on the civil infrastructure?	Chapter 5
RQ7. Can DL models detect driver distraction patterns using very short time windows, while remaining deployable on low-cost vehicular platforms under strict constraints on latency, model size, and energy consumption?	Chapter 6
RQ8. How can neural networks be further quantized and optimized for FPGA prototyping, while preserving accuracy and enabling deployment across different application domains?	Chapter 7

1.2 Use-Case Scenarios

To complement the research questions summarized in Section 1.1, Table X provides a synoptic view of the main use-case scenarios addressed in the thesis. The purpose of this overview is to make explicit why these scenarios were selected, which IoT and Edge AI challenges they represent, and how they complement each other within the proposed end-to-end workflow.

The selected scenarios were not intended as isolated case studies, but as

representative domains covering different yet recurrent challenges in IoT time-series management and deployment-oriented machine learning. In particular, they span heterogeneous sensing conditions, data structures, latency and energy constraints, safety-critical requirements, and hardware targets, ranging from microcontroller-class devices to FPGA platforms. In this sense, the scenarios jointly provide a broad and balanced validation setting for the methodology proposed in this thesis. This framing is consistent with the thesis objective of developing a unified workflow that connects data ingestion, management, model development, and deployable inference across multiple application domains. thesis is organized as follows:

Table 1.2: Use-case scenarios addressed in the thesis and corresponding IoT challenges

Use-case scenario	**Main chapter**	**Why selected**	**Main IoT challenges covered**
IoT data management	Chapter 3	To validate the data-management backbone of the thesis across heterogeneous datasets and application contexts	Heterogeneous data ingestion, semantic organization, reproducibility, scalability, metadata management
Sport and Human Activity Recognition	Chapter 4	To assess the proposed workflow in wearable and highly resource-constrained embedded settings	Real-time sensing, edge inference, low-power operation, compact models, end-to-end data collection
Structural Health Monitoring	Chapter 5	To evaluate the methodology on long, noisy, infrastructure-scale time series in a safety-relevant domain	Large-scale time-series analysis, robustness, continuous monitoring, deployability, interpretability
Driver Distraction Detection	Chapter 6	To test the workflow in a multimodal and safety-critical automotive scenario	Multimodal fusion, user variability, low latency, robustness, embedded deployment constraints
FPGA-based Edge AI for Time Series	Chapter 7	To extend the workflow toward hardware-specialized deployment and deterministic inference	Quantization, hardware-aware optimization, energy efficiency, deterministic low-latency execution

Taken together, these use-case scenarios provide a representative coverage of the challenges addressed in this thesis: heterogeneous time-series ingestion and management, real-time embedded inference, scalability to long and noisy signals, multimodal fusion, robustness under safety-critical constraints, and deployment across heterogeneous hardware platforms. For this reason, they should be interpreted as complementary validation settings for the same end-to-end methodology, rather than as isolated application examples.

1.3 Deployment platforms

An important aspect of this thesis is the evaluation of time-series machine learning solutions on different hardware platforms. Since these devices have different characteristics and are used for different purposes, it is useful to clarify how they are considered in this work and which role they play in the application scenarios addressed. In this thesis, the term **extreme edge** refers to devices located in the immediate proximity of the sensor, typically battery-powered and characterized by very limited computational, memory, and energy resources. These devices are responsible for direct data acquisition and, when possible, for the first level of on-board inference. Typical examples considered in this thesis are microcontroller-class platforms such as Arduino Nano and STM-based boards. Their main advantages are low cost, compactness, ultra-low-power operation, and direct deployability close to the physical process, but these benefits come with strong limitations in terms of memory size, storage, floating-point support, and admissible model complexity. For this reason, extreme-edge deployment is mainly associated in this thesis with TinyML-oriented solutions and highly compact model formats. This perspective is particularly relevant in Chapter 4, where human and sport activity recognition is deployed on Arduino-class wearable devices, and in Chapter 6, where STM-based measurements are used to assess the deployability of compact driver distraction detection models under severe resource constraints. It is also coherent with Research Question 4, which explicitly addresses the feasibility of reliable inference on extremely resource-limited and low-cost devices.

By contrast, the term **edge** is used here for more capable embedded platforms that are still relatively close to the data source but provide significantly higher computational resources than microcontrollers. These platforms typically include single-board computers and embedded GPU-enabled systems such as Raspberry Pi boards and NVIDIA Jetson devices. Compared with extreme-edge nodes, they offer higher clock frequencies, larger RAM and storage capacities, and in some cases hardware support for floating-point or GPU-accelerated inference. As a consequence, they can host more demanding time-series models and support lower inference latency, while still remaining suitable for local or near-local deployment. In this thesis, this class of devices is mainly considered in Chapters 5 and 6, where Raspberry Pi and Jetson platforms are used to study the trade-offs among latency, energy consumption, and model format for structural health monitoring and driver distraction detection tasks. These platforms are therefore interpreted as edge devices in a stricter sense: more powerful than sensor-near microcontrollers, but still intended for local inference rather than cloud-scale training.

A third role is played by **fog devices**, which are not necessarily the main inference target but act as intermediaries between sensing nodes and the cloud. In the architecture developed in Chapter 4, for example, the smartphone-based fog layer receives data from the wearable edge device through BLE, manages buffering and session control, and forwards the data to the cloud infrastructure through Measurify. In this sense, the fog layer is distinguished from both the extreme edge and the edge inference platforms: it is closer to the user and to the sensing process than the cloud, but its primary role is orchestration, communication, and temporary storage rather than sustained model execution.

Finally, the **cloud** and workstation-level systems are used in this thesis mainly for dataset management, model training, large-scale validation, and experiment organization. This role is fulfilled by Measurify and by the associated model development and registry infrastructure presented in Chapter 3. In some experiments, workstation- or laptop-class systems are also used as a reference for high-performance inference, not because they belong to the edge layer, but

because they provide an upper baseline in terms of execution speed and model flexibility.

Instead, in this thesis **FPGAs** are treated as a specialized deployment target for edge AI rather than as standard extreme-edge devices. On the one hand, FPGA-based solutions can be physically deployed close to the sensing process and can provide deterministic low-latency inference with high energy efficiency. On the other hand, they require dedicated design flows, hardware-aware model optimization, and a more complex hardware/software integration process than typical microcontroller- or single-board-computer-based deployments. For this reason, Chapter 7 treats FPGAs as a distinct hardware-specialized edge layer, suitable for applications where deterministic timing, aggressive optimization, and custom parallelism become more important than software flexibility.

For this reason, the platforms considered in this thesis should not be seen as interchangeable boards, but as devices with different constraints, roles, and deployment objectives. Arduino and STM-based systems represent the extreme edge, where model compactness and energy efficiency are dominant requirements. Raspberry Pi and Jetson platforms represent the edge layer, where more complex models become feasible and latency can be significantly reduced. Smartphones can act as fog devices for coordination and communication. FPGAs represent a hardware-specialized edge solution for deterministic and highly optimized inference. The cloud, finally, remains the natural location for persistent data management, experiment traceability, training workflows, and orchestration.

1.4 Structure of the thesis

The thesis is organized as follows:

Chapter 2 surveys the state-of-the-art in IoT time series management, Edge AI, TinyML, and domain-specific applications (HAR, SHM, DDD).
Chapter 3 presents the extensions to Measurify, the CSV ingestion workflows, large-scale validations, usability studies, and the Model Registry.
Chapter 4 introduces the Edge-to-Cloud workflow and its application to sports activity recognition, including lightweight embedded models.
Chapter 5 applies the methodology to structural health monitoring, proposing advanced models (WaveNet, MiniRocket, Binary Neural Networks), deployment on embedded devices, and explainability analysis.
Chapter 6 addresses driver distraction detection using multimodal time series and evaluates deep models under realistic constraints.
Chapter 7 explores FPGA-based prototyping, providing a forward-looking analysis of hardware-level deployment for time-series AI.
Chapter 8 concludes the thesis, summarizing the contributions and indicating future directions.

CHAPTER 2

2. Related work

This chapter reviews the state-of-the-art related to the main challenges addressed in this thesis, focusing on the management and analysis of time-series data in IoT and Edge AI systems. The review spans multiple layers of the data and computation pipeline, covering data formats and database technologies, Edge-to-Cloud computing paradigms, lightweight machine learning models, and

representative application domains relevant to time-series analysis in IoT systems.

2.1 Time-Series Data, Formats, and Database Support in IoT

A large proportion of machine learning datasets, excluding multimedia data, is distributed in tabular form, most commonly as Comma Separated Values (CSV) files. In such datasets, each row typically corresponds to a measurement instance, while columns represent features, labels, or auxiliary information. Despite their apparent simplicity, CSV datasets exhibit substantial heterogeneity: fields may contain scalar or vector values, numerical or categorical data, and, in some cases, descriptive context such as device identifiers or acquisition locations. In many publicly available datasets, however, contextual information is either implicit or completely absent.

In IoT applications, these issues are further intensified by the continuous and high-frequency nature of sensor-generated time series. Measurements are often produced by heterogeneous devices, at different sampling rates, and under varying environmental conditions. As a result, effective data management solutions must support large-scale ingestion, preserve precise temporal information, and provide mechanisms to associate measurements with their acquisition context. Traditional relational databases struggle to meet these requirements, particularly when data schemas evolve or when workloads become highly intensive. For these reasons, non-relational database management systems (DBMSs) have gained increasing relevance in IoT and big data scenarios. Several studies have shown that NoSQL solutions outperform relational databases in terms of scalability and performance under heterogeneous and high-throughput workloads [16], [17]. In particular, MongoDB has been demonstrated to perform better than MySQL for collecting data from heterogeneous sensors and applications [18], [17]. Its document-oriented model and schema-less nature make it well suited to storing semi-structured time-series data.

However, flexibility at the storage level does not automatically translate into

semantic clarity. While MongoDB efficiently stores JSON documents, it does not enforce any specific data model for representing IoT concepts such as devices, measured quantities, or observed entities. As a consequence, application developers must impose their own conventions, which often leads to fragmented and domain-specific solutions.

Alongside database technologies, several data formats have been adopted for storing and exchanging time series data. XML provides a highly structured and semantics-preserving representation, but its verbosity and parsing overhead limit its use in large-scale IoT systems [19], [20]. JSON has emerged as a lightweight and widely adopted alternative, offering ease of parsing and direct compatibility with document-oriented databases [21]. For scientific and archival purposes, HDF5 is frequently employed due to its support for n-dimensional datasets and high-performance I/O [22]. Nevertheless, HDF5 is typically used for batch access and is less suited to incremental ingestion or interactive workflows.

Despite these alternatives, CSV remains the de facto standard for data exchange and exploratory analysis in many domains, including economics [23], data analysis [24], and machine learning [25]. Even complex pipelines often rely on CSV as an intermediate representation. For instance, in the L3Pilot project, vehicular time-series are stored in a Common Data Format (CDF) encoded in HDF5, but are later exported as CSV files to enable analysis by traffic and safety experts [1], [24].

A critical limitation of CSV files is the lack of a formal standard [26]. Variations in delimiters, file encodings, quoting rules, and handling of missing values can cause compatibility issues and data integrity problems [27]. These inconsistencies complicate the design of automated ingestion pipelines, especially when dealing with large collections of heterogeneous datasets. Tools such as mongoimport partially address this problem by enabling CSV ingestion into MongoDB collections, but they offer limited type validation, no duplicate detection, and reject entire files upon encountering a single malformed row. While acceptable for backup purposes, such behavior is inadequate for robust IoT data integration workflows.

These limitations motivate the need for higher-level abstractions and data management frameworks, which are discussed in the following sections.

2.2 Measurement-Oriented Frameworks for IoT Time-Series Data

To overcome the limitations of flat tabular storage and ad-hoc data modeling, measurement-oriented data management frameworks have emerged in the context of IoT and large-scale sensing applications. These frameworks adopt abstract schemas that explicitly model the context in which measurements are generated. Rather than storing isolated values, measurements are associated with well-defined entities such as devices, measured features, and observed objects, enabling richer semantic representation and more consistent downstream processing.

Before the improvements introduced in this thesis, Measurify, formerly known as Atmosphere [28], was an open-source measurement-oriented framework developed to support IoT applications and large-scale experimental campaigns. It represented application contexts as interrelated resources, including Thing, Feature, Device, and Measurement, exposed through a RESTful API and backed by MongoDB storage [18], [24]. This abstraction allowed heterogeneous domains to be described in a uniform way, while preserving flexibility in the underlying data representation.

A defining characteristic of measurement-oriented frameworks is that mandatory entities must be explicitly defined for each measurement. While this requirement complicates data insertion compared to flat CSV-based approaches, it enables richer contextualization and improves data consistency, filtering, and reuse. However, most publicly available datasets do not conform to such schemas, as they typically omit redundant contextual information such as device identifiers, feature definitions, or measurement metadata. As a result, direct ingestion of tabular datasets into measurement-oriented databases is not straightforward and generally requires an intermediate adaptation layer.

In parallel, several general-purpose IoT platforms and cloud-based solutions have been proposed to support data ingestion and management at scale. Commercial platforms such as Amazon AWS IoT and Microsoft Azure IoT provide guided procedures for uploading structured and semi-structured data, including CSV files, into their internal data stores. Nevertheless, these solutions are often vendor-locked and provide limited transparency and customization of the underlying data model. By abstracting away schema details, they make it difficult to reason about data structure, semantic consistency, and reproducibility, which are critical aspects in research-oriented and safety-critical contexts.

Beyond commercial offerings, a number of open-source and standardization-driven IoT frameworks have been proposed, each emphasizing different architectural goals such as service interoperability, device discoverability, or cloud integration. However, many of these platforms are not primarily designed for measurement-centric time-series management and exhibit limitations when dealing with heterogeneous, high-frequency, and context-rich temporal data. A comparative overview of representative general-purpose and open-source IoT platforms, highlighting their architectural focus and limitations with respect to measurement-oriented time-series data management, is provided in Table 2.1.

Table 2.1: Representative projects proposing general-purpose and open source IoT architectures / platforms

Publicly available open source IoT projects
OpenIoT [29]: IoT middleware supporting semantic data modeling, service discovery, and cloud integration. It is designed mainly for smart city applications and academic use. Interoperability is achieved at the service level via the Global Sensor Network, but the system is relatively complex and not tailored to lightweight measurement-centric deployments.
Web of Things / Objects [30]: Conceptual framework developed by W3C to enable semantic interoperability across smart devices using standard web technologies. It provides a unified description of Things via Thing Descriptions, focusing on metadata, discoverability, and linking capabilities. However, it lacks a concrete implementation for full-stack data collection, storage, and processing.
BIG IoT [31]: A project aimed at enabling interoperability across IoT platforms through a

common Web API and a service marketplace. It focuses on high-level service registration, discovery, and monetization, primarily in smart city scenarios. While it fosters platform-to-platform integration, it does not define low-level data models or provide concrete tools for measurement acquisition and storage.
IoT AWS [32]: A cloud-based platform by Amazon Web Services offering scalable and secure IoT services, including device management, data ingestion, analytics, and integration with other AWS services. While powerful, it is tightly coupled to the AWS ecosystem, often requiring complex configuration and limiting transparency and customization for smaller or domain-specific deployments.
MS Azure [33]: A highly integrated cloud platform offering end-to-end IoT services, including device provisioning, data streaming, analytics, digital twins, and machine learning integration. While feature-rich, it introduces complexity and cost, with limited transparency in internal workflows and strong dependence on Microsoft infrastructure and cloud services.

2.3 Edge-to-Cloud Architectures for Time Series Data

The increasing availability of time-series data generated by IoT devices has led to the adoption of distributed computing paradigms that go beyond traditional cloud-centric architectures [34]. While centralized cloud platforms provide virtually unlimited storage and computational resources, relying exclusively on the cloud for data processing presents significant limitations when dealing with high-frequency, latency-sensitive, and safety-critical time-series data.

In cloud-only architectures, raw sensor data are continuously transmitted from edge devices to remote data centers for storage and processing. Although this

approach simplifies system design, it introduces several drawbacks, including network latency, bandwidth consumption, dependency on continuous connectivity, and potential privacy concerns. These issues become particularly evident in IoT scenarios characterized by large numbers of devices generating dense temporal streams, where transmitting all raw measurements to the cloud may be inefficient or even infeasible.

To address these challenges, Edge-to-Cloud and Fog computing paradigms have been proposed [35], [36], [37]. In these architectures, data processing responsibilities are distributed across multiple layers, typically including edge devices, intermediate fog nodes, and centralized cloud services. Edge devices are located close to the data sources and are responsible for data acquisition, preliminary processing, and, in some cases, real-time decision making. Fog nodes act as intermediaries that aggregate data from multiple edge devices, perform additional processing, and manage communication with the cloud [38]. The cloud remains responsible for long-term storage, large-scale analytics, model training, and system orchestration.

For time-series data, this hierarchical distribution of computation is particularly advantageous [35]. Many IoT applications require prompt reactions to temporal patterns, such as anomaly detection, event recognition, or control actions, which cannot tolerate the delays introduced by cloud. By performing time-critical operations at the edge, Edge-to-Cloud architectures enable low-latency responses while reducing the volume of data transmitted over the network [39]. At the same time, the cloud can operate as a backbone for dataset consolidation, historical analysis, and the development of more complex models that are later deployed closer to the data sources.

Another important aspect of Edge-to-Cloud architectures is robustness [38]. In real-world deployments, network connectivity may be intermittent or unreliable. Systems that rely entirely on cloud connectivity are vulnerable to service disruptions, whereas distributed architectures can continue operating locally even in the absence of a stable connection. This property is essential in industrial, energy, and transportation domains, where uninterrupted operation is often a

requirement.

From a data management perspective, Edge-to-Cloud workflows introduce additional complexity. Measurements may be generated, processed, and stored at different layers of the architecture, requiring consistent data models, synchronization mechanisms, and metadata management across the system. Without a coherent abstraction, integrating edge-generated data into cloud-based repositories can lead to fragmented datasets and loss of contextual information, undermining reproducibility and traceability [36].

Recent research has therefore emphasized the need for unified workflows that integrate data ingestion, storage, and processing across edge and cloud layers. In this view, the cloud is not merely a remote processing endpoint, but a central component for dataset management, experiment reproducibility, and coordination of distributed analytics pipelines. This perspective is particularly relevant for time-series applications, where the same data may be used for real-time inference at the edge and for offline analysis, model training, or validation in the cloud [36].

In summary, Edge-to-Cloud architectures represent a fundamental shift in the way time-series data are managed and processed in IoT systems. By distributing computation across edge, fog, and cloud layers, these architectures address the limitations of cloud-only solutions and provide the foundation for scalable, low-latency, and resilient IoT applications. The following sections build upon this architectural context to review approaches for deploying machine learning models on constrained devices and to analyze the implications of Edge AI and TinyML for time-series processing.

2.4 Edge AI and TinyML for Time Series Processing

Edge AI [40] focuses on executing ML models directly on devices located close to the data sources, enabling local inference under strict constraints on memory, computation, and power consumption. In the context of IoT time series data, this paradigm is particularly relevant, as models must process continuous temporal

streams while meeting real-time and reliability requirements on resource-constrained hardware [41].

A specialized branch of Edge AI is TinyML [42], which targets microcontroller-class devices characterized by extremely limited computational resources. Typical platforms include low-power microcontrollers such as Arduino, STM32, and ESP-based boards, which often operate with only kilobytes of RAM and limited flash memory. Despite these constraints, recent advances in model design and optimization have shown that meaningful inference on time-series data can be achieved even on such devices.

Time series processing poses specific challenges in the TinyML context. Unlike static inputs, temporal signals require models to capture sequential dependencies and evolving patterns over time. Common approaches include one-dimensional convolutional neural networks (1D-CNNs), recurrent architectures such as recurrent neural networks (RNNs) and gated recurrent units (GRUs), and models based on dilated convolutions. While effective, these architectures must be carefully adapted to embedded environments by limiting model depth, reducing parameter counts, and constraining intermediate activations to fit within the available memory and computational budget.

A central trade-off in Edge AI concerns the balance between predictive accuracy and resource consumption [41]. Increasing model complexity often improves performance, but at the cost of higher memory usage, longer inference latency, and increased power consumption. In real-time or battery-powered systems, these factors may be more critical than raw accuracy. As a result, Edge AI research emphasizes the evaluation of models not only in terms of classification or prediction metrics, but also with respect to deployability indicators such as execution time, memory footprint, and power consumption.

Another defining aspect of Edge AI workflows is the separation between training and inference. Due to hardware limitations, model training is typically performed in the cloud or on dedicated compute platforms, where large datasets and computational resources are available. Trained models are then converted, optimized, and deployed on edge devices for inference. This separation naturally

complements Edge-to-Cloud architectures, in which the cloud supports dataset management and model development, while the edge layer is responsible for real-time decision making [38].

From a system perspective, deploying ML models at the edge introduces additional challenges related to model versioning, updates, and consistency across distributed devices [43]. As models evolve over time, ensuring that edge deployments remain synchronized and validated becomes increasingly complex, particularly in large-scale IoT systems. These considerations motivate the need for structured workflows that integrate data management, model optimization, and deployment within a unified framework.

In summary, Edge AI and TinyML enable the deployment of time-series machine learning models on constrained devices, shifting intelligence closer to the data sources while introducing new constraints on model design and execution. These challenges have driven the development of model compression and optimization techniques, which are reviewed in the following section.

2.5 Model Compression and Deployment Toolchains for Edge AI

Building upon the challenges discussed in the previous section, research has focused on model compression and optimization techniques to enable the deployment of machine learning models on constrained edge devices [44], [45]. These techniques aim to reduce memory footprint, computational complexity, and power consumption while preserving acceptable levels of accuracy, which is particularly critical for time-series processing tasks. Model compression techniques aim to reduce the memory footprint and computational cost of neural networks. Among the most widely adopted approaches is quantization, which

reduces the numerical precision of model parameters and activations [46], [47], [48]. Common strategies include fixed-point quantization (e.g., INT8) and mixed-precision schemes, which have been shown to significantly decrease model size and inference latency with limited impact on accuracy. Quantization is especially relevant for time-series models deployed on microcontrollers, where floating-point operations may be expensive or unsupported.

Another class of optimization techniques involves pruning [45], [49], [50], which removes redundant or low-importance weights and neurons from a trained model. By eliminating unnecessary parameters, pruning can reduce both memory usage and computational load. Structured pruning approaches, which remove entire filters or channels, are particularly attractive for embedded deployment, as they lead to predictable reductions in execution time. However, pruning strategies must be carefully designed for time-series models, as aggressive parameter removal may degrade the model's ability to capture temporal dependencies.

Binary Neural Networks (BNNs) [51], [52] represent a more extreme form of model compression, in which weights and activations are constrained to binary values. This approach enables inference using simple bitwise operations, resulting in substantial reductions in memory footprint and energy consumption. While BNNs often exhibit lower accuracy compared to full-precision models, recent studies have demonstrated their feasibility for certain time-series classification and anomaly detection tasks, especially when deployment constraints are severe.

In addition to model-level optimizations, deployment toolchains play a crucial role in enabling Edge AI workflows. Frameworks such as TensorFlow Lite [53] and TensorFlow Lite Micro [54], [55] provide conversion and runtime support for deploying trained models on embedded devices. These toolchains offer functionalities for model quantization, operator optimization, and hardware-specific code generation, facilitating the transition from cloud-trained models to edge inference. Similarly, platforms such as Edge Impulse [56] provide integrated pipelines for data acquisition, model training, and deployment, targeting rapid prototyping on microcontroller-based systems.

Despite their benefits, existing deployment toolchains also present limitations.

Automated conversion pipelines may restrict model architectures to a subset of supported operators, limiting flexibility in model design. Moreover, end-to-end solutions often abstract away low-level details of memory allocation, scheduling, and execution, making it difficult to precisely control performance and resource usage. This lack of transparency can be problematic in safety-critical or highly optimized systems, where deterministic behavior and fine-grained control are required.

Overall, model compression techniques and deployment toolchains constitute essential enablers for Edge AI and TinyML applications. They allow complex time-series models to be adapted to constrained hardware platforms, balancing accuracy and efficiency. At the same time, their limitations motivate continued research into alternative execution platforms and hardware accelerators, such as FPGAs, which are discussed in the following section.

2.6 FPGA-Based Edge AI for Time Series Processing

Certain time-series applications impose requirements, such as deterministic latency, high-throughput signal processing, and strict energy efficiency, that motivate the adoption of hardware-accelerated Edge AI solutions. In this context, FPGAs are a promising platform for time-series inference at the edge [57]. FPGAs offer a fundamentally different execution model compared to CPUs and GPUs. By enabling custom hardware data paths and fine-grained parallelism, they allow computations to be tailored to specific algorithms and dataflows. This characteristic is especially advantageous for time-series processing, where repetitive operations such as convolutions, filtering, and recurrent updates can be efficiently mapped to parallel hardware structures. As a result, FPGA-based implementations can achieve low and predictable inference latency, which is difficult to guarantee on shared or multi-tasking processors.

Energy efficiency is another key motivation for using FPGAs in Edge AI systems. Unlike GPUs, which are optimized for high throughput but often incur significant power consumption, FPGAs can be configured to execute only the required

operations, minimizing unnecessary switching activity [58]. This makes them attractive for continuous monitoring applications, where models must run persistently over long periods [59]. In domains such as structural monitoring [60], energy systems [61], and transportation [62], this property can be critical for enabling always-on intelligence under strict power constraints.

Research on FPGA-based machine learning has explored a variety of time-series models, including 1D-CNNs [63], RNNs [64], GRUs [65], and hybrid pipelines combining signal processing with lightweight neural classifiers. High-Level Synthesis (HLS) tools, such as Vitis HLS [66] and Vivado [67], have significantly lowered the barrier to FPGA adoption by allowing designers to describe algorithms in high-level languages and automatically generate hardware implementations. Nevertheless, achieving efficient FPGA designs still requires careful optimization of data movement, memory access patterns, and parallelism, particularly when deploying deep learning models.

Despite their advantages, FPGAs also introduce notable challenges. Development workflows are typically more complex than those for microcontrollers or GPUs, and design iterations can be time-consuming. Toolchains may impose constraints on supported model architectures, numerical precision, and memory organization. Furthermore, integrating FPGA-based accelerators into broader Edge-to-Cloud workflows requires additional effort in terms of data interfacing, model management, and system orchestration [68].

From a system-level perspective, FPGA-based Edge AI occupies a complementary position within the spectrum of edge devices. While microcontrollers excel in ultra-low-power scenarios and GPUs offer high flexibility for complex models, FPGAs provide a middle ground where performance, energy efficiency, and determinism can be jointly optimized. However, much of the existing literature focuses on isolated accelerators or single-model deployments, with limited attention to their integration into end-to-end IoT data management and machine learning pipelines.

In summary, FPGAs represent a powerful but underexplored platform for time-series inference at the edge. Their ability to deliver deterministic performance and

high energy efficiency makes them particularly suitable for safety-critical and real-time applications. At the same time, their integration into Edge-to-Cloud workflows remains an open challenge, motivating further research at the intersection of hardware acceleration, data management, and deployable machine learning systems [68]. The following section reviews representative application domains that highlight these challenges and requirements.

2.7 Time Series Analysis in Application Domains

Time-series data constitute the primary source of information in a wide range of real-world IoT applications. While the underlying data management and processing principles are often shared, application domains differ significantly in terms of data characteristics, operational constraints, and performance requirements. Among the most extensively studied and practically relevant domains are human activity recognition (HAR) [69], structural health monitoring (SHM) [70], and automated driving (AD) [71], which collectively illustrate the diversity and challenges of time series-driven intelligent systems.

HAR, including sports and wearable-based scenarios, has long served as a benchmark domain for time-series analysis. In these applications, inertial sensors such as accelerometers and gyroscopes generate multivariate temporal signals that must be classified into discrete activities or gestures. Early approaches relied heavily on handcrafted feature extraction combined with classical machine learning classifiers. More recent work has demonstrated the effectiveness of deep learning models in automatically extracting discriminative temporal patterns [72]. Despite promising accuracy results, much of the literature focuses on cloud-based evaluation or offline analysis, often overlooking constraints related to real-time inference, energy consumption, and deployment on embedded devices.

SHM represents a markedly different but equally demanding application domain. In SHM, long and high-frequency time-series data are collected from distributed sensors installed on civil infrastructures, such as bridges or buildings, to detect anomalies, degradation, or damage. These signals are often noisy, non-stationary,

and strongly influenced by environmental factors. Recent studies have shown that deep learning models can effectively process raw vibration signals without extensive preprocessing. However, SHM systems are frequently subject to stringent requirements on reliability, interpretability, and continuous operation, making deployment considerations and energy efficiency as important as classification accuracy [70]. Moreover, SHM highlights the need for scalable data management solutions capable of handling large volumes of temporally dense measurements.

AD and driver monitoring systems further highlight the challenges of time-series analysis in safety-critical contexts [70]. These systems integrate heterogeneous temporal signals originating from vehicle sensors, driver monitoring cameras, and physiological measurements. Tasks such as driver distraction detection (DDD) or behavioral assessment require models that can operate with low latency and high robustness, as delayed or incorrect decisions may have severe consequences. While deep learning approaches have achieved state-of-the-art performance in controlled settings, their deployment in real-world vehicular architectures remains challenging due to computational constraints, certification requirements, and the need for predictable system behavior.

Despite the diversity of application domains, a recurring challenge lies in the integration of time-series data management, ML, and deployment under real-world constraints. While significant progress has been made within individual domains, existing approaches often remain fragmented across data handling, model design, and system deployment. This fragmentation motivates the need for unified methodologies that support reproducible data management and deployable ML pipelines across heterogeneous IoT applications. The following chapters build upon this background to present an integrated approach addressing these challenges.

CHAPTER 3

3. Enhancing the Measurify Framework

Building upon the background and related work discussed in Chapter 2, this chapter presents the methodological and architectural contributions developed in this thesis to enhance the Measurify framework for large-scale time-series data management and Edge-to-Cloud ML workflows. While existing solutions address individual aspects of IoT data ingestion, storage, or analytics, the goal of this chapter is to establish a unified and reproducible infrastructure that connects data collection, dataset management, and machine learning readiness within a coherent system.

Measurify was originally designed as a measurement-oriented framework for managing IoT data through a semantically structured abstraction of Things, Features, Devices, and Measurements. In the context of this thesis, the framework has been extended to support structured time-series datasets originating from heterogeneous sources, with particular attention to scalability, interoperability, and ease of use. These extensions directly address the limitations identified in the state-of-the-art analysis, such as the difficulty of integrating flat tabular datasets into semantically rich data models and the lack of transparent, reproducible data management workflows.

The first part of the chapter introduces a CSV-based workflow designed to simplify the ingestion and retrieval of large time-series datasets. This workflow enables users to upload heterogeneous tabular data without requiring advanced database expertise, while preserving semantic consistency through explicit mappings between CSV fields and measurement-oriented entities. The proposed approach has been validated through large-scale data reporting in energy-related IoT applications, where ingestion latency, resource usage, and scalability were systematically analyzed.

The chapter then presents the definition of a protocol-based mechanism for experiment configuration and metadata management. By formalizing experiment descriptors, metadata, and reporting structures, this mechanism supports reproducibility and consistent data organization across different use cases and application domains. The effectiveness and accessibility of the proposed workflows were further evaluated through a usability study involving users with diverse technical backgrounds, providing insights into the practicality of the framework in real-world settings.

Finally, the chapter introduces the Model Registry, a core component designed to manage datasets, algorithms, and trained models within a unified, version-controlled infrastructure. The Model Registry provides both graphical interfaces and programmatic APIs, enabling traceability and reproducibility across the ML lifecycle. This component establishes the basis for the ML developments presented in the subsequent chapters, where lightweight models are trained, deployed, and evaluated on constrained edge devices across multiple application domains.

3.1 Measurify

Measurify, previously known as the Atmosphere IoT Framework [73], is a cloud-based API for managing smart objects in IoT ecosystems. Developed at the Elios Lab of the University of Genoa and released as open source under the MIT license, the framework is abstract and measurement-oriented, with the concept of Measurement at its core. This abstraction makes it possible to address heterogeneous domains in a consistent way, providing a general foundation for handling data generated by connected devices.

From an architectural perspective, Measurify is built on top of MongoDB, a schema-less NoSQL database that stores measurements as JSON documents. The choice of MongoDB enables flexibility in handling heterogeneous data structures and scalability when managing large volumes of IoT data. The API layer, implemented in Node.js, connects to MongoDB through the Mongoose object data modeling library, which enforces schema validation, manages relationships

among data entities, and ensures consistency between the logical model and its database representation.

The framework exposes its functionalities via a set of RESTful APIs, allowing external applications and services to interact with stored data in a standardized manner. To further support usability and adoption, a companion Python library has been developed, providing data analysts and researchers with a high-level interface for uploading, querying, and downloading datasets without requiring in-depth knowledge of the internal API routes.

Beyond simplifying data upload and retrieval, this library also supports the development of external client-side scripts for application-specific workflows, such as data conversion, preprocessing, and feature extraction. These scripts are not part of the Measurify server itself, but can use the Python library to retrieve raw data from the platform and to upload processed results back into the system. This design keeps Measurify focused on data management, while preserving flexibility for domain-dependent processing pipelines.

This library integrates seamlessly with widely used environments such as Jupyter and Colab notebooks, thus enabling streamlined data analysis workflows.

At the core of the system are four essential resources: Thing, Feature, Device, and Measurement.

- A Thing represents the object of interest in the real world.
- A Feature describes the measurable quantity (e.g., temperature, acceleration, voltage).
- A Device is the instrument that performs the measurement.
- A Measurement records the values of a Feature measured by a Device for a specific Thing.

The Figure 3.1 shows the relationships among the resources, highlighting the central role of the measurement concept. As an example, it illustrates a simple scenario of collecting meteorological information (e.g., temperature and wind speed).

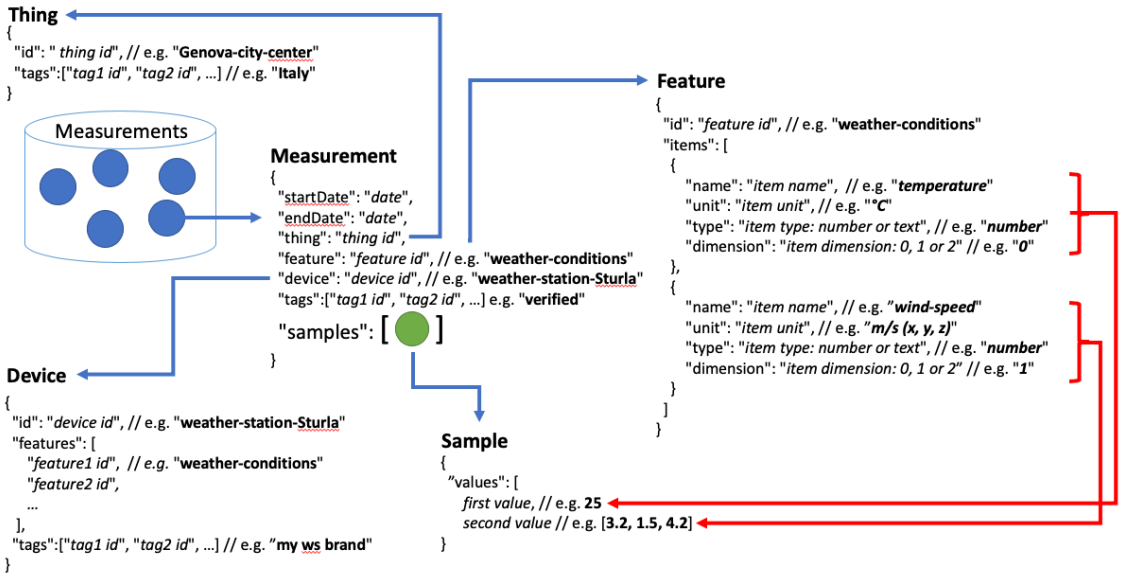


Figure 3.1: Measurify measurement concept

Each Measurement can itself be a scalar (e.g., ambient temperature), or a vector (e.g., three-axis acceleration). This flexibility allows Measurify to support a wide range of IoT signals, from simple environmental variables to more sophisticated sensor outputs.

To ensure integrity and consistency, the Feature resource validates every Measurement. Features define both the semantic meaning (e.g., “temperature,” “position,” “voltage”) and the structural attributes of the data. In particular, the attributes type and dimension must correspond to the structure of the associated samples: type indicates whether the values are numeric or string-based, while dimension specifies whether the data is scalar (0), vector (1), or tensor (2). This mechanism enforces schema validation at ingestion, preventing malformed or inconsistent data from being stored in the database.

This structure is designed to capture the semantics of IoT data while remaining general enough to support multiple application domains, from environmental monitoring to biomedical signals and industrial processes.

Beyond these core elements, Measurify also provides supplementary resources such as Users, Tags, and Roles, which enrich the framework with functionalities for user management and metadata annotation. This modular architecture allows researchers and developers to design workflows that are both flexible and

reproducible.

The guiding principle of Measurify is to reduce the complexity of dataset management in IoT contexts. By abstracting the underlying infrastructure while remaining open and interoperable, it enables a workflow where raw sensor data can be ingested, validated, stored, and later retrieved in forms suitable for analysis. In doing so, it provides the backbone for the next chapters of this thesis, where CSV-based ingestion, big data reporting, and domain-specific applications build upon this foundational architecture.

3.2 CSV-Based Workflow Extension for Time Series Management

Prior to the contributions of this PhD, the Measurify architecture was originally designed around JSON-based data ingestion, which alone proved to be a limitation in practice. In most IoT and ML domains, time series are collected, exchanged, and shared in tabular form, with the CSV file being the de facto standard for dataset representation. The absence of native CSV support therefore restricted interoperability with external data sources and created a barrier for researchers wishing to reuse existing datasets or integrate large-scale measurement logs into the framework.

However, working with CSV files is not straightforward. Unlike JSON, the CSV format is not formally standardized [26]. Different tools and domains adopt their own conventions for structuring tabular data, which can result in inconsistencies and errors during ingestion [27]. One of the most common sources of variation is the delimiter character: while commas are the recommended practice and most widely used, alternative delimiters such as semicolons, tabs, or vertical bars are frequently encountered. Other differences arise from file encoding (UTF-8,

Parts of this chapter has been published in Fresta, M. et al. (2023). Efficient Uploading of.Csv Datasets into a Non-Relational Database Management System. In: Berta, R., De Gloria, A. (eds) Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2022. Lecture Notes in Electrical Engineering, vol 1036. Springer, Cham. https://doi.org/10.1007/978-3-031-30333-3_2.
The material has been adapted and extended to fit the structure and scope of this thesis.

ASCII, or proprietary encodings), the rules for quoting or escaping text values, and the handling of special characters and line breaks. If these conventions are not clearly managed, compatibility problems can occur, leading to corrupted records or loss of information when datasets are shared across different systems.

For this reason, providing robust CSV support in Measurify required more than simply accepting tabular files; it demanded a well-defined workflow capable of mapping heterogeneous CSV structures onto the abstract model of Thing-Feature-Device-Measurement. This mapping ensures that values, timestamps, and metadata are correctly interpreted regardless of the source conventions, and that datasets remain reproducible and semantically valid once stored.

To address these limitations, support for CSV-based workflows was introduced in Measurify. The extension provides explicit mechanisms for importing and exporting tabular datasets while preserving the measurement-oriented data model of the framework. CSV files are mapped to the underlying MongoDB representation through a set of defined conventions, allowing datasets originating from public repositories, experimental logs, or external measurement systems to be ingested and managed in a consistent way.

The design of this extension is described in the following section, which presents the dataset loading module and the conventions adopted to handle heterogeneous CSV structures while ensuring correctness and scalability.

3.2.1 Designing the dataset loading module

To effectively support CSV-based workflows, it was necessary to design a loading module capable of handling heterogeneous datasets while preserving the Thing–Feature–Device–Measurement abstraction of Measurify. The key requirement was that the system should not only accept tabular files but also provide a clear mapping from the structure of each dataset to the internal resource schema of the database.

To achieve this, the design introduces a dual-file approach. Along with the raw CSV dataset, users must provide a companion JSON configuration file.

This description file acts as a map, telling the server how to interpret the dataset

columns and how to store them in the database. It establishes correspondences between CSV columns and Measurify resources, while also allowing the injection of additional context information that may be missing from the original dataset.

The JSON description file should not be interpreted as a validation mechanism by itself, but rather as a mapping specification that tells the server how the columns of the CSV file must be interpreted and associated with the Measurify data model. Based on this description, the server performs a sequence of consistency and validation checks before and during the ingestion process.

First, the server checks that the structure described in the JSON file is coherent with the structure of the CSV file, in particular with respect to the expected number and position of columns. The server then verifies that the referenced Feature exists in Measurify and that the number and type of the values associated with that Feature are compatible with its definition. Since each Feature specifies the expected dimensionality of the measurement and the type of its items (e.g., numeric or textual), this step ensures that the CSV content can be consistently mapped to the target schema. If some positions are intentionally left empty in the JSON file, the corresponding slots can be preserved as empty values, provided that the overall structure remains coherent with the Feature definition.

The server also checks the existence of the referenced Thing and Device resources. If they are missing, the upload request returns an error, unless the user explicitly enables their automatic creation through the corresponding “force” option in the POST API request. Once these preliminary checks have been completed, the rows are processed individually. At this stage, each row is validated against the expected structure and types, so malformed records, inconsistent values, or rows containing textual and numerical data in the wrong positions can be selectively rejected without discarding the entire file.

Therefore, the role of the JSON file is to define the mapping between the CSV dataset and the Measurify resource model, while the server uses this information to enforce structural coherence, semantic consistency, and row-level validation during ingestion.

Upon processing the request, the server returns a report indicating which rows

were stored successfully and which failed due to format or mapping inconsistencies.

The code below provides an example of such a description file, in which each Measurify entity required for a Measurement is linked either to a CSV column index or to a fixed string value. The file has a key-value structure mapping each Measurify entity either to a number (representing the corresponding column number in the csv, starting from column 1) or string value, which is assigned to all the measurements in the dataset.

{ "thing":1,
"feature":2,
"items":{ "feature_A":[3],
"feature_B":[4],
"feature_C":[3,4]
},
"tags":[5,6],
"device": "control-unit"}

In this case, the first column of the CSV specifies the Thing related to each measurement, while the second column provides the Feature. Depending on the actual feature value, measurements may have different dimensionalities: “feature_A” and “feature_B” are one-dimensional, while “feature_C” combines the values in columns 3 and 4, making it bi-dimensional. The configuration also shows how multiple columns can be grouped as array entities (e.g., two tags in columns 5 and 6). Finally, fixed values can be assigned across the entire dataset when certain mandatory entities are missing: here, the Device field is set to “control-unit” for all rows. This approach ensures flexibility and completeness even for datasets that lack explicit identifiers for all Measurify resources.

3.2.1.1 Unit test dataset

For development and validation purposes, a simple unit test was carried out with a CSV file containing a single measurement spread across four columns (Figure 3.2).

thing-1	tag-1	device-1	10

Figure 3.2: Unit test measurement

To give meaning to this line, the description file was generated:

{"thing":1,

"tags":[2],

"device":3,

"feature":"feature-1",

"items":{ "feature-1":[4]}}

After processing the mapping, the server correctly stored one measurement of type feature-1, with one item whose numerical value was 10. This minimal example demonstrated the correctness of the mapping logic and the ability of the system to enforce schema consistency even on trivial datasets.

3.2.2 Results

The flexibility of the loading module was tested with real-world datasets from different domains. Two representative cases are presented below to illustrate its ability to accommodate datasets with distinct structures.

3.2.2.1 Smart City

The first dataset comes from the air quality monitoring system of the municipality of Bologna, Italy, provided by ARPAE Emilia Romagna [74]. It contains pollutant measurements collected in real time by three monitoring stations.

In this dataset (Figure 3.3), the last column specifies the pollutant type, which naturally maps to the Feature resource.

987792	2022-01-01T00:00:00	GIARDINI MARGH.	35	PM10
988241	2022-01-01T00:00:00	GIARDINI MARGH.	2	O3
989200	2022-01-01T00:00:00	VIA CHIARINI.	17	NO2

Figure 3.3: Three lines of Bologna.csv dataset

The first column contains a unique identifier for each measurement, while the penultimate column reports the measured concentration in µg/m³. These two values are grouped as the items of the feature: the first represents the actual measurement, while the second is its identification code. The timestamp column is used for startdate, and the third column indicates the Thing, in this case the location of the monitoring unit. Since there is no dedicated Device column, a fixed value ("Monitor_Bo") was assigned to all rows.

The corresponding JSON configuration file is:

{ "feature":5,
"items":{ "PM10":[4,1],
"O3":[4,1],
"NO2":[4,1]},
"startdate":2,
"thing":3,
"device":"Monitor_Bo"}

This mapping illustrates how heterogeneous CSV structures can be systematically interpreted and ingested by Measurify without altering the original dataset, while still conforming to the Thing-Feature-Device-Measurement abstraction.

3.2.2.2 Biomedical dataset

The second dataset was taken from the biomedical domain, containing cardiac measurements of several patients along with demographic data. Unlike the Smart City dataset, this file provides only numerical values with no explicit mapping to higher-level entities. It was therefore chosen as a representative case for datasets consisting almost exclusively of items.

An excerpt of the dataset (Fig. 3.4) shows a header followed by rows of patient measurements.

age	sex	cp	tre	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	1

Figure 3.4: Header plus four lines of Heart.csv dataset

Since contextual information such as Thing, Device, and Feature were missing, these had to be added explicitly in the configuration file.

{ "thing":"Patients",

"device":"Heart_monitor",

"items":{ "heartValues":[1,2,3,4,5,6,7,8,9,10,11,12,13,14]},

"feature":"Heart_disease_indicators" }

Here, all 14 columns of the CSV are mapped as item values under the feature “heartValues”. The Thing and Device were added as fixed strings, while a descriptive Feature name was assigned to capture the meaning of the dataset. This example shows how the loading module enables the ingestion of datasets with minimal structure, enriching them with the metadata needed to fit into the Measurify schema.

In summary, the CSV workflow extension provides a domain-independent mechanism for ingesting heterogeneous datasets into Measurify while preserving the abstraction of Thing-Feature-Device-Measurement.

3.3 Large-Scale Validation of the CSV Workflow

This section evaluates the CSV-based ingestion workflow introduced in the previous section through a large-scale experimental validation. The workflow was applied to hundreds of publicly available datasets from online repositories, covering heterogeneous domains such as health, environmental monitoring, energy, and demographics.

The evaluation focuses on two main aspects: the robustness of the ingestion mechanism when applied to real-world datasets with diverse structures, and its performance when compared to existing database ingestion tools. In particular, the workflow is benchmarked against mongoimport, the command-line utility provided by MongoDB for bulk data loading.

Unlike mongoimport, which offers limited validation and no support for semantic enrichment, the proposed workflow enables controlled ingestion, error handling, and the integration of tabular data into the measurement-oriented model of Measurify. The results presented in this section assess the effectiveness of the approach beyond a proof-of-concept scenario.

3.3.1 Workflow and Architecture

Fig. 3.5 shows the CSV-based ingestion workflow integrated into Measurify, from dataset submission to validation and storage.

Parts of this chapter has been published in M. Fresta, A. Capello, F. Bellotti, L. Lazzaroni, M. Cossu and R. Berta, "Supporting a .csv-based Workflow in MongoDB for Data Analysts," 2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE), Helsinki, Finland, 2023, pp. 1-4, doi: 10.1109/ISIE51358.2023.10228044.
The material has been adapted and extended to fit the structure and scope of this thesis.

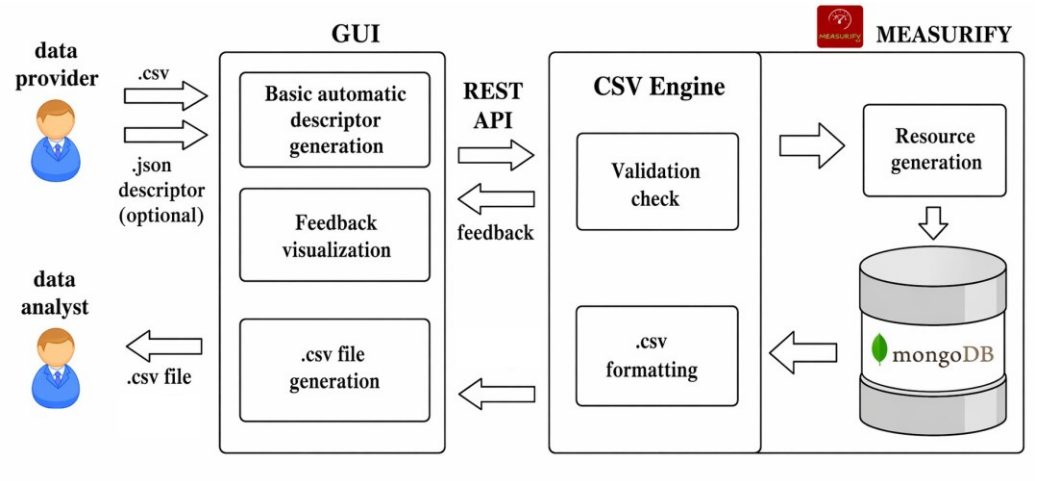


Figure 3.5: The Measurify-based .csv workflow proposed

The process starts with the data provider, who supplies the raw .csv dataset together with an optional .json descriptor file. If no descriptor is provided, the system supports a basic automatic generation to assist the user.

These inputs pass through a graphical user interface (GUI), which offers two main functionalities:

- Feedback visualization, so that users can see in real time which rows have been correctly processed and which have failed;
- CSV file generation, which allows analysts to export validated and formatted datasets.

The GUI communicates with the CSV Engine through the REST API. The engine performs a validation check, ensuring that every row in the CSV is consistent with the schema specified by the descriptor. If inconsistencies are detected (e.g., malformed rows, missing values, invalid types), the engine returns detailed feedback to the GUI.

After validation, the data undergoes CSV formatting and are mapped into the Measurify resource model. The resource generation module then stores the validated measurements into MongoDB, ensuring that the tabular input is converted into structured resources aligned with the Thing-Feature-Device-Measurement abstraction.

Finally, the data analyst can access the enriched dataset via the GUI, download it in a desired format, or directly exploit the resources within Measurify for further

processing and analysis.

To illustrate the workflow in practice, the following example shows how a CSV dataset and its companion JSON description file are jointly used to upload measurements into Measurify. Each row of the CSV is validated against the corresponding Feature to ensure type and structural consistency; duplicates are discarded unless explicitly allowed by the user, and a report summarizing successful and rejected insertions is returned at the end of the operation.

{ "thing":1,

"feature": "feature_A",

"items":{ "feature_A": [2, 3, 4] },

"device": "device_A",

"startdate": 5,

"tags": [6]}

In this case, column 1 identifies the Thing, all measurements belong to the Feature “feature_A”, and their items are found in columns 2-4. A single Device (“device_A”) is associated with all rows, while the timestamp comes from column 5 and the Tag from column 6. This simple case demonstrates how a flat CSV table can be seamlessly integrated into Measurify while retaining the semantic structure required for downstream applications.

3.3.2 Experiments

We validated the approach on a large set of real-world datasets. We randomly sampled from data.world [75] 312 datasets as representative of different application fields (e.g., health, atmosphere, energy, census). This popular website provides a vast free collection of quality-ensured datasets, in tabular formats such as csv, xls, xlsx.

The first step analyzed the compatibility of datasets with the actual .csv format. Excel spreadsheets that could be exported in .csv through the Excel native tool or the Pandas Python library were included. We consider a minor tweak the need for removing the first/last rows of a .csv file, that are often used for comments; we consider a major rework the need for an actual data format change (i.e., .csv-

named files featuring a structure incompatible with the .csv format). Statistics are reported in Table 3.1.

Table 3.1: Statistics on datasets

Category	**Number**	**Percent**
CSV compliant files	149	77%
CSV files requiring minor tweaks	38	19%
CSV files requiring major rework	7	4%
Excel files exportable to CSV	51	43%
Excel files requiring minor tweaks before export to CSVs	0	0%
Excel files requiring major rework before export to CSVs	67	57%

The analysis suggests that the lack of Excel files requiring minor tweaks is because of the fact that one or more sheets are dedicated to comments, while the others do not have any. We also notice that the majority of Excel files have a structure incompatible with the .csv format, typically because of the hierarchical indexing.

All files requiring minor tweaks or major reworks were excluded from the next steps of the experiment. In the second step, the suited Excel files were converted to standard .csv through a Python script. As an Excel spreadsheet may contain several sheets, the resulting collection consists of 210 .csv files, with a total of 9,103,077 records.

Selected files are then uploaded to Measurify, through a simple Python script that automatically builds the required description file from the CSV header (assuming that all datasets are homogeneous, as in the example in section III – in the third step of this experiment we will analyze more refined assumptions). Results are reported in Table 3.2, first column. Time performance was measured for a Measurify installation on a Dell Inspiron with a i7 11th Gen Intel CPU, 16GB of RAM and RTX 3050Ti laptop GPU.

Table 3.2: Statistics on uploaded records

Item	**Measurify**	**mongoimport**
Uploaded records	9,085,545	9,100,401
Rejected records	17,532	2,676
Overall success on uploaded records	99.81%	99.97%
Files with 100% of records uploaded	91.37%	93.33%
Files with 99+% of records uploaded	95.43%	93.33%
Files with 95+% of records uploaded	96.95%	93.33%
Files with 75+% of records uploaded	100%	93.33%
Files with 0% of records uploaded	0%	6.67%
Average latency per row	173 µs	28 µs

The results show that 99.81% of measurements were correctly uploaded. The cause of rejection is the presence of anomalies that could affect the integrity of the data, leading to incorrect conclusions and decisions. Two types of anomalies are relatively common: presence of duplicate measurements (we ignore implicit indexing with the number of the row, in this case, even if Measurify provides this option) and inconsistent data types in a column. Mongoimport allows a user to load .csv files directly to internal collections. Data type validation check is optional and requires that the header of the file contains the column name concatenated with data type.

Table 3.2 shows that the optimized Measurify workflow still has a higher raw ingestion latency than mongoimport, with an average latency per row of 173 µs compared to 28 µs, corresponding to a factor of about 6.2. However, this difference should be interpreted in light of the different purposes of the two tools. While mongoimport is designed for fast bulk import, Measurify performs row-level validation, semantic mapping to the measurement-oriented data model, and duplicate checking during ingestion.

These additional operations introduce an overhead, but they also make the workflow more robust and flexible in practical data integration scenarios. In particular, a single error in a file may cause mongoimport to reject the entire file, whereas Measurify can preserve valid rows and successfully uploaded at least 75% of the records in every tested file. By contrast, mongoimport discarded

6.67% of the files entirely. This trade-off reflects the different intended uses of the two systems: mongoimport is a backup-oriented import utility, whereas Measurify is a framework for application development and deployment, where correctness, traceability, and semantic consistency are primary requirements. It should be noted that the results reported in Table 3.2 are based on a later revision of the published article, aimed at optimizing the code and making the solution competitive with Mongotool.

The third step of the experiment intended to quantitatively assess the value of the semantic extension provided by the Measurify resources (Measurement, Thing, Device, Tag), and check its compatibility (given the provision of an additional descriptor file) with the .csv plain tabular format. Analyzing the provided datasets, we identified four common patterns compatible with the Measurify structure.

A first, quite common, pattern concerns datasets containing Independent and Identically Distributed (IID) values consisting of measurements with a set of columns that could be used to filter data or train ML algorithms. Table 3.3 gives an example of this pattern.

Table 3.3: IID example

Food	**Calories**	**Protein (g)**	**Type**
Carrot	30	1	Vegetables
Potato	110	3	Vegetables
Apple	130	1	Fruit

In this example, the first and the last columns define a food's name and type and could be interpreted respectively as Thing and Tag. All other columns are the values of a Measurement of the relevant Thing. The corresponding JSON descriptor needed to upload on Measurify the .csv file in a semantic-enhanced way is:

{"thing": 1,
"feature": "NutritionalValues",
"items": {"NutritionalValues" : [2,3]},
"tags": [4]}

Table 3.4 reports another example of this first pattern, corresponding to the results of a questionnaire.

Table 3.4: Questionnaire example

Person_ID	**Question A**	**Question B**	**Question C**
1542	True	A	1
7528	True	B	7

The associated JSON descriptor is shown below:

{"thing": 1,

"feature": "Usability questionnaire",

"items": {"Usability questionnaire" : [2,3,4]}}

A second common pattern is the time series (an example is shown in Table 3.5).

Table 3.5: Time series example

Date	**Electric Production (MWh)**
01/01/1985	72.50
02/01/1985	70.67

These datasets contain a series of data points indexed in time. The corresponding JSON is:

{"thing": "Power_plant_Civitavecchia",

"feature": "Electric_prodution_per_day",

"items": {"Electric_prodution_per_day" : [2]},

"startdate": 1}

A third pattern is the typical IoT schema, in which different Things are monitored by a specific device to collect values of a physical signal (e.g., Table 3.6).

Table 3.6: IoT application example

Room	**Device**	**Temperature (°C)**
Bedroom	TermometerA	19.6
Bedroom	TermometerB	19.5
Bathroom	TermometerC	18

The associated JSON descriptor:

{"thing": 1,

"device": 2,

"feature": "Temperature",

"items": {"Temperature": [3]}}

A fourth, quite common pattern is the trivial combination of the second and third ones into an IoT time-series.

The application of these semantic enhancements is seamlessly managed by Measurify, thus the performance values reported in Table 3.2 remain unchanged.

Overall, the experimental results demonstrate that Measurify is able to ingest heterogeneous CSV datasets while enforcing semantic consistency and data quality and without requiring user-side modifications. In the evaluated experiments, the framework successfully processed more than 9 million time-series samples originating from 210 CSV files, with an average ingestion time of approximately 173 µs per row.

During ingestion, each measurement was explicitly validated, enabling automatic detection and removal of duplicated records as well as selective rejection of malformed or inconsistent rows. This row-level validation introduces an overhead when compared to native database import tools, which ingest the same datasets in approximately 28 µs per row without performing any semantic checks.

Despite this additional processing, the achieved throughput remains compatible with large-scale data integration workflows. These results quantitatively show that Measurify reaches ingestion performance comparable to state-of-the-art CSV import mechanisms, while simultaneously providing guarantees on data integrity, traceability, and semantic enrichment. This trade-off between ingestion speed and data quality is particularly relevant for time-series and IoT applications, and motivates the use of the proposed workflow in the application-oriented studies presented in the following chapters.

3.4 Energy Applications of Measurify

Following the validation of the CSV-based workflow, Measurify was applied to the energy domain as a representative application scenario. This domain is characterized by the continuous generation of large-scale time series data and by heterogeneous measurement sources, making it a suitable testbed for assessing the framework under realistic conditions.

In this section, Measurify is evaluated through three real-world energy use cases, ranging from vehicular battery monitoring to power distribution systems and smart home appliances. These applications illustrate how the same measurement-oriented abstraction can be reused across different contexts to support domain-specific services such as range prediction, power flow analysis, and appliance scheduling.

3.4.1 Energy data characterization in the three target domains

To apply Measurify to the energy domain, the first step consisted in characterizing the data sources and measurement domains involved. This analysis was required to identify the structure, semantics, and temporal properties of the time series generated in each scenario, and to assess their compatibility with the Measurify data model.

To this end, an analysis of publicly available datasets was carried out, supported by a targeted review of the relevant literature. Based on this study, three representative energy domains were selected: vehicular batteries, smart grids, and building management systems.

3.4.1.1 Vehicular batteries

Among the market available batteries for electric vehicles (EV), lithium-ion batteries feature high energy density, high power density, long service life, fast

Parts of this chapter has been published in Dhungana, H.; Bellotti, F.; Fresta, M.; Dhungana, P.; Berta, R. Assessing a Measurement-Oriented Data Management Framework in Energy IoT Applications. Energies 2025, 18, 3347. https://doi.org/10.3390/en18133347.
The material has been adapted and extended to fit the structure and scope of this thesis.

charging, lower self-discharge rate, and environmental friendliness and thus have found wide application in the area of consumer electronics [76]. State estimation of batteries in EVs is essential to optimize energy management strategy, extend the life cycle, reduce cost, and safeguard a safe application of batteries in EVs [77]. The main purpose of a battery management system (BMS) is to ensure safe and optimal use of the battery energy and provide accurate battery state information for the EV's overall energy management. Alongside this, the BMS should have the capability to give proper interventions for the battery system if it is working in an abnormal condition such as over-charging or over-discharging. Overcharging of batteries can bring safety issues and generate chemistry degradation and short circuits that lead to fire and gas hazards. In a similar way, the over-discharging can damage the battery and shorten its lifetime by more than 50%. Precise battery measurement focuses on smaller unit measurement millivolt and milliamp accuracy and time synchronizing these voltages and current [78]. Precise and reliable battery state estimation is a critical tool for boosting the lifespan of the battery packs from 10 years to 20 years in the best case and generally result in a 30% lifetime improvement [79].

BMSs are well-established in portable electronic devices like cellular phones and notebooks. However, the implementation of BMS in EVs is still in the early stage because the number of cells in an EV is a hundred times higher than on portable devices. EV battery banks are designed to provide high voltage, current, and power, which makes BMS more complex than portable electronics. Figure 3.6 shows a typical EV BMS. The functional module consists of hardware as well as software components to collect data. The hardware structure consists of many kinds of sensors, actuators, controllers, cloud storage, and signal lines, whereas the software perspective contains measurement data parameters, states calculating algorithm (analytics tools) and methods, user interface (visualization), fault detection, and monitoring mechanism. Typical measured data for BMS consists of bus voltage, total current, cell voltage, temperature detection, and smoke detection. After sampling and preprocessing, measured data is forwarded to the control circuit as well as the cloud server.

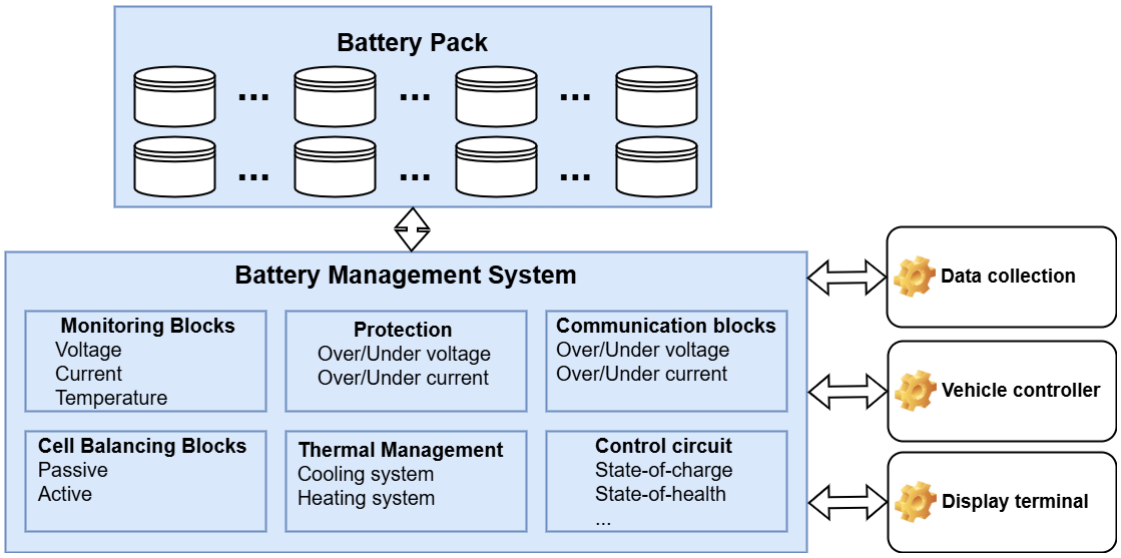


Figure 3.6: General function of a battery management system.

3.4.1.2 LV test feeder

A LV feeder is a part of the electrical distribution system that delivers power from a distribution transformer to end-users, such as homes, businesses, or small industrial facilities. Structure of a low-voltage distribution grid connected to the medium-voltage grid through a transformer is presented in Figure 3.7. Smart meters are used as a sensor to measure household or small commercial loads. The sensor measures the energy usage data and forwards it to cloud storage. The measurements collected from an LV feeder are then used to feed services for power flow analysis. The LV test feeder is a radial distribution system operating at a base frequency of 50 Hz [80]. It is connected to the medium voltage network via a substation transformer, which steps down the voltage from 11 kV to 416 V. Both the main feeder and its laterals operate at 416 V.

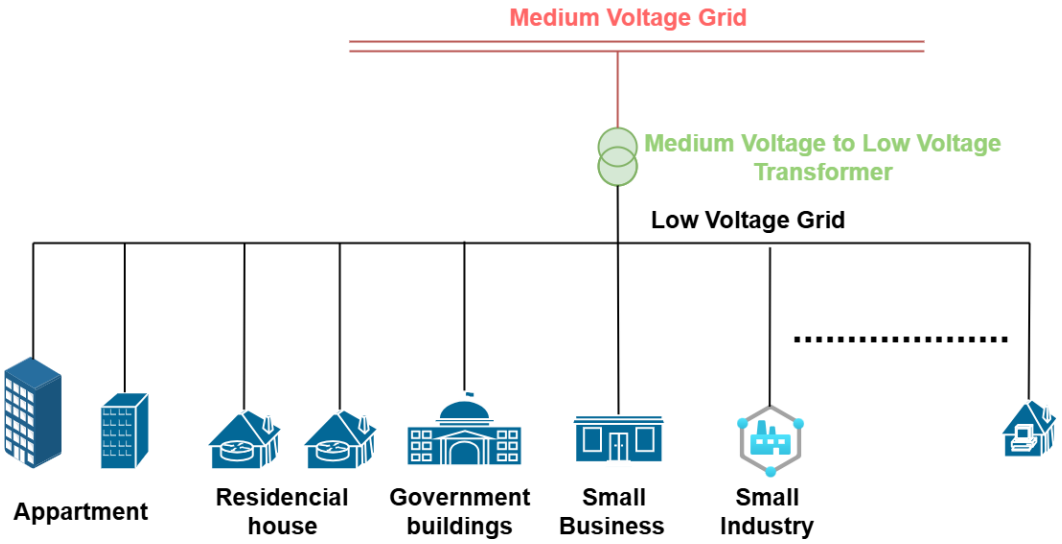


Figure 3.7: The low-voltage grid supplies electricity to various consumer types, including apartments, residential houses, government buildings, small businesses, and small industries.

Data collected from the LV feeder distributor can be used to generate insights on market trends, consumer behavior, system health, and critical alarms. By leveraging those datasets, various researchers modeled real-world scenarios, optimized grid efficiency, and enhanced voltage stability [81], [82]. These analyses are crucial to improving the reliability of the distribution system, integrating renewable energy sources, and ensuring the smooth operation of the feeders. Interactive information sharing between feeder distributors and consumers allows better scheduling and energy management. Among those services we choose power flow analysis because it enables the simulation of voltage drops, power losses, and load distribution in feeder networks, helping engineers assess system performance under different operating conditions.

3.4.1.3 Home energy management system (HEMS)

A HEMS is a smart system that monitors, controls, and optimizes energy usage in a household. It integrates various devices such as smart meters, appliances, renewable energy sources, energy storage systems, and demand-side management strategies to improve energy efficiency and reduce costs [83]. HEMS typically uses automation, data analytics, and machine learning to schedule appliance

operation, balance power consumption, and take advantage of time-of-use electricity pricing. It can also enhance grid stability by coordinating energy consumption with supply fluctuations. According to Ruano et al. [84], HEMS can be managed in four stages. The first stage collects data, such as electrical readings (current, voltage, and power) measured by smart meters/acquisition boards or specific sensors. In the second stage, namely the event detection, any change in the state of an appliance over time is noticed through thresholding based on historical data. In the third stage, feature extraction of each appliance is performed based on the load signature. Finally, in the fourth stage, the load is identified by using the extracted features.

The introduction of smart metering produces fine-grained energy consumption data, allowing for the extraction of more valuable information about energy demand and production compared to having only an energy balance over an extended period of time. A precise demand-response functionality is expected to benefit from a much finer granularity of information, such as appliance load monitoring [85]. The appliance load monitoring focuses on implementing detailed energy sensing and delivering information on the breakdown of the energy spent. There are two major appliance load monitoring approaches, namely Intrusive Load Monitoring (ILM) and Non-Intrusive Load Monitoring (NILM), based on the number of sensors used. ILM, also referred to as distributed sensing, performs different activities such as occupancy detection, behavior monitoring, and contextual sensing. NILM, on the other hand, senses only the power consumption of each appliance as a single measurement. The ILM method is more accurate in measuring appliance-specific energy consumption, but less widespread because of practical difficulties, such as multiple sensor configuration, high costs, and complex multimodal analysis and decision-making.

A typical NILM data collection framework for HEMS is depicted in Figure 3.8 Electrical appliances such as computers, ovens, and washing machines are monitored using sensors or meters for electrical quantities (e.g., current, voltage, power). These measurements are influenced by parameters like sampling rate and accuracy. Data is transmitted via communication protocols (e.g., ZigBee, MQTT,

HTTP) to a central server, stored in various file formats (e.g., CSV, HDF5, WAVE), and used for HEMS analytics including appliance scheduling and load forecasting.

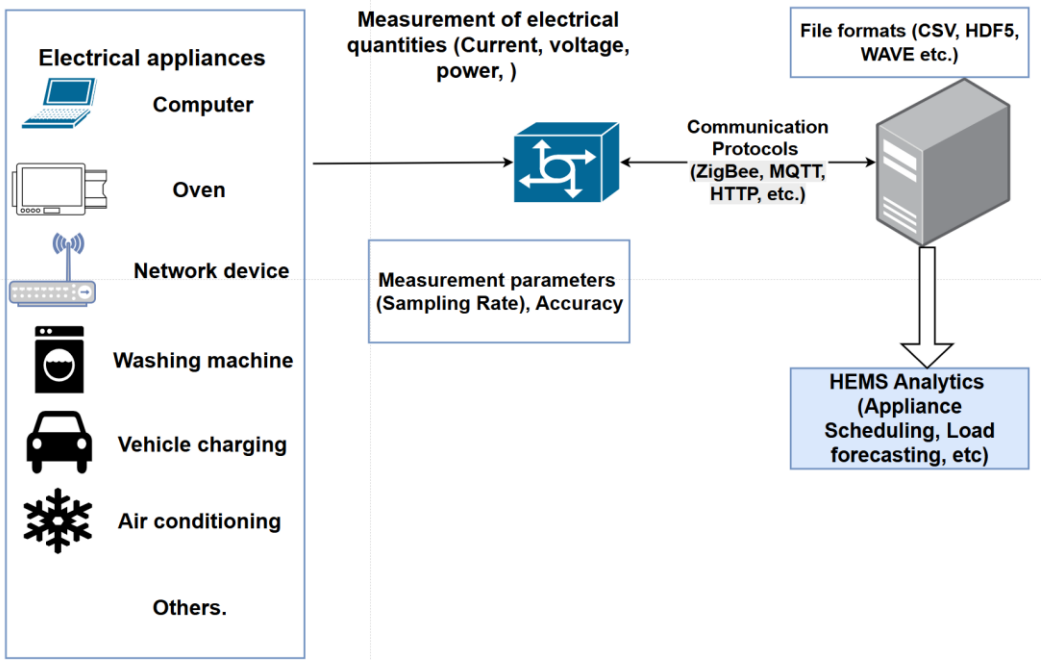


Figure 3.8: Overview of a Home Energy Management System (HEMS) architecture for appliance-level monitoring and analytics

The HEMS should offer advantages to both residential occupants and electricity suppliers for better exploitation of the power sources. For electricity suppliers, two-way communication allows a better organization of the network and the implementation of several mechanisms known as demand response to dynamically control load based on capacity analysis, emergency demand response, and supplementary service analysis by suppliers. Nowadays, dynamic electricity retail tariffs are commonly used to support renewable energy integration, allowing residential occupants to reduce their electricity costs by scheduling appliance usage efficiently.

3.4.2 Synopsys

To consolidate the characterization of the three domains described above, we provide a synoptical overview through Tables 3.7 and 3.8. First, Table 3.7 provides an outlook of the three energy IoT services, where the metamodeling illustrates inputs, outputs, data processing algorithms, stakeholders, benefits, and constraints. Then, Table 3.8 summarizes the main data characteristics and its mapping to Measurify resource types.

Table 3.7: Synopsys of the investigated IoT services.

	Range prediction from BMS data	**Power flow analysis from LV feeder data**	**Appliance scheduling from HEMS data**
Goal of service	Predict travel distance from remaining SoC	Analyze feeder power losses	Recommend optimum hours for using high-power appliances
Inputs	Environmental, vehicle and battery data	Load data	Power data
Output	SoC, Driving range	Power loss	Optimum schedule
Computation algorithm	Data-Driven Method for range prediction	Newton-Raphson method	Descriptive analytics and cost optimization techniques
Service stakeholders	Driver, battery supplier and vehicle manufacturer	Power distributor	Residential customer, Power supplier
Benefits	Drivers can easily plan destinations and safe charging zones	Optimize energy distribution	Electricity cost reducing for residents
Constraint	Accurate range prediction requires minimum uncertainties in modeling battery limits, vehicle dynamics, environmental factors, and driving behavior.	Association between load dynamics and renewable energy integration	Some appliances must run immediately based on user needs

Table 3.8: Characteristics of IoT services and mapping with Measurify resources.

	Range prediction from BMS data	**Power flow analysis from LV feeder data**	**Appliance scheduling from HEMS data**
Things	Battery cell and driving context	LV test feeder	Electrical appliance in house
Devices	Battery measurement emulator (ISL78714) other sensor	Smart meters	Plugwise Kit
Measurements	Environmental data, Vehicle data, Battery data	Feeder load of nodes	Power consumption data
Features	Temperature, elevation, speed, throttle, voltage, current, SoC	Load	Active Power
Sensitivity / accuracy of measurement	Device accuracy: ±2.5mV	Voltage measurement error smaller than 0.1%	Measurement reading ±0.5W
Representation of data	Time-series representation in CSV format	Time-series representation in CSV format	Time-series representation in CSV format

The widespread use of IoT for data measurement, combined with the long-term accumulation and large-scale analysis of such data, requires careful definition of sampling rates (typically ranging from a few Hz to several kHz) to avoid both redundancy and the loss of transient, potentially valuable events. Heterogeneity across sensor data is frequent, and can be due to different data sources, network and spatial distribution across the monitored area. These factors introduce synchronization challenges and variability in data collection intervals, especially in dynamic environments or under unreliable network conditions. Such inconsistencies, caused by different sampling rates, node failures, packet loss, or measurement precision, can compromise data quality and hinder effective analysis. To address this, Measurify assigns an individual timestamp to every single sample within a measurement, ensuring accurate temporal alignment across heterogeneous data streams. Moreover, each measurement is linked to its source device, which allows spotting and taking into account individual effects (e.g., drift).

3.4.3 Experimental result and Analysis

The experimental evaluation of the three energy IoT services assesses the reliability and applicability of the Measurify framework in managing heterogeneous sensor data across various domains. To ensure consistency in latency measurements and eliminate variability caused by network transmission, all tests were performed by deploying a local instance of Measurify directly on the host machine. This set-up avoids the influence of external communication delays, which depend on specific network configurations and hardware, and cannot be generalized a priori. All experiments were carried out locally on a mid-range laptop equipped with an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz CPU, 16 GB of RAM, and an NVIDIA RTX 3050 Ti Laptop GPU. This hardware configuration represents a balanced and accessible development environment, suitable for evaluating lightweight IoT middleware like Measurify. In addition to performance metrics, Table 3.9 summarizes the structural characteristics of the datasets used in the three domains, reporting the number of files, columns, total rows, and total file size. This comparison highlights the variety of the datasets and the scalability of the framework.

Table 3.9: Overview of dataset structure across the three application domains

Domain	**# Files**	**CSV Columns**	**Total Rows**	**Total Files Size (MB)**
Vehicular Batteries	33	38	157.114	48
LV Test Feeder	100	2	144.000	3
HEMS	231	10	18.288.288	1.400

Using Jupyter Notebooks [86], we performed automated data uploading of the datasets (batch upload), and measured the average latency per file. Additionally, Docker Desktop [87] was used to measure CPU and RAM utilization for each domain. Table 3.10 reports the average latency and resource usage observed during the experiments.

Table 3.10: Measured latency and system resource usage for each IoT service (batch upload)

Application	**Avg. Latency per row (µs)**	**Avg. CPU Usage (%)**	**Avg. RAM Usage (MB)**	**Application**
Vehicular Batteries	189	5.4	775	Vehicular Batteries
LV Test Feeder	92	3.8	675	LV Test Feeder
HEMS	115	6.7	3100	HEMS

As shown in Table 3.10, latency per row correlates strongly with the number of columns per file. For example, the vehicular batteries dataset, despite its relatively modest size in rows, reached the highest latency due to its 38-column structure. In contrast, the LV test feeder dataset, with only 2 columns per file, had the lowest latency. However, in all tested domains, the measured latencies remain within the range of 90-190 microseconds per row, aligning well with the requirements of real-time processing scenarios.

RAM usage was mostly affected by the total number of rows, particularly in the HEMS dataset, which required more than 3 GB of memory. This is a natural consequence of bulk uploading and processing large time-series data. Nevertheless, this remains acceptable compared to the capacity of a 16 GB RAM computer. CPU usage remains low and relatively constant in all applications, never exceeding 7% on average, confirming the lightweight nature of the Measurify architecture.

Beside testing real-time reception of the data, we also performed streaming tests by sending single measurements at controlled frequencies of 1 Hz, 10 Hz, 1,000 Hz, and 10,000 Hz. Given Measurify's fast response time (in the order of a few hundred microseconds), no performance degradation nor data loss was observed, even at the highest frequency. Detailed results are omitted since performance metrics were comparable to those in Table 3.10. These results further confirm the suitability of the framework for real-time monitoring, where handling of high-frequency and low-latency data is essential.

Overall, the experimental results quantitatively validate the applicability of Measurify to energy-related IoT services characterized by heterogeneous data structures and large-scale time-series datasets. Across the three evaluated domains, the framework successfully managed datasets ranging from 144,000 rows (LV Test Feeder) up to more than 18 million rows distributed over 231 CSV files in the HEMS scenario, with a total data volume of approximately 1.4 GB.

Measured ingestion latencies remained consistently low, between 92 µs and 189 µs per row, depending on dataset dimensionality. In particular, the results confirm that latency scales primarily with the number of columns per file rather than with the total dataset size. Despite the large variability in dataset structure, average CPU utilization remained below 7% in all cases, while memory usage ranged from approximately 675 MB to 3.1 GB, even for the largest bulk-upload scenario. These figures demonstrate that Measurify can process large energy-related time-series datasets using the resources of a standard development machine.

In streaming experiments, the framework reliably handled input frequencies from 1 Hz up to 10,000 Hz without data loss or observable performance degradation. This confirms that the measurement-oriented architecture supports both batch ingestion and high-frequency real-time data streams. Taken together, these results show that Measurify achieves a favorable balance between scalability, low latency, and moderate resource usage, making it suitable for real-time and production-grade energy IoT applications.

3.5 Protocol and Experiment: Big Data Reporting in the Hi-Drive Project

In the previous sections, Measurify was extended with CSV ingestion and validated across heterogeneous datasets and energy use cases. The next step was to evaluate these capabilities in an industrial-scale scenario, where requirements extend beyond data storage to include structured experiment configuration, metadata management, and automated reporting.

This opportunity was provided by Hi-Drive project, funded under the European Union's Horizon 2020 programme. With a budget of approximately 30 million € and a consortium of more than 40 partners, Hi-Drive aims to extend the operational design domain (ODD) of automated driving functions (ADFs) and reduce takeover requests by leveraging enablers such as connectivity, positioning, cyber-security, and machine learning. The large-scale, cross-country experiments conducted withing the project generate massive volumes of vehicular and contextual data, which must be systematically transformed into progress indicators for project management.

To address these requirements, Measurify was extended with two new resources, Protocol and Experiment, forming the Experimental Metadata Database (EMDB). This extension supports structured experiment configuration, automated tracking of experiment execution, and the management of more than 330 performance indicators per experiment, enabling a homogeneous and transparent reporting process for project stakeholders.

This section presents the design and deployment of the EMDB within the Hi-Drive project, illustrating how Measurify evolves from a flexible data ingestion framework into a big data reporting solution for large-scale collaborative research initiatives.

Parts of this chapter has been published in Capello, A.; Fresta, M.; Bellotti, F.; Haghighi, H.; Hiller, J.; Mozaffari, S.; Berta, R. Exploiting Big Data for Experiment Reporting: The Hi-Drive Collaborative Research Project Case. Sensors 2023, 23, 7866. https://doi.org/10.3390/s23187866.
The material has been adapted and extended to fit the structure and scope of this thesis.

3.5.1 Specifications and Requirements

This section describes the reporting data specifications and the design requirements that were defined during the early phases of the Hi-Drive project. The data specifications were established by the project's Methodology sub-project, also based on the experience in previous projects. The design requirements were collected and then harmonized in a page on the project's online collaboration tool, where partners could detail one or more specific needs in natural language and provide short user stories to illustrate applications and expected benefits. In a user-centric design perspective, this list of requirements was further complemented with the feedback from early lab tests and demos with partners, which informed useful design iterations.

3.5.1.1 Data Specifications

For specifying the data needs, a schema was defined in the initial phase of the project to make all the experiments report their state in a quantitative and consistent way, considering the variety of settings across the different experiments.

The schema is encoded in an Excel sheet (an excerpt of the file is provided in Table 3.11), which gives detailed structure and specification (until the level of the type of the single fields) of the signal list (i.e., the data expected to be stored in the EMDB). The sheet has one row for each unit of data (namely, indicator) quantifying the project/experiment progress in a particular aspect. In a hierarchical approach, which was deemed appropriate to deal with their huge number (330+) and variety, indicators are grouped into 35 topics.

Examples of topics include the following:

- Duration of the experiments, which contains, as related indicators, the number of tests lasting up to 15 min, between 15 and 30 min, 30 and 60 min, etc.;
- ADF operation times, which includes the number of activations with operation time less than 1 min, between 1 and 5 min, between 5 and 10 min, etc., and the number of encountered traffic sections (e.g., pedestrian crossings,

roundabouts, traffic lights, tunnels);

- Travelled distance and driving times, which are in turn segmented in terms of the time of day (e.g., morning, afternoon, evening, night), type of road (motorway, urban, rural, etc.), road conditions, number of lanes, speed limits, state of the ADF and enabler (activated or not).

Table 3.11: Excerpt of data requirements Excel file.

Covered Information		**Description**
Group	**Specific Value**	
Time of tests in the experiment	Between 0 and 6 Between 6 and 12 Between 12 and 18 Between 18 and 24	Number of test runs performed between hour x and hour y, local time.
Duration of tests in the experiment	Less than 15 min Less than 30 min Less than 1 h Less than 1 h and 30 min Less than 2 h Longer than 2 h	Number of test runs performed during the indicated length of time.

A key requirement emerged from the large number of indicators that had to be periodically reported according to the defined data specifications: the automation of indicator extraction. Since providing all these data manually at each reporting period would be impractical, scripts were developed to automatically extract most of the indicator values from the raw experimental files.

3.5.1.2 User Requirements

The main requirement that emerged during the initial analysis and design steps was system configurability, especially in terms of experiment descriptors and indicators.

Support for configurability was intended to facilitate dealing with possible extensions during the actual development, and also allow the possibility of using

the tool in other domains as well. To this end, abstraction was a key design factor. For instance, we generalized the concept of Experiment as a generic project activity, so that it could be applied to other contexts as well, without sticking to domain-specific characteristics.

Other requirements emphasized the importance of user-friendliness and, thus, demanded the availability of a graphical user interface supporting all the data management operations according to the different typologies of users involved. Overall, project partners required support for understanding the system (e.g., through hands-on workshops) and during the actual operations.

Requirements mandated that all the graphical user interfaces (GUIs) be implemented as cloud-based applications accessible from common browsers, to prevent the burden of local executable installations (avoiding platform-dependent compatibility issues and security locks in a company's PCs). This ensures that all the partners can seamlessly employ the latest version of the application. On the downside, this approach limits the interface flexibility and modalities of local file accessibility, but this did not compromise overall effectiveness.

Data integrity is obviously a key requirement. As we will see later, this is targeted through the utilization of schemas that mandate the structure of each datum and the employment of a suitable design pattern (particularly, the "Protocol-Experiment" pattern).

Data confidentiality concerns the fact that experiment managers should be able to upload and access data about their own experiment, but not those of the others, while project managers should be able to read data from all the experiments.

The functioning of the EMDB server itself does not imply strict performance requirements, given the relatively limited amount of data (in the order of MBs) to be uploaded by each experiment manager (the expected number of experiments is 20/30) and downloaded by project managers. However, generalizing the specific use case, the solution is big data ready, being based on MongoDB, a popular (particularly, involving a large, active community of users and rich documentation), high-performance, state-of-the-art non-relational DB system [88], [89]. On the other hand, huge quantities of data are processed by the toolchain at

each experiment's site, for the automatic extraction of the progress/performance indicators from the project's raw files.

3.5.2 System Design and Implementation

In Hi-Drive, the Experimental Metadata Database (EMDB) was realized by instantiating Measurify as an application database.

While the base configuration was sufficient to support the ingestion and organization of measurements, the Hi-Drive use case introduced additional requirements: systematic experiment tracking, metadata management, and periodic reporting of more than 330 indicators. To meet these needs, Measurify was extended with two new resources (Protocol and Experiment), which together provide the abstraction needed to describe experiments, their context, and their temporal evolution.

These resources implement the Experiment-Protocol pattern, conceptually analogous to the existing Measurement-Feature pattern used for measurements. In practice, Protocol defines the structure of descriptors, metadata, and indicators, while Experiment instantiates this structure and maintains the history of an experiment over successive reporting steps. This extension allows experiments to be described in a consistent, project-wide manner while preserving flexibility for future adaptations.

As already described in Section 3.1, Measurify enforces data integrity through the Measurement-Feature pattern, where each measurement is validated against the schema of its associated Feature. This principle provides the foundation for the Hi-Drive extensions, which adopt a similar abstraction at the experiment level.

3.5.2.1 Extensions to Measurify

The requirements for the EMDB were encoded in an Excel file (Table 3.11) , which specifies the structure of the data to be stored. Most fields are single values (numeric or string), while some indicators (e.g., velocity distributions) require arrays.

From an end-user perspective, each experiment is described by three categories of information:

- Descriptors, i.e. basic fields common to all experiments (name, location, managing organization, start/end date).
- Metadata, project-specific context information (e.g., targeted enablers such as V2I, positioning, machine learning).
- History, the dynamic progress indicators that are periodically updated during the experiment (e.g., number of tests per time of day, experiment length, distance travelled with ADF enabled).

Descriptors and metadata define the static properties of an experiment, while history entries capture its evolution over time. Each history record is indexed by an ordinal Step (1, 2, …), which can represent months, quarters, or other reporting intervals. This allows flexible chronologies of experiment states, ensuring consistency across sites.

To formalize this structure, a new Measurify resource called Protocol was introduced. Each experiment must be associated with a protocol, which defines the descriptors, metadata, and indicators it will contain. In Hi-Drive, a single protocol ("Hi-Drive") was sufficient, but the abstraction supports multiple customized protocols within the same project if needed. Together with the Experiment resource, this defines the Experiment-Protocol pattern, conceptually analogous to the Measurement-Feature pattern, but applied at experiment level.

History records are hierarchically structured into Topics and Fields, mirroring the taxonomy defined in the specification file. For example, the Road Condition topic is subdivided into fields such as Road_dry, Road_wet, Road_icy, Road_snowy. The complete Hi-Drive protocol is stored in a JSON file ("Protocol_Hi-Drive.json"), accessible only to administrators. Experiment managers interact with it through forms or CSV templates automatically generated according to the protocol (Figure 3.9).

```
{
    "name": "Hi-Drive",
    "description": "Hi-Drive protocol",
    "metadata": [
        { "name": "Experiment_Focus_TechEnablers", "description": "Experiment focus on technical enablers", "type": "scalar"},
        { "name": "Experiment_Focus_Operations", "description": "Experiment focus on vehicle operations", "type": "scalar"},
        { "name": "Experiment_Focus_Users", "description": "Experiment focus on user study", "type": "scalar"},
        …
],
    "topics": [
        {
            "name": "Road condition",
            "description": "Number of tests with road condition",
            "fields": [
                { "name": "Road_dry", "description": "Dry road", "type": "scalar"},
                { "name": "Road_wet", "description": "Wet road", "type": "scalar"},
                { "name": "Road_icy", "description": "Icy road", "type": "scalar"},
                { "name": "Road_snowy", "description": "Snowy road", "type": "scalar"}
            ]
        },
        …
    ]
}
```


Figure 3.9: Excerpt from the Hi-Drive Protocol specified in .json.

Each experiment is configured through a JSON file (e.g., "Experiment_Italy1.json") specifying its descriptors and metadata, with an initially empty history (Figure 3.10).


```
{
    "name": "Italy1",
    "description": "Description of the experiment being held in Genova",
    "state": "ongoing",
    "startDate": "2022-06-01",
    "endDate": "2023-05-31",
    "manager": "MyCompany",
    "location": "Genova",
    "protocol": "Hi-Drive",
    "metadata": [
        { "name": "Experiment_Focus_TechEnablers", "value": 1 },
        { "name": "Experiment_Focus_Operations", "value": 0 },
        { "name": "Experiment_Focus_Users", "value": 0 }
        …
    ],
    "history": []
}
```


Figure 3.10: Excerpt of Experiment Italy1.json.

The file "Italy1#step2.csv" (an excerpt of which is reported in tabular form in Table 3.12), instead, is an example of a possible History step record that could be uploaded by an experiment manager. The example reports the number of test runs conducted in the second step of the experiment, segmented by road condition. Through a convention on filenames, the toolchain is able to detect the proper experiment's history to update.

Table 3.12: Example excerpt of a History step for the Italy1 experiment.

Field Key	**Field Value**
Road_dry	14
Road_wet	4
Road_icy	0
Road_snowy	2

As a summary (Figure 3.11), the information to be provided for each experiment has been
mapped following these criteria:

- Very basic information about the nature of the experiment is mapped to the Experiment Descriptor fields. This data structure is common to all the experiments/projects;
- More detailed information about the nature of the experiment is mapped to the Experiment Metadata. This data structure is experiment specific;
- The indicators periodically reporting the progress of the experiment are mapped to the Experiment History record. This data structure is experiment specific.

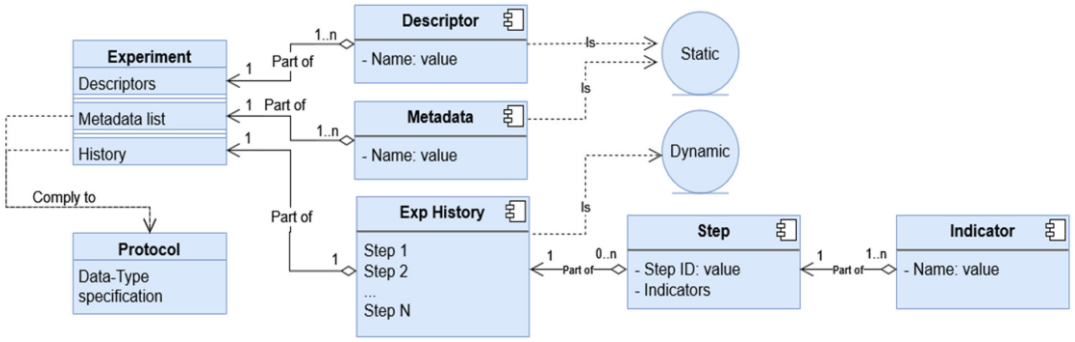


Figure 3.11: Organization of the data structure for an experiment.

Finally, it is important to highlight the generality of the proposed design, since Descriptors, Metadata and History are concepts applicable to virtually any type of experiments/projects, and the Protocol resource abstraction allows this flexibility to be implemented with intrinsic data integrity checking.

3.5.2.2 The EMDB Workflow

According to the Measurify model [73], the workflow consists of two main phases:

1. Configuration, which sets up the specific application database;
2. Operation, which fills it during the progress of the application.

In the first configuration step, the EMDB administrator creates the Protocol resources in the EMDB. In the Hi-Drive case, there is only one resource, which is instantiated by uploading the already presented "Protocol Hi-Drive.json" file. Then, each Experiment manager defines his own Experiment in a .json file and uploads it to the EMDB ("Experiment Italy1.json" is a configuration file example, shown in Figure 3.10). The EMDB configuration steps are depicted in Figure 3.12.

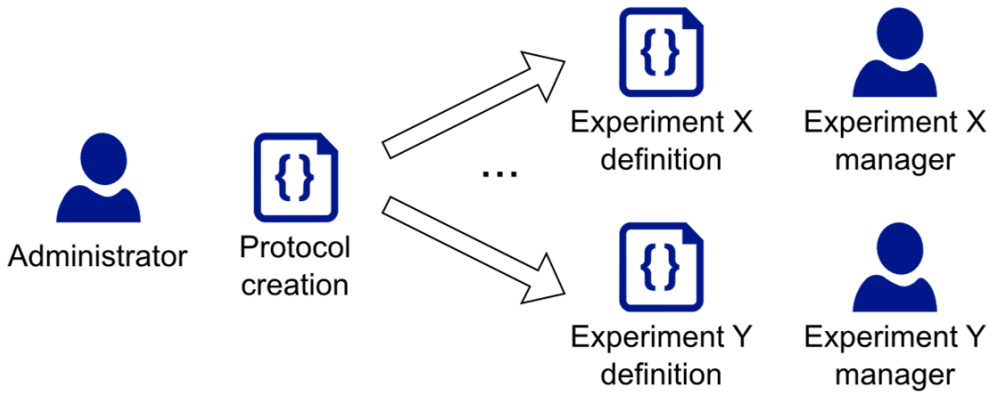


Figure 3.12: Steps for the creation and initialization of the EMDB.

Once configured, the EMDB is ready for receiving data from the various test sites. Each experiment manager can report the state of the experiment at a given history step by uploading a file (e.g., "Italy1#step2.csv" shown in Table 3.12) through the Operation Tool, a graphical user interface which will be described in the next subsection. Then, project managers can retrieve the state of all the Experiments present in the application database. Experiment managers are allowed to update and retrieve data about their Experiment(s), while Project managers are allowed to retrieve data of all Experiments. Figure 3.13 provides an overview of the operation phase of the EMDB.

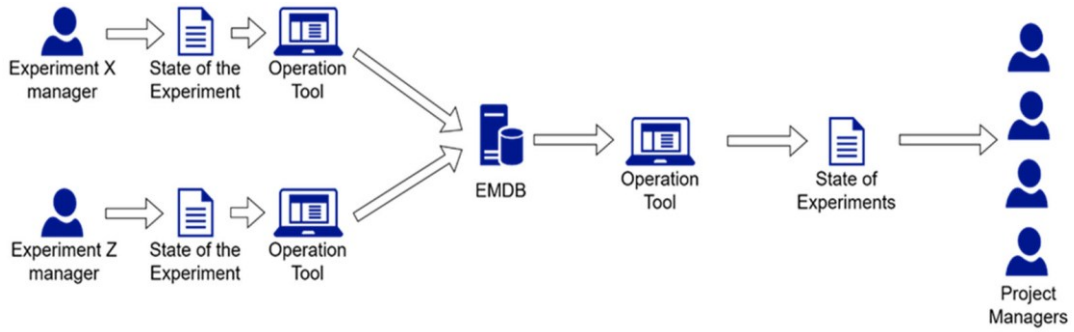


Figure 3.13: Typical steps for the operation phase of the EMDB.

3.5.2.3 Graphical User Interface

Given the relevance of the partners' requirements, special attention was paid to the graphical user interface (GUI). The GUI consists of three tools that are available as simple web applications on browsers: the Admin Dashboard, the Operation Tool and the Data Visualizer.

The Admin Dashboard has been designed to support DB administrators to create and initialize (i.e., configure) their ADBs. The Admin Dashboard supports creation and configuration of all the Measurify resources, including Protocols and Experiments (e.g., Figure 3.14). Thus, the .json files needed for configuring of an EMDB installation (including the single experiments) can be set up by filling in fields in a visual form, if preferred by the user.

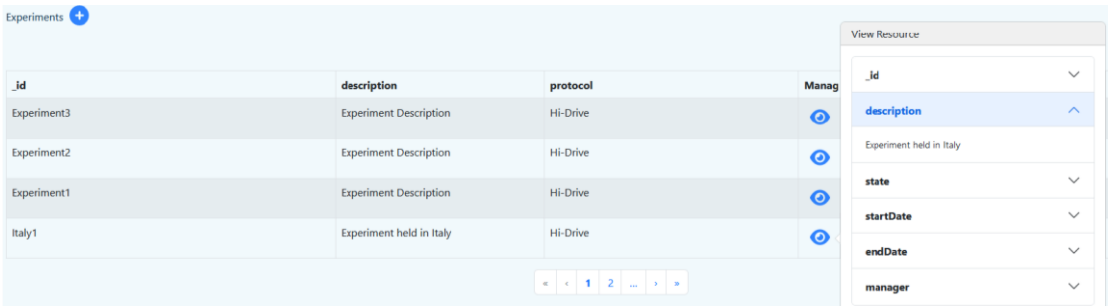


Figure 3.14: Outlook of the Admin Dashboard page for managing experiments.

The Operation Tool is the tool through which the various DB users can perform the typical data upload and download operations to/from an ADB. The tool is designed to support the operations of the workflow presented in Figure 3.13.

The Data Visualizer offers a user-friendly way for experiment and project managers to quickly obtain a graphical overview of the EMDB data. The GUI supports bar- and pie-type interactive charts. The bar chart displays the absolute values of each item within the selected data group (a data group is one of the 35 topics), while the pie chart shows their relative values as percentages. If a data group consists of a single item, the charts will be generated for different steps of the experiment instead of different items within the group. These tools have been developed using state-of-the-art JavaScript libraries for building advanced user interfaces, such as React [90], Vue.js [91], and Chart.js [92].

3.5.2.4 Automatic Extraction of the Progress/Performance Indicators

As per the requirement of reducing the manual effort needed to fill in the periodic reports to be inserted in the EMDB, we designed Python scripts to extract the progress/performance indicators.

These scripts are external to the Measurify server and are not automatically triggered when new data are uploaded or when the dataset is updated. Rather, they are manually developed and executed on the client side according to the specific requirements of the application. Their role is to process source data that may not be directly supported by the platform, such as HDF5 files, or to extract higher-level indicators and statistical features before storing them in the system. To support this workflow, the scripts can exploit the dedicated Python library developed for Measurify, which allows them to retrieve raw data from the platform and to upload processed results back into the database. This separation is intentional: Measurify is designed primarily as a data management framework, while preprocessing and feature extraction are kept external in order to preserve flexibility and avoid embedding application-specific processing logic into the core server.

Extraction is achieved by processing the .hdf5 files that log the timeseries of the signals obtained by the vehicular sensors (e.g., speed, pedal activity, ADF activity, etc.), that are enriched in post-processing with derived/context measures (e.g., detected driving scenarios, relative position of other traffic actors, type of road,

weather conditions, etc.) [93]. Since one .hdf5 file is created for each test session (i.e., test-vehicle's trip), the script that takes in input a directory containing all the .hdf5 files recorded in the reporting period and outputs the .csv file to be uploaded to the EMDB, with most of its fields automatically filled. The script associates a counter to each fillable indicator. Some of these counters (e.g., number of tests on wet roads) are simply incremented by one for each .hdf5 file meeting the relevant criterion. Other counters (e.g., number of take-over requests) are incremented per each .hdf5 file by the value obtained by processing the relevant timeseries in that file. A few other indicators, which are not related to values logged during a trip (e.g., number of trip participants, age, etc.), cannot be computed automatically and are left to be manually filled by the experiment manager.

Figure 3.15 provides an overview of the periodic reporting process. The experiment manager collects all the relevant files in a root directory, executes the automation script, checks the resulting .csv file, manually fills the missing values, and finally uploads the file to the EMDB via the Operation Tool on a browser.

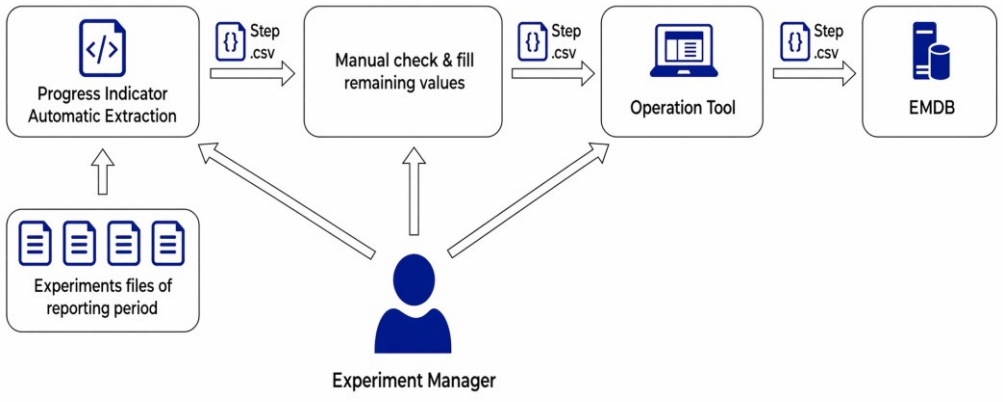


Figure 3.15: Periodic report preparation process.

3.5.2.5 Lab Testing

The system architecture has been implemented with a systematic exploitation of automatic unit testing to aim at achieving robustness and guarantee that each version release could meet the requirements. This is particularly important in languages which are not strongly typed, such as JavaScript, so properties or

methods of variables may depend on inputs. Thus, the scope of static verification is limited, and actual checks should be performed at runtime. Unit tests have been prepared for all the functionalities provided by the APIs, and new releases are made only after all unit tests have been successfully passed. Testing of the server functionalities can be performed through the GUI or directly through API testing routines directly built into Measurify, which is the way employed for the automatic unit testing.

The client's GUI has been verified in lab tests and discussed in demos with the relevant partners. Automatic unit tests have been set up also for the GUI, using the Cypress.io tool [94], which proved itself very useful as an independent test runner that does not need to closely integrate with source code. Through an interactive web page accessible from the browser, the developer can visually execute sequences of actions on the GUI under development. Each one of such sequences is a unit test that is recorded by Cypress.io to be later employed in automatic testing.

3.5.3 Results and Discussion

3.5.3.1 Deployment

After lab validation, the EMDB was deployed and tested in pre-pilot conditions, with the goal of verifying usability and performance under realistic project scenarios. As is common in research projects, design iterations were guided by feedback from partners using early versions of the system. To this end, demos and hands-on workshops were organized from the early phases, supported by tutorials that helped participants familiarize themselves with the workflow.

Then, a pre-pilot phase was organized to deploy and test the systems to verify their overall functioning before the actual execution of the tests that will produce the final project results. Feedback is requested by all the partners during the pre-tests in order to allow the system to achieve the desired level of service. Pre-tests concern 20 operational sites.

Two solutions are requested and implemented for the deployment of the Hi-Drive

DBs: (i) in the cloud; (ii) on premises. In both cases, users are able to manage data through the previously presented GUI. The former solution is particularly suited for project-level DBs but could also be used for EMDB instances available to single partners only. To this end, the Hi-Drive cloud space on Amazon Web Server (AWS) [95] was set up, hosting an EC2 t2 medium machine dedicated to the pre-pilot EMDB with Ubuntu operating system, which can be scaled according to further requirements.

The on-premises EMDB solution has been thought of typically for private installations, in which partners can have local tests with their own data, before sharing them at consortium level in the cloud. Moreover, they may try different local DB configurations (e.g., with new Features implementing different data types), for instance, for exploring possible solutions that could be later extended to the whole consortium. This requires that partners download the source code and proceed with the installation of the server and then the configuration of the EMDB, as described in the previous section.

The actual experience in the pre-pilots is showing a very limited use of on-premises deployment, while the cloud solution, in which installation is managed by the project-level DB-administration team, is more convenient, also for single-partner EMDBs. Pre-pilot tests have been very useful in allowing partners to gain practical experience with all the modules developed during the project and thus provide concrete suggestions on how to improve their usability. For the EMDB, this implied several tweaks in the GUI design (e.g., position and text of the widgets, consistency among the web pages, feedback messages to the user, content of the tutorial/user manual web pages).

A major outcome from the pre-pilot tests concerns verification of efficiency both in deployment and usage. This concerns both the experiment set-up time and the history update time. The developed system allows experts to focus on experiment description by defining the two lists of metadata and of progress indicators as .csv files. The definition of the corresponding data types (Protocol resource, one for the whole Hi-Drive project) is straightforward. Once the files are ready, the EMDB is instantiated in few seconds.

Table 3.13 gives a summative quantitative characterization of an EMDB cloud installation targeted for Hi-Drive. Table 3.14 shows the performance results of this EMDB installation (also in a stress case with 8 h of vehicular signal recordings, instead of the standard case of 4).

Table 3.13: Quantitative summary of an Hi-Drive EMDB cloud installation.

Item	**Value**	**Notes**
# protocols	1	The Hi-Drive protocol only
# experiment descriptors	7	Static values set at the initialization
# metadata	88	Static values set at the initialization
# history step indicators (total)	334	
# history steps to be filled manually	41	
# experiments	20	
# history steps	18	
# hdf5 files	~40	Per experiment and history step
.hdf5 file size	~55 MB	Average, for each file
Reporting frequency	Monthly	
Cloud machine	AWS EC2 t2.medium	2 vCPU, 4 GB RAM
Upload/download speed	250–300 MBit/s	Nominal bandwidth
Upload size	~8 KB	Per experiment and history step
Download size	~500 KB	All experiments and steps
Time to upload one history step	<1 s	By an Experiment manager
Time to download all steps for one experiment	~1 s	By a Project manager
Secure protocol	https	

Table 3.14: Test results for EMDB standard and stress usage for a single experiment.

	Standard	**Stress**
# Data source files (.hdf5) (per experiment and step)	40	40
Signal sampling frequency	10 Hz	10 Hz
# Hours per experimental run per day	4	8
# Samples per day	144,000	288,000
# Vehicular signals sampled	21	21
Single .hdf5 file size	55 MB	110 MB
Total size of files to be processed (per experiment and step)	~2.2 GB	~4.4 GB
Progress indicator extraction time (per experiment and step)	153 s	310 s
Time to fill manual indicators	~10 min	~10 min
Time to upload one history step	<1 s	Same as standard case
Time to download all steps for one experiment	~1 s	Same as standard case

Currently, the amount of data to be uploaded at each periodic reporting step by each experiment manager is in the order of KBs. The download, instead, may involve history data from several sites and time steps, thus in the order of hundreds of MBs. Consequently, the EMDB server, serving about 20 different users, can be safely hosted on a medium-end cloud machine.

On the other hand, more resources are demanded for the computation of the progress/performance indicators from the project's data. But this processing can be carried out using the computational infrastructure set up at each test site for the scientific and technical goals of the project (i.e., processing the vehicular data logged to assess performance of the ADFs enhanced through the newly proposed technological enablers).

It is apparent the reduction in data size from the source data files (in the order of the GB, Table 3.14) to the extracted KPIs (in the order of the KB, Table 3.13).

Results in Table 3.14 reveal that our solution efficiently handles the potentially large file sizes, allowing a rapid extraction of the needed indicators. Tests were performed on a laptop equipped with an Intel Core i7 11800 H @ 2.30 GHz processor and 16 GB of RAM.

Progress indicator extraction automation was key to reduce the reporting time and thus make the overall process acceptable for the experiment managers, who have to deal with more than 300 indicators for each reporting step. Otherwise, the number of indicators would have had to be drastically reduced, and the process would have been more error prone. Overall, these results confirm that the EMDB workflow can transform large volumes of raw vehicular data into compact sets of key performance indicators with minimal manual effort. The reduction from gigabytes of input files (Table 3.14) to kilobytes of structured outputs (Table 3.13) demonstrates both the efficiency and the scalability of the system, making periodic reporting feasible even in large collaborative projects.

3.5.3.2 Instantiating the EMDB in Other Projects

The Measurify system's code is freely available online (https://github.com/measurify/server/, accessed on 31 October 2025). The /gui sub-folder of the repository contains the code for the Admin Dashboard and of the Operation tool. All the resources are provided with instructions for installation and use. From the deployment experiences, the computing requirements for running Measurify are relatively modest. The framework does not require GPU acceleration for its core data-management functionalities and can be deployed on standard CPU-based machines. The configurations reported in this thesis, such as the AWS EC2 t2.medium instance with 2 vCPUs and 4 GB of RAM for the Hi-Drive cloud deployment, or a standard laptop with 16 GB of RAM for large bulk-ingestion experiments, should be regarded as comfortable reference configurations that ensure very good performance. However, simpler configurations can also be sufficient for smaller installations, development setups, or moderate workloads. Overall, these observations indicate that Measurify can be adopted without specialized hardware, while still scaling to heterogeneous and data-intensive IoT

scenarios when more resources are available.

In this sub-section, we summarize the steps necessary for third parties to instantiate their own EMDB:

1. Define the static information needed to describe each project's experiment (i.e., the Metadata).
2. Define each experiment's static information values (i.e., the values of the experiment Descriptor and Metadata).
3. Define the dynamic information needed to describe each project's experiment (i.e., the Progress Indicators).
4. Define one or more Protocols to specify the data types of Metadata and Indicators.
5. Create an empty EMDB installation by using the Admin Dashboard.
6. Encode the Protocol(s) in a .json file (Figure 3.9), or through the Admin Dashboard GUI, without directly editing the file. Insert the protocols into the EMDB.
7. Instantiate the project's Experiments, specifying their Descriptor, Metadata, and referenced Protocol by uploading to the server the Experiment's .json file (Figure 3.10). This step can also be performed through the Admin Dashboard GUI, without directly editing the file.
8. Develop the script to automatically extract performance indicator values from the project/experiment's data files. This step is optional, and completely project specific. It is time consuming, but highly beneficial to reduce the final reporting time.
9. Use the Operation tool for uploading and downloading the project's history steps (Figures 3.12 and 3.15, if the system includes scripts for automatic extraction of the indicators, as mentioned in the previous step).

It is important to highlight that the big data aspect of the system is project specific, so scripts need to be developed ad hoc to process the project data file (step 8 in the bullet list above). However, the EMDB may be used not only for data-intensive experiment reporting, but also for projects where the indicators for a work-package or task could be quickly manually filled on the .csv template.

The deployment within the Hi-Drive project shows that the extended Measurify framework can support periodic reporting in large, multi-partner settings. The system managed 20 experiments across multiple sites, processing gigabyte-scale vehicular data (up to 4.4 GB per reporting step) to extract more than 300 structured performance indicators, reducing the final reporting payload to the order of kilobytes. Upload and download operations consistently required less than one second. These results confirm the scalability and practical usability of Measurify for industrial research projects.

3.6 Usability Study

After validating the CSV workflow across heterogeneous datasets and demonstrating its applicability in large-scale projects such as Hi-Drive, the next step was to evaluate the usability of the system in a more controlled educational environment. While technical validation confirms correctness, scalability, and efficiency, the adoption in real-world contexts also depends on how easily end-users can understand and operate the framework. In collaborative projects, users are not always software developers; they can be engineers, researchers, or managers with different levels of technical expertise. For this reason, it is crucial to assess whether the workflow can be effectively learned and applied by non-expert users. To this end, a dedicated laboratory session was organized, involving 45 students with backgrounds in electronic and strategic engineering. The aims of this laboratory were: first, to measure how quickly and accurately participants could perform the main operations of creating the JSON description file and uploading to Measurify datasets in CSV format; and second, to collect structured feedback on the overall user experience, including the clarity of the workflow and the usability of the graphical tools.

The test was designed to simulate a realistic working scenario in which users receive data files and are asked to prepare them for ingestion into the system. In this context, participants had to bridge the gap between raw tabular data and the structured resource model adopted by Measurify (Thing, Feature, Device, Measurement). By observing how users approached this task, the experiment

provided valuable insights into the intuitiveness of the system, the effectiveness of the learning material, and the main sources of difficulty encountered.

In addition to quantitative metrics (e.g., completion times, error rates), the laboratory also served as a channel for qualitative observations. Students were encouraged to highlight confusing aspects, suggest improvements for the graphical interface, and reflect on the practicality of the approach. This feedback is especially important because it informs not only the refinement of Measurify itself but also the design of future training materials and tutorials to support adoption in broader contexts.

3.6.1 Laboratory setup

The laboratory session was carried out in a classroom setting with 45 participants, each equipped with a workstation and guided through a short introduction to the Measurify framework. The goal of this preliminary briefing was to provide only the essential concepts, ensuring that students could rely on the documentation and the graphical tools rather than on step-by-step assistance. This simulated the conditions of a typical deployment scenario, in which new users are expected to become familiar with the workflow with minimal training.

During the introduction, participants were presented with the key principles of Measurify's resource model (Thing, Device, Feature, and Measurement) and with the role of the CSV-JSON workflow in mapping tabular datasets to this structure. The descriptor file (JSON) was highlighted as the central element of the workflow, since it links the raw CSV columns to the appropriate Measurify entities.

An annotated template of the descriptor file was provided as a reference:

{

"thing":,

"device":,

"items":{

"":[]

},

"tags":[],

"startdate":"12/05/2023",

"enddate":"12/05/2023",

"feature":

}

The hands-on part of the lab consisted of three structured exercises, each simulating a realistic application domain:

1. Questionnaire dataset: Students were given a CSV file with survey results. Their task was to create a descriptor file where the participant ID was mapped as the Thing, the survey answers were mapped as Items of a "Questionnaire" Feature, and a fixed Device value was assigned to all records (Figure 3.16).

IdPerson	Question1	Question2	Question3	Question4	Question5	Question6	Question7	Question8	Question9	Question10
3152	A	A	B	A	A	C	A	D	D	D
3689	B	C	B	A	B	D	B	D	B	A
1185	B	C	B	A	B	D	B	D	B	A
2690	C	A	A	A	A	A	A	A	A	A
1499	C	C	A	C	D	C	D	A	D	C
6488	D	D	A	D	C	A	D	B	D	C
7315	C	C	A	C	D	C	D	C	D	C
4472	A	A	B	A	A	C	A	D	D	D
8415	B	C	B	A	B	D	B	D	B	A
5970	A	A	C	D	A	B	A	B	B	A

Figure 3.16: Questionnaire dataset

2. Time series dataset: In this exercise, the CSV contained vehicular sensor signals. Students had to represent the data as a single time series Feature, with both the Thing and the Device fixed, while all signal values were mapped as Items of that Feature (Figure 3.17).

timestamp	km	velocity[km/h]	accelerometer[m/s2]	temperature[C]
2023-05-15T10:10:00Z	1.6667	73.33	0.75	17
2023-05-15T10:11:00Z	2.0167	74.67	0.825	17
2023-05-15T10:12:00Z	2.4	76	0.9	16.9
2023-05-15T10:13:00Z	2.8167	77.33	0.975	16.9
2023-05-15T10:14:00Z	3.2667	78.67	1.05	16.9
2023-05-15T10:15:00Z	3.75	80	1.125	16.9
2023-05-15T10:16:00Z	4.2667	81.33	1.2	16.9
2023-05-15T10:17:00Z	4.8167	82.67	1.275	16.9
2023-05-15T10:18:00Z	5.4	84	1.35	16.9
2023-05-15T10:19:00Z	6.0167	85.33	1.425	16.9

Figure 3.17: Time series dataset

3. Trip dataset: Here the CSV described trips recorded from different vehicles. The students were required to map the TripID as the Thing, the Vehicle as the Device, and all remaining columns as Items of the relevant Feature (Figure 3.18).

TripID	Date	Vehicle	Km	Weather	Road	Passengers
48	2023-05-18T16:30:00Z	Vehicle1	42	Sunny	TestTrack	2
66	2023-05-19T07:55:00Z	Vehicle2	45	Rainy	Public	1
58	2023-05-21T12:10:00Z	Vehicle1	124	Foggy	Public	1
30	2023-05-23T18:00:00Z	Vehicle4	2	Cloudy	Private	2
50	2023-05-24T10:35:00Z	Vehicle3	145	Rainy	Public	3
38	2023-05-26T21:45:00Z	Vehicle5	152	Cloudy	Public	1
72	2023-05-27T14:20:00Z	Vehicle2	80	Sunny	Public	3
42	2023-05-29T19:05:00Z	Vehicle1	72	Sunny	Public	1
74	2023-05-30T08:50:00Z	Vehicle3	50	Rainy	Public	1
52	2023-05-31T16:15:00Z	Vehicle1	40	Cloudy	Public	1

Figure 3.18: Trip dataset

For each exercise, students were required to:

- Understand the exercise,
- analyze the CSV file,
- prepare the corresponding JSON descriptor,
- upload the dataset using Measurify,
- verify correct ingestion by downloading and checking the stored records,
- Provide feedback by answering specific questions in a Google Form.

To avoid bias due to task ordering, participants were evenly divided into three groups, each starting from a different exercise. After completing the initial task, students proceeded with the remaining exercises in sequential order. This ensured that every participant eventually carried out all three activities, while still balancing the influence of the first exercise attempted. In this way, the experimental design followed a generic and systematic method, making the results more robust and comparable.

This setup also provided a balanced mix of conceptual understanding and practical application, enabling the collection of both quantitative performance data and qualitative feedback on usability.

3.6.2 Results

3.6.2.1 Completion Time and Accuracy

As a first result, we analyzed the responses collected through the Google Form. Each participant reported the time (in minutes) required to complete each exercise. Since the order of tasks was rotated, we aggregated the results according to the position in which an exercise was attempted: as the first, second, or third exercise performed, regardless of the specific dataset used. For each position we computed the minimum, maximum, mean, and median values (Table 3.15).

Table 3.15: Statistics of minutes required to complete each exercise

Statistic	**First exercise attempted**	**Second exercise attempted**	**Third exercise attempted**
Min	3	3	2
Max	44	28	26
Mean	21.0	10.3	9.4
Median	20	10	8

The results clearly show a strong reduction in completion times after the first exercise. The mean time decreased from 21 minutes for the initial task to around 10 minutes for the second and under 10 minutes for the third. The same trend is confirmed by the median (20, then 10, to 8 minutes). This indicates that once

participants understood the workflow through the first exercise, they were able to complete subsequent tasks much faster and more consistently. It should also be noted that the reported times include not only the execution of the technical steps but also the initial effort of understanding the task and clarifying what had to be done. For this reason, very low completion times are not expected, and the observed reduction from the first to the subsequent exercises mainly reflects the learning of the workflow rather than differences in task complexity.

Outliers remain visible in the maximum values, but overall the data confirm a significant learning effect.

Following the analysis of completion times, we also evaluated the accuracy of the answers provided by participants in the Google Form. Each exercise was associated with verification questions to ensure that the JSON descriptor had been prepared correctly and that the dataset was properly uploaded. The percentage of incorrect answers is reported in Table 3.16.

Table 3.16: Error rate for each exercise

Exercise type	**Error rate (%)**
Questionnaire	13.9%
Time series	11.1%
Trip dataset	16.7%

The error rates confirm the trend observed in completion times: once the workflow was understood, students were generally able to apply it correctly. The time series exercise achieved the lowest error rate (11.1%), while the trip dataset was the most error-prone (16.7%), likely due to its slightly more complex mapping structure. Overall, error percentages remain relatively low across all exercises, suggesting that the workflow is understandable and can be applied with reasonable accuracy after minimal training.

3.6.2.2 Questionnaire Statistics and User Feedback

After analyzing completion times and accuracy, we then examined the statistics collected through the questionnaire to better understand participants' background

and perceived usability of the workflow.

Figure 3.19 summarizes the educational level of the respondents. Most participants were Bachelor students, followed by Master and PhD candidates, with a small number of researchers. This distribution ensured a heterogeneous sample in terms of prior experience, allowing us to evaluate how users with different academic backgrounds approached the proposed tasks.

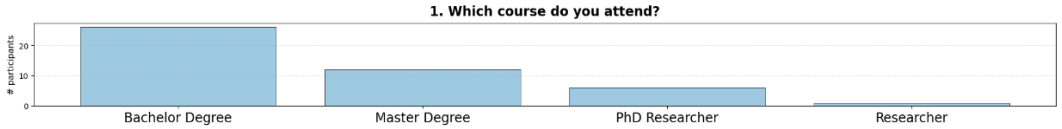


Figure 3.19: Statistics on educational level

Figure 3.20 reports the perceived level of effort required for each exercise and the general feedback on Measurify usage. As expected, the first description file was rated as the most difficult, while the second and third were considered considerably easier. This confirms that, after the initial learning phase, participants could autonomously replicate the procedure.

In the same figure, the statements related to Measurify usage received positive evaluations. Most users agreed or strongly agreed that the web interface was easy to access and that the platform could simplify data management, even for those with limited technical knowledge.

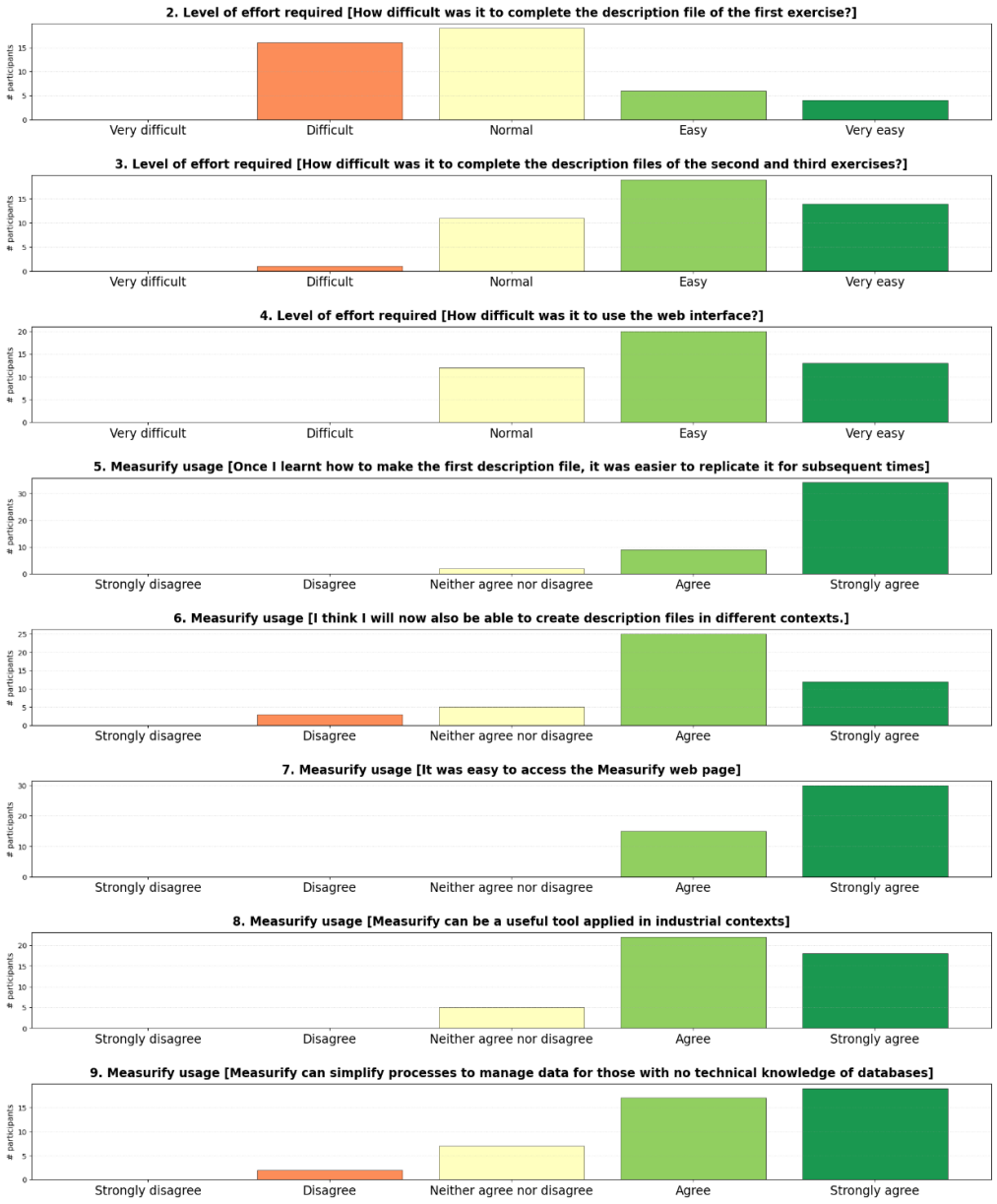


Figure 3.20: Statistics on level of effort required

Figure 3.21 collects the feedback related to the usage of Measurify and its GUI. The results are consistently positive, showing that users found the interface intuitive, easy to navigate, and effective for managing data within the database. Most participants agreed or strongly agreed that loading and downloading operations were straightforward and that the GUI provided clear and comprehensive information on the actions to perform. The overall perception of responsiveness and aesthetics was also favorable, confirming the adequacy of the

graphical components for supporting laboratory activities.

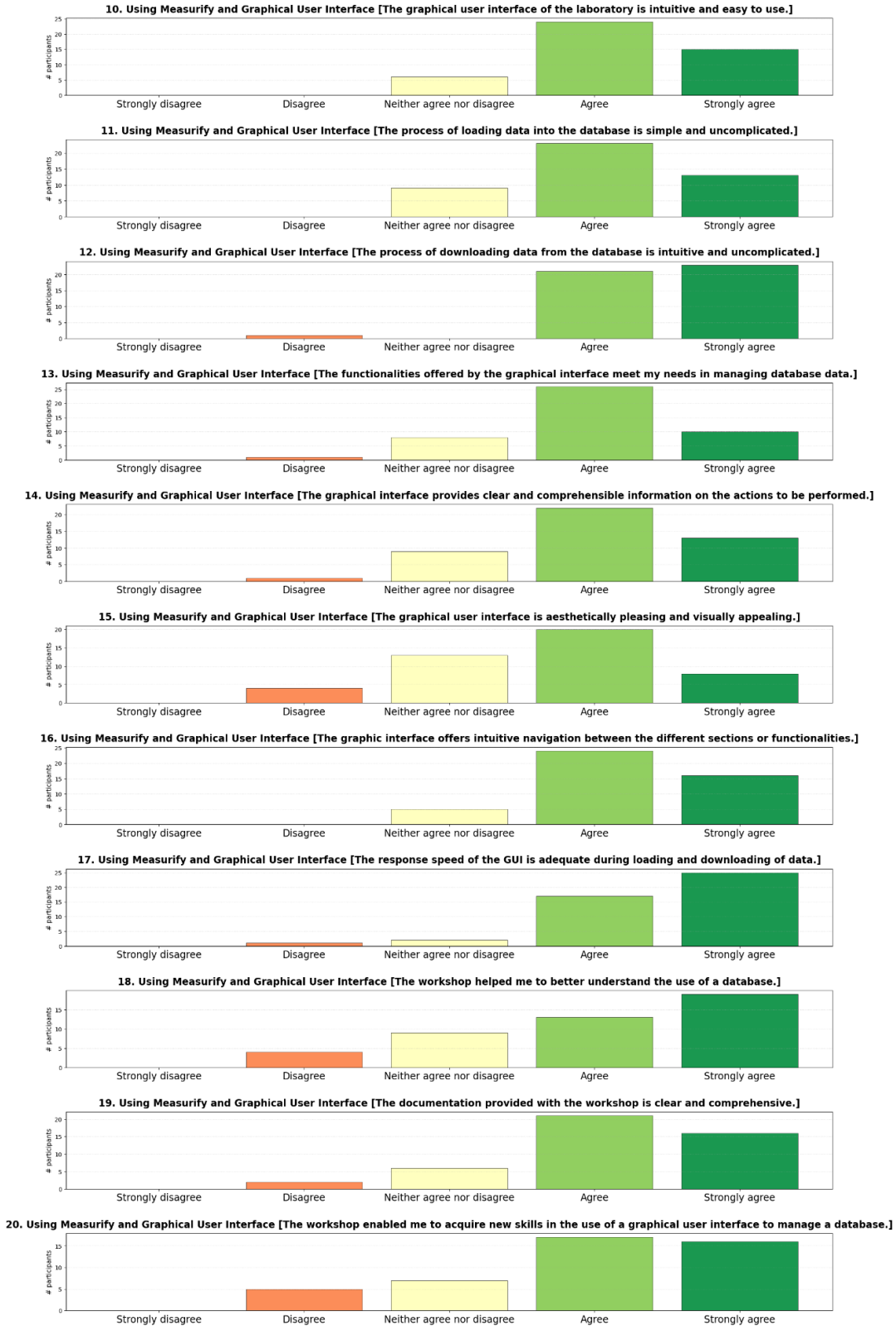


Figure 3.21: Statistics on usage of Measurify and its GUI

Figure 3.22 summarizes the general user experience evaluation. The feedback again indicates a high level of satisfaction, with nearly all participants finding the interface readable, well-organized, and responsive. Users particularly appreciated the clarity of response messages, the quick execution of operations, and the ease of navigation across pages. These results highlight the robustness of the system's usability and its ability to provide a smooth, predictable interaction flow.

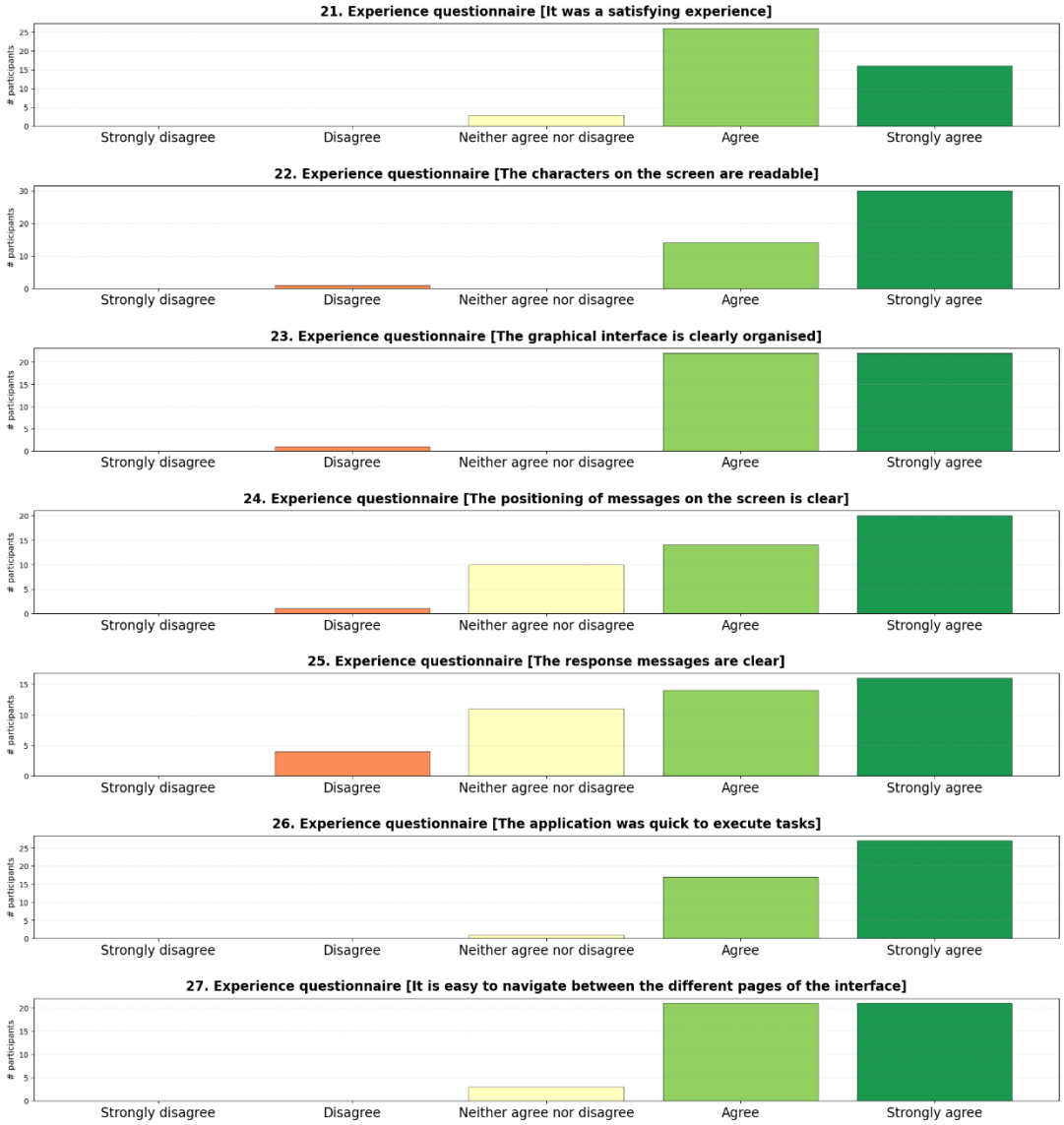


Figure 3.22: Statistics on general user experience

Figure 3.23 focuses on the participants' self-assessment of skills and familiarity with the main database-related concepts introduced during the workshop. The responses display a broad distribution, with some users reporting strong prior

knowledge and others approaching the topic for the first time. This confirms that the workflow and interface were successfully applied across a wide range of user profiles, from novices to more experienced participants.

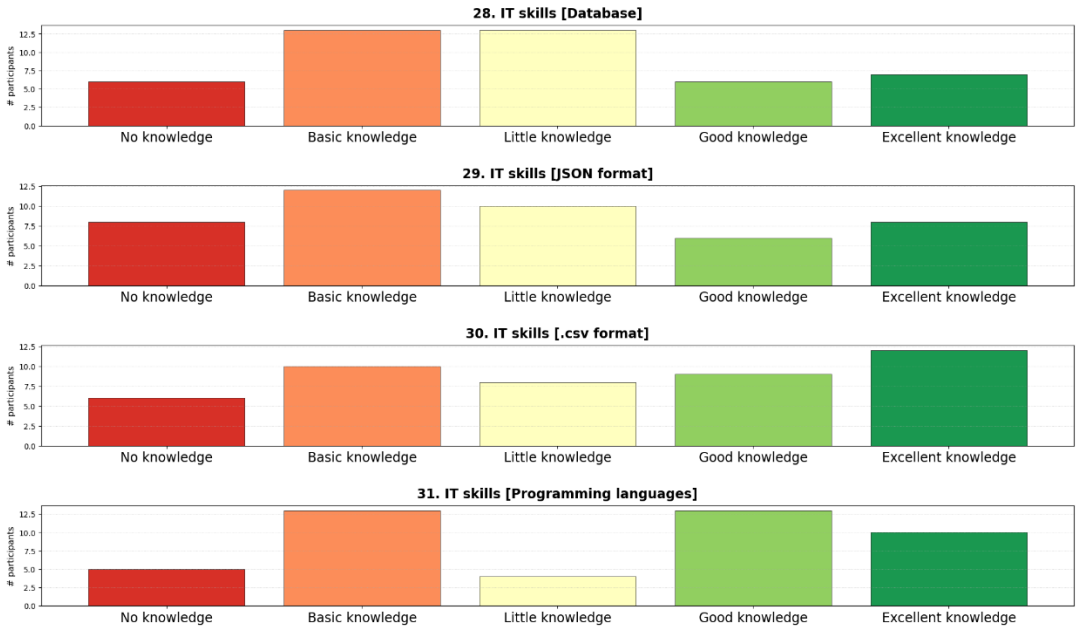


Figure 3.23: Statistics on self-assessment of skills and familiarity with concepts

Overall, the experimental results quantitatively support the usability and accessibility of the proposed workflow and its GUI. Task completion times show a clear learning effect: the average time required to complete an exercise decreased from 21 minutes for the first task to approximately 10 minutes for the second and under 10 minutes for the third. A similar trend is observed in the median values, which dropped from 20 to 8 minutes across successive exercises.

Error rates remained relatively low across all tasks, ranging between 11.1% and 16.7%, indicating that participants were generally able to apply the workflow correctly after minimal training. Questionnaire feedback further confirms these findings, with the majority of users reporting positive evaluations of the interface clarity, responsiveness, and overall ease of use.

Taken together, these results show that, once the initial concepts are understood, users with heterogeneous backgrounds can efficiently and accurately operate the proposed workflow, supporting its adoption in educational and collaborative research settings.

3.7 Model Registry

After validating the usability of Measurify for data management, the next step was to extend the framework toward model management and version control. This extension represented not only an architectural enhancement but also the foundation for all subsequent phases of the thesis, as it supports the full ML workflow: from dataset preparation to algorithm development, training, and deployment. The goal was to provide a unified infrastructure capable of handling not only datasets but also the algorithms and trained models derived from them.

The Model Registry is a component of the Hi-Drive Machine Learning Toolkit. It acts as an online repository designed for versioning and managing ML assets: algorithms, trained models, and datasets. For each resource, the registry stores metadata such as training configuration, hyper-parameter values, and performance metrics, enabling straightforward comparison and traceability across experiments. Every item is assigned a unique identifier and can include multiple versions.

The system was developed using Node.js [96] and MongoDB [97], with deployment options both locally (also via Docker) and on cloud infrastructures, including the Hi-Drive AWS space [95]. The registry supports both a GUI for immediate usability and a REST application Programming Interface (API) for programmatic access, allowing developers to integrate the functionality into external tools or custom interfaces. User authentication is managed through the dedicated account system integrated within Measurify, requiring a username and password to generate an API token for secure access (Figure 3.24).

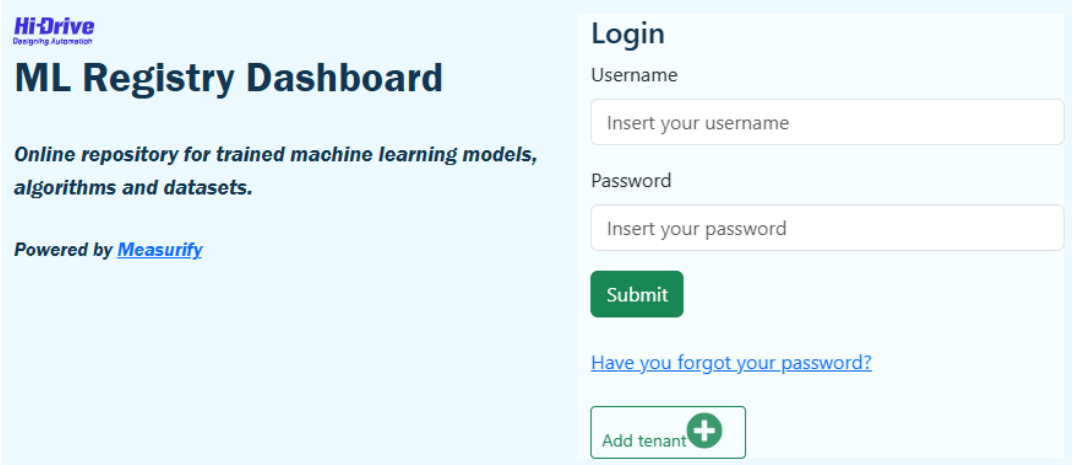


Figure 3.24: Login page

Each registered resource belongs to one of three categories:

- Algorithm: source files (typically Python) implementing the model structure and training logic.
- Model: trained weights or checkpoints produced by a training run.
- Dataset: data used to train or validate machine learning models.

Every resource supports metadata (key-value pairs) and tags for efficient organization and search.

Figure 3.25 shows an example of dataset creation through the graphical interface. In this example, the user defines the TemperatureDataset and specifies metadata such as Location, Season, and Year, as well as a visibility setting and relevant tags. The interface is designed to make dataset configuration intuitive and human-readable, reducing the need for manual editing of configuration files.

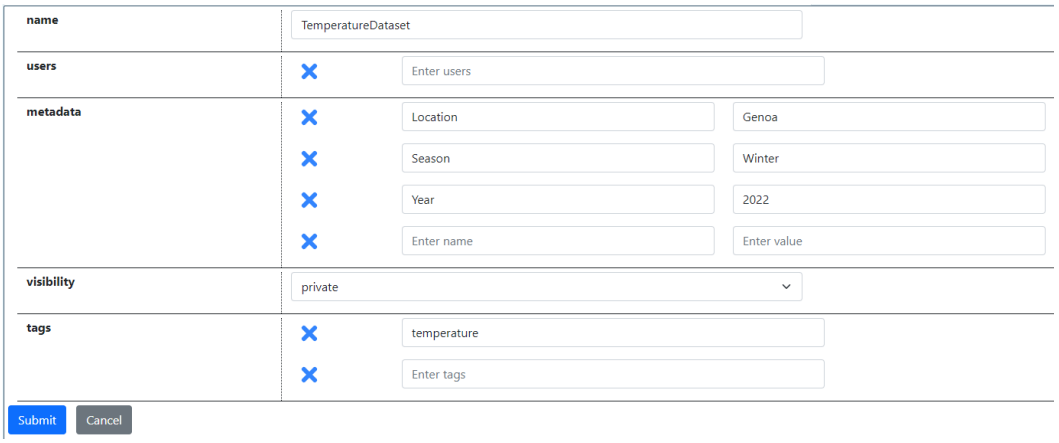


Figure 3.25: Create dataset page

Once created, the datasets appear in the main registry datasets table (Figure 3.26). Each entry lists the defined metadata and tags, while the actions column provides four main options: viewing detailed information, editing metadata, adding new dataset versions, or deleting the resource. By selecting the third icon (adding new dataset versions), users can upload multiple versions of the same dataset (Figure 3.27), for example to store different version, or both raw and preprocessed data, thus enabling fine-grained dataset versioning.

Figure 3.26: List of datasets

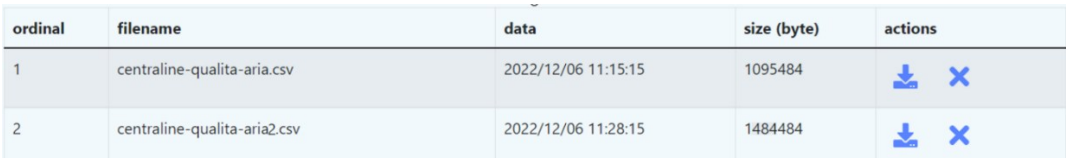


Figure 3.27: List of files uploaded for one dataset

A similar workflow applies to the management of algorithms and models (Figures 3.28 and 3.29). Algorithms can be described through metadata such as training parameters or labels, while models reference the corresponding algorithms and store attributes like architecture configuration or training status. This integrated approach ensures consistency between datasets, algorithms, and models, allowing each element of the ML workflow to be tracked and retrieved over time.

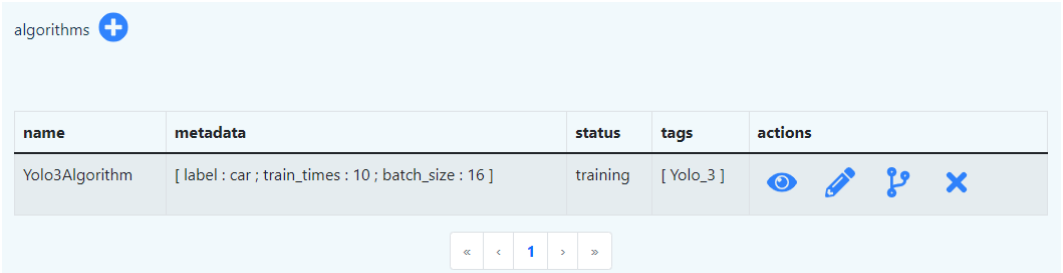


Figure 3.28: List of algorithms

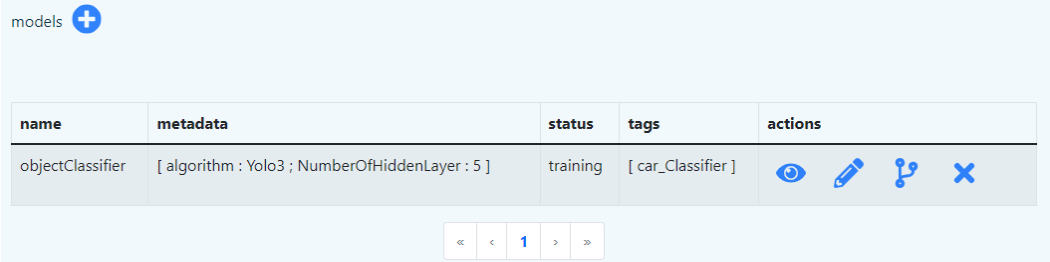


Figure 3.29: List of models

It is important to note, however, that the current version of the registry manages datasets, algorithms, and models as versioned resources, but does not yet enforce a native dependency-tracking mechanism linking each trained model to the exact version of the dataset used during training. At present, this relationship can be documented through metadata, tags, or naming conventions, but it is not automatically checked by the system.

As a consequence, if a dataset is updated after a model has been trained, the registry does not currently mark that model as out of sync with the newer dataset version. The present implementation therefore supports versioning of the individual resources, but not yet a full lineage management workflow comparable to a version control system with automatic dependency tracking. Extending the registry in this direction, for example by introducing explicit references between model versions and dataset versions, provenance metadata, and synchronization checks, represents a natural direction for future work.

Functionality tests were conducted to verify correct behavior for all asset types: creation, configuration, upload, download, and versioning. After internal validation, Hi-Drive developers were invited to use the system voluntarily in their ongoing projects. The resulting resources covered the three Hi-Drive Tech Enabler domains: Perception, Decision-making, and Driver Monitoring, including diverse algorithm families such as CNNs, YOLO, LSTMs, and Transformers, trained on datasets ranging from vehicular signals and video clips to physiological and eye-tracking data.

The overall experience within Hi-Drive demonstrates that the Model Registry is ready to manage the versioning of datasets, algorithms, and models in a unified way.

More detail on this enabler can be found in the online repository: https://github.com/measurify/model-registry/blob/master/README.md (accessed on 31 January 2026).

It is important to note that, in addition to the GUI, the same functionalities are available through REST APIs, allowing complete programmatic control. By issuing standard POST and GET requests, developers can interact with the registry directly from Python scripts or Jupyter notebooks [86] to upload, download, and query resources. This integration enables the automation of training pipelines, supports continuous experimentation, and provides a seamless bridge between human-friendly interaction through the GUI and large-scale ML workflows running in cloud or edge environments.

Looking ahead, the Model Registry will be particularly relevant in the next chapters of the thesis, which focus on ML applications built upon this infrastructure. The registry acts as a central hub connecting all the components developed later: from dataset curation and preprocessing to algorithm design, model training, evaluation, and on-device deployment.

Through systematic versioning of datasets, algorithms, and models, the registry ensures reproducibility, traceability, and consistency across experiments: three essential requirements for developing robust and verifiable AI workflows. This infrastructure also facilitates comparisons between model versions, experimental configurations, and dataset revisions, providing a reliable foundation for evidence-based analysis and incremental improvement.

CHAPTER 4

4. Sport and Human Activity Recognition

The previous chapter presented the extensions to the Measurify framework that enable scalable time-series data management, usability-oriented workflows, and systematic version control of ML assets through the Model Registry. Building upon this infrastructure, this chapter shifts the focus from data management mechanisms to the design and validation of an end-to-end Edge-to-Cloud workflow that connects data acquisition, storage, analysis, and deployment of ML models.

The proposed workflow aims to provide a coherent architecture for managing the full lifecycle of time-series data, from signal collection on embedded devices to intelligent inference at the edge.

To validate this approach in a realistic and well-established context, the workflow is applied to the domain of sport and human activity recognition, which represents a representative use case involving wearable sensors, continuous data streams, and lightweight on-device inference.

The proposed architecture is organized into three main layers:

- a wearable edge device, responsible for acquiring and preprocessing motion signals;
- a fog device, managing short-range communication, buffering, and configuration;
- the cloud, which provides scalable data storage, labeling, visualization, and remote access through Measurify's REST APIs.

At the cloud level, Measurify provides scalable data storage, metadata management, labeling, and visualization services through its RESTful APIs. The Model Registry complements this infrastructure by managing datasets, algorithms,

and trained models in a consistent and reproducible manner, enabling transparent tracking of ML experiments and deployments.

The following sections describe the system design and workflow in detail, including the selection of sensors and microcontrollers, communication technologies, and power management strategies. They also present the fog-level application for session orchestration and data synchronization, the cloud-based data handling pipeline, and the deployment of embedded neural models for real-time activity recognition.

Finally, the proposed workflow is evaluated across multiple sports scenarios to assess classification accuracy, inference latency, and energy consumption. These experiments demonstrate the general applicability of the Edge-to-Cloud approach to human activity recognition and provide insights into the trade-offs involved in deploying ML models on resource-constrained wearable devices.

4.1 Edge-to-Cloud Workflow Architecture

4.1.1 System Architecture

Unlike traditional human activity recognition [98], [99], [100] settings that focus on a fixed set of activities and sensors, multi-sport scenarios introduce additional challenges related to variability in movement patterns, sensor placement, sampling requirements, and activity-specific labels. A single data collection system must therefore adapt to heterogeneous sports disciplines while preserving consistency in data management and annotation. These requirements motivate the design of a flexible Edge-to-Cloud architecture capable of supporting multi-sport data acquisition within a unified workflow. In order to address the challenge of building a framework for efficient collection of multi-sport activity data, we design a system architecture consisting of three main layers: a wearable edge

Parts of this chapter has been published in Fresta, M.; Bellotti, F.; Capello, A.; Dabbous, A.; Lazzaroni, L.; Ansovini, F.; Berta, R. End-to-End Dataset Collection System for Sport Activities. Electronics 2024, 13, 1286. https://doi.org/10.3390/electronics13071286.
The material has been adapted and extended to fit the structure and scope of this thesis.

device (ED), a personal fog device (FD), and the cloud. The ED is a microcontroller that collects raw sensors signals, while the FD serves as the primary storage interface. It is typically connected to the ED using a short-range wireless connection (e.g., Bluetooth, Wi-Fi) to avoid latency issues and any line or Internet connection failures. The FD can also be used to configure the system (e.g., in terms of sampling rate and sport activities' recognizable labels). During data collection, the ED continuously sends raw data to the FD, which also processes it for subsequent transmission to the cloud. This data delivery step, which typically is performed after the sports session, requires an Internet connection and the use of HTTPS Representational State Transfer (REST)-ful APIs. Finally, the cloud database stores and organizes the data for easy retrieval and analysis. The proposed workflow is summarized in Figure 4.1.

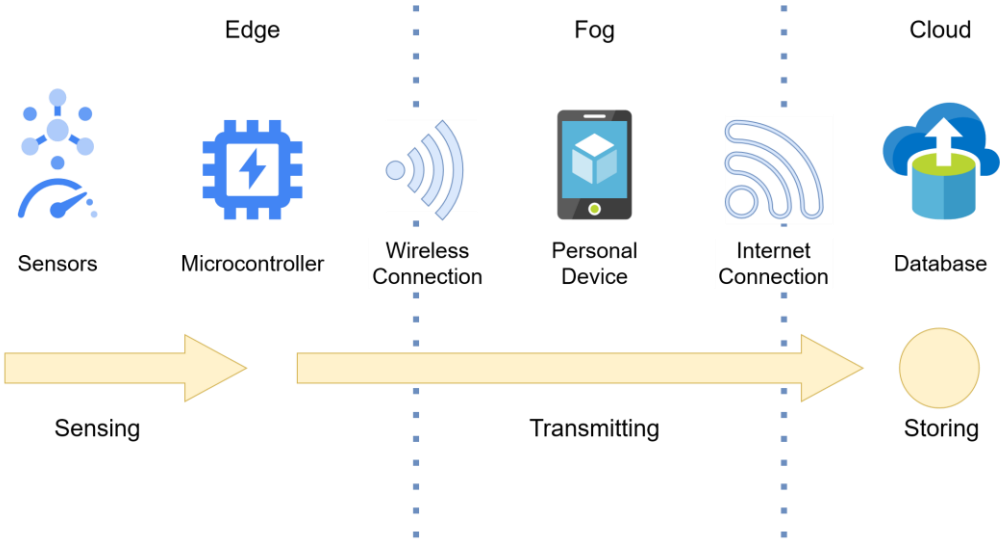


Figure 4.1: Data collection system workflow.

In the following subsections, we explain and discuss the design choices and the challenges we faced to meet the requirements for the ED in terms of performance, power, affordability, versatility, and compactness of size. Such requirements, collected by informally interviewing relevant sports experts, are synthesized as follows:

Operative range: >150 m, based on the standard soccer field's diagonal length;

Battery duration: >4 h, longer than most match/session durations;

Size: <50 mm × 50 mm × 30 mm, to allow the device to be placed without hindering the sport activity;
Wearable device: the device should be easily attachable to body/equipment parts such as the wrist, waist, leg, handle, etc.;
Ease of use: users who are deeply engaged in a sport activity cannot be distracted nor are they supposed to have particular knowledge about devices and systems.

4.1.1.1 Edge

The edge layer incorporates the device that records and preprocesses physical signals before transmitting them to the next fog layer. In the system design, we evaluated different options for communication, sensor, microcontroller, and power supply, considering the target application discussed in the following.
For the choice of the most suitable communication technology for our architecture, we identified three options: GSM, Wi-Fi, and BLE. Each one presents unique advantages and trade-offs, necessitating a consideration of factors such as, particularly, power efficiency and operational range. Table 4.1 provides an outlook, considering our aims. We assessed Sim900 as a GSM module [101], Nina W102 as a Wi-Fi module [102], and Nina B306 as Bluetooth module [103], as they are commonly used modules in IoT applications due to their affordability and reliability.

Table 4.1: Comparison between GSM, Wi-Fi, and BLE.

Module	**GSM**	**Wi-Fi**	**BLE**
Peak current	~600 mA	~190 mA	~14 mA
Distance	NA 1	<50 m	<400 m
Byte per message	NA 2	NA 2	244
Additional fog requirements	NA 1	Wi-Fi Internet connection	Device with Bluetooth and Internet connection
Additional shield	Yes	Often onboard	Often onboard
Additional costs	SIM contract	No	No

[1] GSM does not require near-range devices. 2 HTTP protocol has no limitation on body size.

The comparison highlights the strengths and weaknesses of each communication type, leading to different possible applications for each one:

- GSM: This option is more expensive and demands a higher power supply. It is best suited for environmental data collection applications with a powerful edge carrying out the activities of the fog layer as well, thus also transmitting data to the cloud. It offers wide coverage, making it ideal for utilization in remote areas. For these reasons, GSM in wearables is typically used for safety and health systems [104], [105], which require prompt action in cases of serious need, thus bypassing the fog layer;
- Wi-Fi: This communication system is preferred for applications where the size of the board is relevant and a power source is available. It utilizes a fog layer as a bridge between the device and the database, without the possibility of controlling data recording. Wi-Fi is suitable for indoor environments or areas where connectivity is stable and reliable [106].
- BLE: This solution fits applications that require both a compact device size and low power consumption. It needs a fog layer to relay data to the cloud server. This layer can also serve as a human–computer interface with the ED, enabling dynamic functionality changes remotely. BLE is ideal for wearable devices or scenarios where energy efficiency and user inputs are priorities [107].

In the process of collecting sports activity data, the requirements for power efficiency, affordability, and compactness make BLE the optimal connectivity option [108]. This solution ensures an operating range that can entirely cover a sports field while maintaining low power consumption, which is essential for continuous operation throughout a whole match. Moreover, BLE is particularly suited for applications requiring the ongoing transmission of numerous small data packets, so this choice optimizes both performance and reliability.

The type of deployable sensors strictly depends on the objective of data collection. For our SAR task, we consider inertial sensors, which have proven to be reliable and efficient for sports activities [109], [110]. Given the size constraint of the wearable device, it is necessary to have sensors and a connectivity module already

present on board. We discarded the option of creating a custom board, which could have optimized efficiency and compactness but would have raised issues in terms of cost and open-source hardware architecture replicability. Thus, we chose to use a commercially available board and adapt it to our use case, with some modifications in terms of power supply, as discussed in the next sub-section.

Particularly, we opted for an Arduino Nano family board [111], given its large developer community, online support, comprehensive documentation, and compact size. Considering the choices related to sensors and the BLE module, we could have selected either the Arduino Nano 33 BLE or the Arduino Nano 33 BLE Sense. The only difference between these two boards is the wide variety of sensors (i.e., light, humidity, temperature, pressure, etc.) on the latter. Therefore, we built our prototype using the Arduino Nano 33 BLE Sense, but the developed architecture supports both models.

The firmware running on the board has been designed to collect and transmit data from the sensors to the fog layer via its built-in BLE connectivity module. This system provides 9-axis IMU readings (i.e., accelerometer, gyroscope, and magnetometer) over BLE. Although the BLE hardware theoretically supports useful payloads up to 244 bytes, the communication setup adopted in this work, based on the standard ArduinoBLE library configuration on the Arduino Nano 33 BLE Sense, effectively handled 20 bytes per BLE transmission. Therefore, to transmit the array of IMU values, each reading was encoded as a 16-bit signed integer (int16_t). This choice preserved the full resolution of the 16-bit sensors while keeping the communication lightweight and compatible with the adopted software configuration. It also limited the amount of transmitted data, thereby helping to contain communication overhead and energy consumption. Larger payloads were not used in the present implementation.

The type of power supply on the board is strictly dependent on the application:

- Battery-less: it requires additional hardware to harvest energy from the environment [112] (i.e., antennas, solar panels, etc.). This solution would add significant complexity to the design and is not the focus of our research but could be considered at a later stage.

- External battery: this option exploits the USB connection of the Arduino board, allowing the user to connect an external battery. It adds weight to the device and could increase its size.
- Embedded battery: this option requires the development of a dedicated board to connect a small battery to the device. While this solution does result in a slight increase in the device's size and weight, it retains its overall compact design.

Considering the SAR task, we opted for an embedded battery. Given the lack on the market of battery shields tailored to the Arduino Nano's compactness, we designed and developed a dedicated board to equip the Arduino Nano 33 BLE Sense with a battery, enabling complete standalone operation.

The battery shield is designed to deliver power to the Arduino board through a rechargeable battery and enables the battery's recharging when the Arduino board is connected to an external power source. The specific battery employed is a commonly available button-cell Lithium-Ion (LIR2450) with a nominal capacity of 120 mAh.

The battery shield circuit consists of the battery charger; a step-up converter to raise the battery voltage from 3.6 V to 5 V, which is the level indicated by Arduino specifications, and the connection to the Arduino board. The step-up circuit is represented in Figure 4.2. S1 (top of the picture) is the switch to turn on/off the ED connection to the battery. IC1 (TLV6112) is the operational amplifier implementing the step-up. IC2 (74LVC1GD4Z) is an inverter toggle to disable IC1 and enable the battery charger IC3 (MCP73831) (Figure 4.3). With this last configuration, the battery is recharging and the Arduino is powered from its built-in USB port. Two LEDs notify the user about the current status: D3 indicates the recharging of the battery, and D1 shows that 5 V is supplied by the step-up converter.

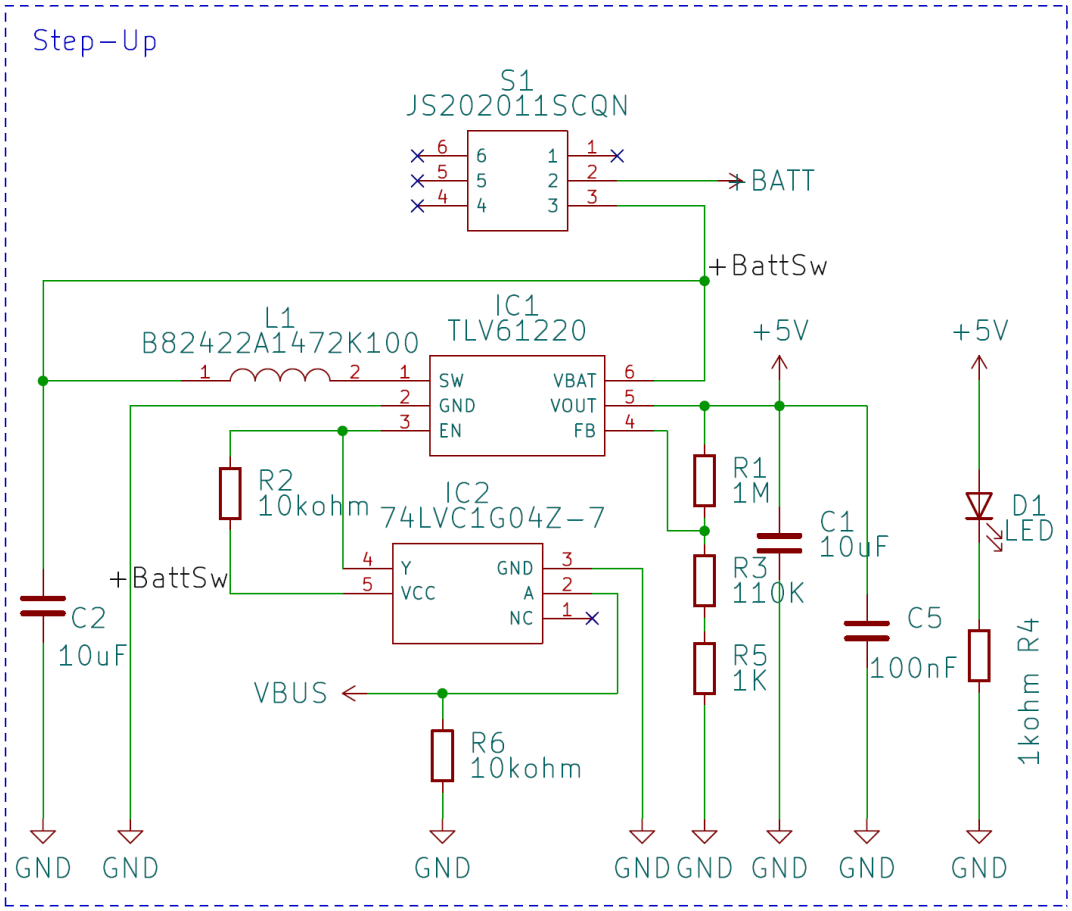


Figure 4.2: Step-up circuit of the battery shield.

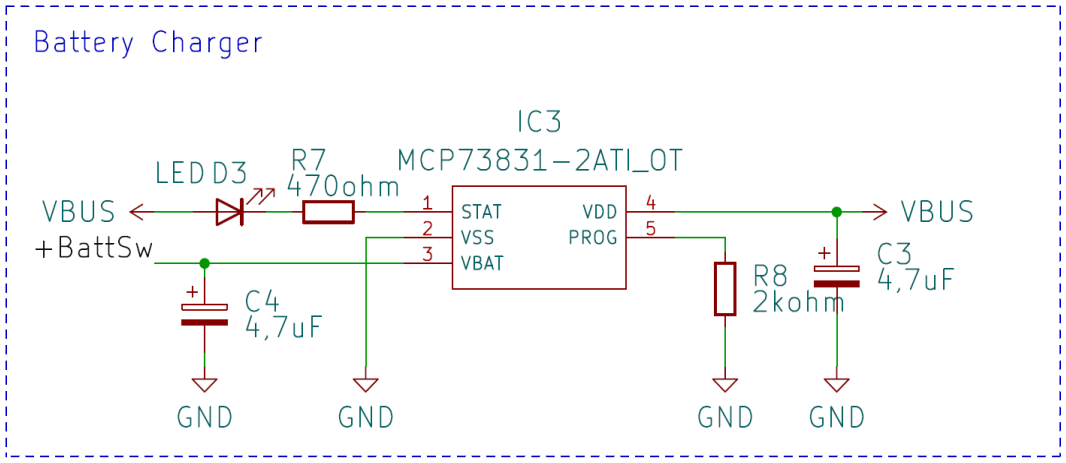


Figure 4.3: Battery charger circuit of the battery shield.

Figure 4.4 finally shows the connections between the Arduino pins and the battery shield. Pin 12 (VUSB) forwards the power from the Arduino's USB port to IC3 to re-charge the battery. When the USB is not connected, on the other hand, pin 15

(VIN) is connected to the output of the IC1 to supply the microcontroller with 5 V. The D2 Zener diode has been placed to block any possible current flow back from the board (VIN) to IC1 during a re-charge. The picture clearly shows that several Arduino pins are left unused and freely employable to connect sensors or other outboard devices (e.g., Real Time Clock), depending on the specific application needs.

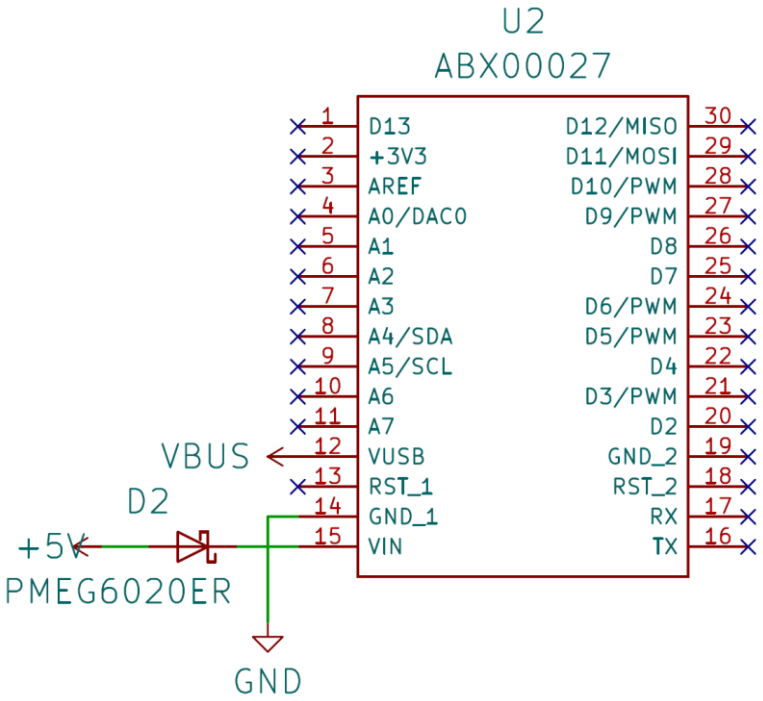
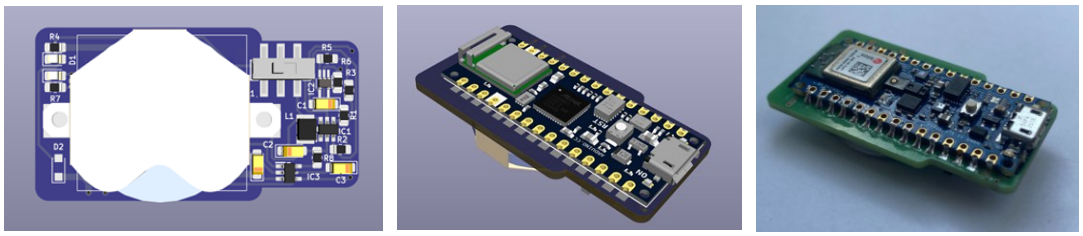


Figure 4.4: Arduino power connections.

Our final ED prototype is sized 48 mm × 26 mm × 12 mm, maintaining the compact size of the original Arduino board, which is 45 mm × 18 mm × 3 mm. A render of the ED is shown in Figure 4.5.



(a) (b) (c)

Figure 4.5: Render of the ED. (a) Bottom view of the ED, with indication of the components on the battery shield; (b) Top view of the ED, showing the Arduino board placed on the battery shield.; (c) Final ED used.

4.1.1.2 Fog

The fog layer serves as the intermediary system tier, typically utilizing personal devices like smartphones and tablets to receive BLE communications and forward data to the cloud through HTTPS REST APIs. To support data collection operations, we designed and developed a smartphone app we called “Smart Collector”. We avoided limitations related to the operating system in use (i.e., iOS, Android, Windows) by deploying the app with Flutter [113], a well-established cross-platform development framework. Flutter allows users to develop responsive and reliable applications with essential device-level functionalities, such as Bluetooth and Internet connectivity, through dedicated packages, namely quick_blue and connectivity_plus.

The Smart collector allows users to configure the ED by setting the sampling rate. Moreover, it allows for the labels of the specific sports actions to be recognized. Finally, it is used to start the recording process. Upon stopping the recording, the app transmits the data (i.e., signal timeseries and their corresponding labels) to a cloud database. Data transmission to the cloud occurs over an HTTPS connection using a RESTful API POST method. Typically, data are sent in either JSON or CSV format [114]. To reduce the amount of data transferred over the internet, the Smart Collector exploits a compact CSV format [115]. CSV is particularly effective when the data size remains consistent across samples, which is the typical case of datasets.

Figure 4.6 shows the application page in which users can select the sport activity and the type of sport action that is going to be recorded (i.e., the class label of the sample in the dataset) and optionally specify the name of the time series.

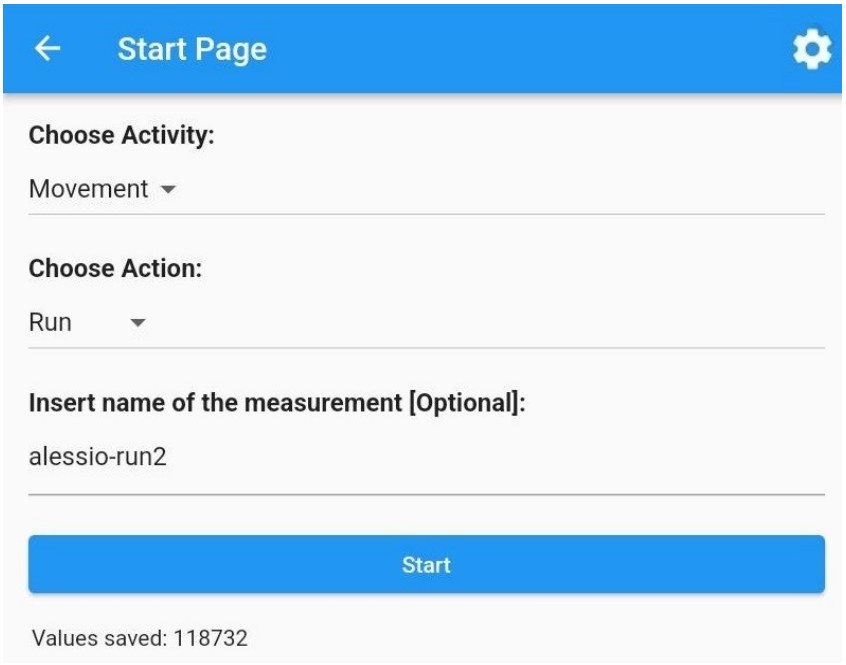


Figure 4.6: Snapshot of the Flutter application.

During the recording, the user can check the number of collected samples in the bottom-left corner of the app (Figure 4.6). The Smart Collector will try to send data to the database as soon as the recording is stopped; in the case of a missing or unstable internet connection, the data are stored locally, and automatic upload attempts are executed once a stable internet connection is re-established.

While dataset creation is a peculiar functionality of the overall system, the FD can also work in pure data collection modality, thus sending the collected samples to a ML inference service that can run in the cloud but also directly on the FD.

4.1.1.3 Cloud

The cloud layer is the framework to store data received from the fog layer. For generic data collection, the framework should support time series, be efficient, and be capable of ingesting a large amount of data from the FD. A user management system is also required, to guarantee data security and user privacy.

The cloud layer is implemented using the enhanced version of Measurify [116], as described in Chapter 3.

For user authentication, Measurify provides a JSON Web Token (JWT) after successful authentication; EDs are authenticated by a distinct token provided upon

device definition. This token offers limited access to database resources but has an unlimited lifetime, which eliminates the need for repeated login requests to renew the token at the beginning of each data collection session. Furthermore, Measurify aggregates data sent from various sources and provides a time series visualization dashboard (Figure 4.7).

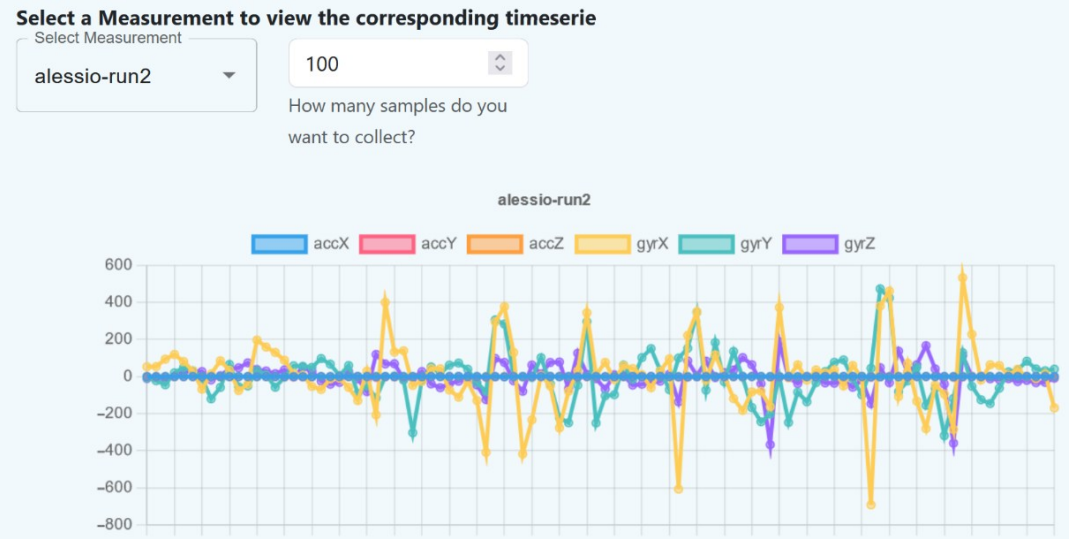


Figure 4.7: Snapshot of the time series visualization on Measurify.

4.1.1.4 Resulting Architecture

Figure 4.8 synthesizes our system architecture as described in the previous subsections.

The edge layer is composed of an Arduino Nano 33 BLE Sense connected to a battery shield. It collects data from the LSM9DS1 IMU sensor present onboard and transmits packets of 9 int16_t to the fog layer via BLE.

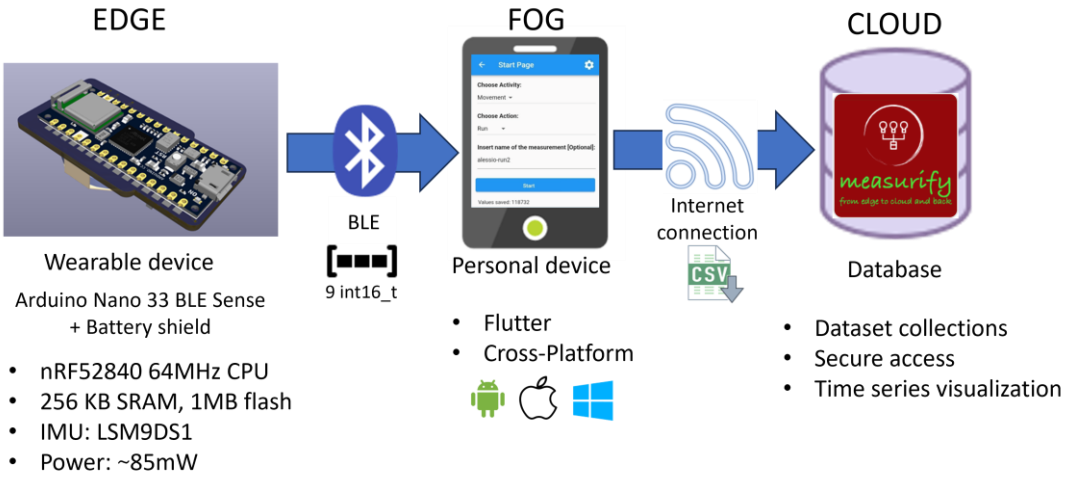


Figure 4.8: Developed architecture.

The fog layer serves as a user interface to configure and manage the recording process. Received data are then formatted as a CSV file and sent to Measurify through dedicated RESTful APIs. If Measurify is configured once for each new sport activity dataset that will be collected, then no additional effort is requested by the user. To map the collected data to the Measurify resource model, the Thing was defined as "UserN", where N identifies the subject associated with the recording session, the Device was set to "edge-meter", and the Feature was defined as "IMU", corresponding to the 9 sensor values acquired at each time step, namely the x, y, and z components of the accelerometer, gyroscope, and magnetometer.

4.1.2 Results

4.1.2.1 Applications

We assessed the proposed system by setting up four end-to-end dataset collection use-cases. We selected various sports (namely: fitwalking, tennis, soccer, and boxing), each one with a different set of recognizable actions, as reported in Table 4.2. It is important to highlight that the goal of this assessment is to verify the overall workflow of a versatile architecture, not the implementation of a single ML application that overcomes the state of the art. For this reason, the size of the collected datasets may be smaller than that of similar ML datasets in the literature.

Table 4.2: Tested applications.

Sport	**Actions**	**ED Position**	**Number of Athletes**	**Recording Duration**	**Sampling Frequency**
Fitwalking	Walking, Running, Climbing Stairs	Chest	10	150 min	10 Hz
Tennis	Forehand, Backhand, Serve	Racket	10	190 min	10 Hz
Soccer	Shot, Pass, Stop	Shin guard	10	180 min	10 Hz
Boxing	Cross, Left Hook, Right Hook	Belt	10	130 min	10 Hz

Data were collected directly from the field while the athletes were training. EDs were placed on the player's body or installed in a playing tool without interfering with user's actions (Figure 4.9 - ED prototype embedding for (a) fitwalking, (b) tennis, (c) soccer, and (d) boxing). Sensor placement was selected depending on the target activity, with the aim of positioning the device where the motion patterns associated with the considered actions were expected to be most informative. For this reason, different placements were adopted across applications, including the racket in tennis, the shin guard in soccer, the belt in boxing, and the chest in fitwalking and skiing (Figure 4.9). These results suggest that the proposed workflow is not tied to a single sensor location, and that the machine learning models can extract useful discriminative patterns from different placements, provided that the selected position is meaningful for the motion to be recognized. At the same time, once a placement is chosen for a given task, it should be kept consistent throughout data collection and validation, since uncontrolled variations in positioning may affect signal quality and introduce unnecessary variability. The working frequency depends on the specific activity carried out. In all our test applications, 10 Hz was sufficient,. Additionally, we verified that the system is able to correctly operate at a sampling frequency of 30 Hz.

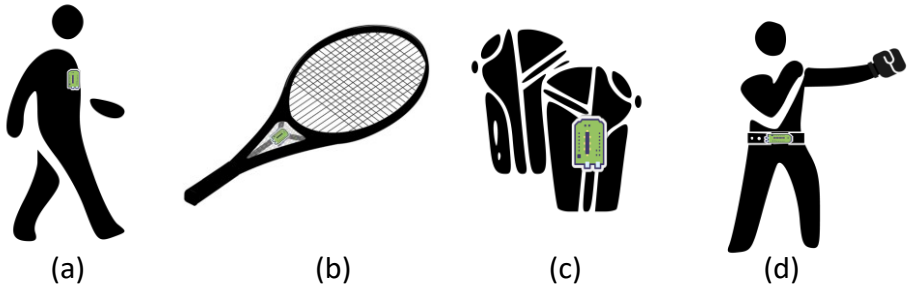


Figure 4.9: ED prototype embedding: (a) fitwalking; (b) tennis; (c) soccer; (d) boxing.

In our typical application workflow, for data collection convenience, the athlete is asked to perform a series of actions of the same class (e.g., the forehand shot in tennis), whose label is selected on the Smart Collector. All the signal samples collected are stored in a single time series, so the result of the data collection is a number of labeled time series in each of which a single action is repeated several times.

4.1.2.2 Supervised ML Dataset Assessment

To assess the validity of the proposed end-to-end system, including the ED's correct positioning in the different SAR use cases and capability to correctly capture the physical signals' features, we tested the four datasets stored in the cloud with simple DL neural models.

For each dataset, its time series are split and trimmed into time windows of different sizes, each one exactly representing a single instance of the labeled action. Details on the Dynamic Time Warping (DTW) techniques we employed for aligning the different-length samples are provided in [117]. Each dataset was finally divided into a 70% training set, a 10% validation set, and a 20% test set.

For classifying the time windows, we used a class of DL models that are commonly used to classify time series [118], such as 1D CNNs. CNN classifiers typically include several convolutional layers aimed at extracting the feature, followed by a few dense layers for the final classification (Figure 4.10). Since deep models are typically more effective, even if they take longer to train [119], we decided to use models with a simple structure (stacking only convolutional and dense layers), few layers, and small kernel sizes (Table 4.3), so as to better reflect

the quality of the dataset. Results in Table 4.3 show the high action recognition accuracy levels achieved in all the use cases.

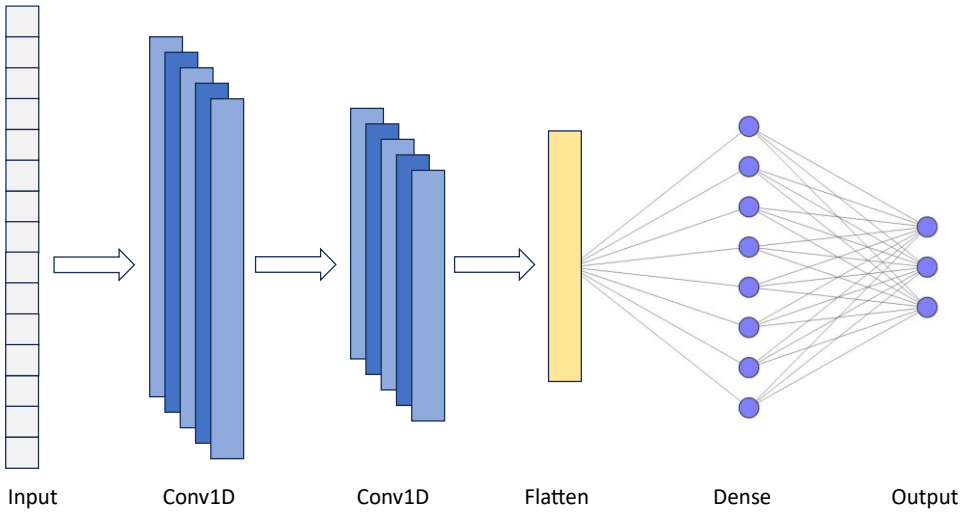


Figure 4.10: Example of CNN architecture.

Table 4.3: Details on training and testing of the CNN with collected datasets.

Sport	**Input Size** [1]	**Kernel Size**	**NN Layers**	**Test Accuracy**
Fitwalking	[25, 6]	3	[Conv1D, Conv1D, Conv1D, Dense, Dense]	93.81%
Tennis	[15, 6]	3	[Conv1D, Conv1D, Dense, Dense]	97.54%
Soccer	[15, 6]	3	[Conv1D, Conv1D, Dense, Dense]	96.35%
Boxing	[15, 6]	3	[Conv1D, Conv1D, Dense, Dense]	95.97%

[1] The first axis is the length of the time window, the second one is the signal dimensionality (i.e., 3-axis accelerometer, 3-axis gyroscope).

4.1.2.3 Power Consumption Analysis

To assess the power consumption in relation to the firmware running on the ED, we tested the ED under four different working conditions. In each condition, the battery is first fully charged (45 min) then fully depleted. The first condition, indicated as Idle, should be interpreted as a baseline board-level condition rather than as a low-power sleep state of the MCU. In this test, the data collection firmware was replaced with an empty sketch, and the measured consumption therefore refers to the complete edge device under the adopted hardware

configuration, including board-level and power-management overheads, rather than to the microcontroller alone operating in a dedicated sleep mode. For this reason, the reported Idle power should be considered as a practical baseline for the proposed prototype, and not as a reference value for the intrinsic low-power capability of the MCU. In the second one, named Standalone, our firmware collects data without transmitting it to the FD. The third and fourth conditions involve continuous data collection and transmission to the FD, but at different rates: 10 Hz and 30 Hz, respectively. Reported values are averages of over five systems running under the same battery and working conditions. The power consumption from the battery is calculated as Power = Voltage × Current, measuring the current drawn by the board through a digital multimeter (Agilent 34401A, SGLabs, Perugia (PG), Italy), with the board powered by a 3.6 V DC power supply (Agilent E3631A, SGLabs, Perugia (PG), Italy) simulating a fully charged battery. Conversely, the current and power consumption from USB are measured using a USB-Multimeter (JT-UM120 [120]) connected to a Laptop PC. Results in Table 4.4 highlight the energy efficiency of the proposed system, allowing a battery duration of more than four hours, which makes it usable for several sport types. Standalone operation is the main consumption factor, while transmission impact is relatively lower, increasing with the sampling frequency. Table 4.4 also reports, for each ED condition, the average current and power consumptions in both the ED power modes: 3.6 V coin battery (columns 3 and 4, respectively) and 5 V USB cable (columns 5 and 6, respectively). When powered by a battery, the ED drains more current (hence power) than when it is powered by USB. This is caused by the additional energy absorbed by the battery control circuit and by the operative led D1 (Figure 4.2), which signals the functioning of the battery and could be removed to reduce energy consumption.

Table 4.4: Power consumption analysis results with Arduino Nano 33 BLE SENSE.

ED Condition	**Battery Duration [min]**	**Battery-Powered (3.6V)**		**USB-Powered (5V)**	
		Average Current [mA]	**Average Power Consumption [mW]**	**Average Current [mA]**	**Average Power Consumption [mW]**
Idle (Empty sketch)	355	44.41	159.88	21.33	106.67
Standalone	301	52.22	187.99	25.02	125.10
10 Hz Continuous	293	52.65	189.54	25.14	125.69
30 Hz Continuous	286	53.02	190.87	25.19	125.97

To better interpret the difference between the USB-powered and battery-powered measurements reported in Table 4.4, an additional investigation was carried out. First, the current measurement methodology was unified across the two supply conditions by using the same industrial instrumentation and a direct supply approach in both cases. In particular, the board current was measured with the same digital multimeter (Agilent 34401A, SGLabs, Perugia (PG), Italy), and power supply (Agilent E3631A, SGLabs, Perugia (PG), Italy), using 5 V for the USB-powered case and 3.6 V for the battery-powered case, so as to reduce possible inconsistencies due to different measurement procedures.

A second set of improvements was then introduced at both firmware and hardware level. The LED D1 on the shield was removed, the power LED was disabled, and the firmware was further optimized to manually switch off the sensors that were not required during the considered operating conditions. In addition, the original Arduino Nano 33 BLE Sense board used in the first prototype, that it is out of production, was replaced with the Arduino Nano 33 BLE Sense Rev2, which is the current official revision of the platform.

The additional measurements in Table 4.5 showed a substantial reduction in the overall board-level power consumption. To better characterize the power behavior of the edge device, the previous idle condition was replaced with a minimal active firmware based on a NOP sketch. In addition, the measurements were repeated

both with the power LED enabled and disabled, so as to explicitly quantify its contribution to the total consumption.

The comparison between the two NOP configurations shows that the power LED has a non-negligible impact on the board-level power budget. In the battery-powered case, disabling the power LED reduces the average power consumption from 49.00 mW to 41.54 mW, corresponding to a reduction of about 15.2%. In the USB-powered case, the same change reduces the consumption from 42.25 mW to 36.10 mW, corresponding to a reduction of about 14.6%. This confirms that, although the LED is not the dominant source of consumption, disabling it is a simple and effective optimization for low-power operation.

After identifying this effect, the power LED was kept disabled for the remaining measurements, in order to better isolate the impact of the actual operating conditions. Under this optimized configuration, the transition from the baseline NOP sketch to the Standalone condition increased the power consumption from 41.54 mW to 59.80 mW in the battery-powered case and from 36.10 mW to 50.70 mW in the USB-powered case, reflecting the additional cost of sensor acquisition and firmware activity. Enabling continuous communication produced only a limited further increase, with power rising from 59.80 mW to 60.55 mW at 10 Hz and to 60.88 mW at 30 Hz in the battery-powered case, and from 50.70 mW to 51.30 mW and 51.60 mW, respectively, in the USB-powered case. These results indicate that, once the sensing and processing pipeline is active, the incremental contribution of BLE transmission is relatively small in the tested configuration.

The ratio between the USB-powered and battery-powered power values also provides an estimate of the overall efficiency of the battery-powered conversion path. The corresponding efficiencies are approximately 86.9% for the NOP sketch with power LED off, 86.2% with power LED on, 84.8% in the Standalone condition, 84.7% at 10 Hz continuous operation, and 84.8% at 30 Hz continuous operation. Therefore, after the hardware and firmware optimizations introduced in this revised prototype, the battery-powered configuration operates with a stable effective efficiency of about 85-87%, which is consistent with a reasonably efficient power-conversion stage for the considered operating regime.

These values are also compatible with the expected behavior of the adopted power-conversion stage. In particular, according to the datasheet of the TLV61220 converter (in Figure 4.2), for an input voltage of 3.6 V and an output voltage of 5 V, the efficiency in the considered operating region is close to 90% (wrt. Fig. 6 in the datasheet https://www.ti.com/lit/ds/symlink/tlv61220.pdf). Therefore, the experimentally observed effective efficiency of about 85-87% is reasonably consistent with the expected converter performance once board-level overheads are taken into account, and can be considered acceptable for the target application scenario.

To better understand the markedly higher consumption observed in the first board revision compared to the Rev2 version, an additional qualitative thermal inspection was carried out on multiple Nano 33 BLE Sense v1 boards. This inspection showed that one board-level DC/DC converter component, identified as MPPS3610052M, reached temperatures of about 40 °C, whereas the average temperature measured on the rest of the board was around 20 °C. Since this component is not present in the Rev2 board, where it appears to have been replaced by a different solution, this observation suggests that the excess dissipation in the original prototype was likely associated with the power-conversion stage of the first revision. Although this does not constitute a complete electrical characterization, it is consistent with the significantly lower consumption measured on the updated board.

Table 4.5: Power consumption analysis results with Arduino Nano 33 BLE SENSE rev2.

ED Condition (without LED D1)	**Battery-Powered (3.6V)**		**USB-Powered (5V)**	
	Average Current [mA]	**Average Power Consumption [mW]**	**Average Current [mA]**	**Average Power Consumption [mW]**
NOP sketch, power LED off	11.54	41.54	7.22	36.10
NOP sketch, power LED on	13.61	49.00	8.45	42.25
Standalone	16.61	59.80	10.14	50.70
10 Hz Continuous	16.82	60.55	10.26	51.30
30 Hz Continuous	16.91	60.88	10.32	51.60

4.1.2.4 Comparison with State-of-the-Art Solutions

As a final summary, Table 4.6 presents a comparison between our model and a selection of state-of-the-art solutions. The outlook highlights that while all the analyzed solutions are, at least in principle, applicable to different types of sports, our system has been defined as multi-sport by design. This is particularly due to the high configurability of all the components, such as ED/FD (where the sport action classes can be selected from a list or custom-defined) and cloud (where Measurify's resources, such as a dataset, can be configured in a matter of a few minutes). This makes the overall system usable by anyone with basic computing skills, overcoming the technical difficulties involved in creating the system from scratch. This allowed our BSc Electronic Engineering students to deploy several end-to-end dataset creation applications in about two weeks of work, including system understanding and installation, while an expert developer can learn and deploy an instance of the system in about one working day. Moreover, our system allows users to inspect and manage uploaded datasets through a powerful GUI. On the other hand, it requires a FD to send data to the cloud.

Table 4.7 complements the qualitative comparison of Table 4.6 by providing a more explicit interpretation of the main characteristics of the reviewed solutions. In particular, it helps clarify that several competing systems rely on commonly

available wearable sensors and may share some architectural elements with the proposed approach, but are typically tailored to a specific sport, activity domain, or experimental setting. By contrast, the proposed solution is multi-sport by design, since the same acquisition and data-management workflow can be reused across different sport scenarios with limited reconfiguration effort.

Table 4.6: Functional feature comparison between data collection systems.

Solution	**Wearable**	**End-to-End Architecture**	**Open Source**	**Fog Device**	**Multiple EDs**	**GUI Dashboard**	**Cloud Connection Required**	**Multi-Sport Ready Deployability**
[121]		✓					✓	
[98]	✓	✓						
[122]	✓							
[99]	✓	✓			✓		✓	
[100]	✓	✓						
[123]	✓	✓		✓	✓	✓		
[124]	✓	✓		✓	✓	✓	✓	
[125]	✓	✓	✓	✓			✓	
Ours	✓	✓	✓	✓	✓	✓	[1]	✓

[1] Internet is not required while recording data, data will be stored locally until a (stable) internet connection is available.

Table 4.7: Descriptive comparison between data collection systems.

Solution	Wearable	End-to-End Architecture	Open source	Fog Device	Multiple EDs	GUI Dashboard	Cloud Connection Required	Multi-Sport Ready Deployability
[121]	smartphone-based	mobile + server HAR	no	none	single device	mobile app only	server-assisted	generic HAR, not sport-oriented
[98]	Arduino wearable	edge inference + server training	no	none	single device	none reported	training-side server	single use-case HAR
[122]	smart glove	sensing + local/app processing	no	none	single glove	none reported	no	single-sport / gym-focused
[99]	GNSS sports wearable	wearable + receiver + PC	no	data receiver	team version supported	PC software	no	soccer-specific
[100]	wearable localization node	sensing + fusion pipeline	no	none	single node	host PC	no	motion analysis, not multi-sport
[123]	inertial sensor network	sensors + PC pipeline	no	PC gateway	two body nodes	PC interface	no	daily/sport recognition, fixed setup
[124]	IMU wearable IoT node	IoT-edge + cloud	no	IoT-edge server	multiple sensors	web dashboard	yes	smart-home, not sport-oriented
[125]	edge-capable IoT nodes	programmable edge-to-cloud	yes	edge runtime	configurable	none native	yes	generic IoT, not sport-specific
Ours	wearable ED	ED + FD + cloud	yes	smartphone FD	multiple ED	cloud GUI	optional after session[1]	multi-sport by design

[1] Internet is not required while recording data, data will be stored locally until a (stable) internet connection is available.

In order to support further research in the field, the proposed framework is published as open source and available here: ED: https://github.com/measurify/edge-meter (accessed on 31 January 2026); FD: https://github.com/measurify/smart-collector (accessed on 31 January 2026); Cloud: https://github.com/measurify/server (accessed on 31 January 2026).

Overall, the experimental results confirm the effectiveness of the proposed end-to-end workflow for multi-sport activity recognition on resource-constrained devices.

Across four sports and more than 650 minutes of real-world recordings collected from 40 athletes, the system achieved classification accuracies between 93.8% and 97.5% using simple 1D-CNN models, validating the quality of the collected datasets and the correctness of the end-to-end pipeline.

From a deployability perspective, power measurements show that continuous operation at 10 Hz allows more than 4 hours of battery life on the wearable edge device, with average power consumption below 200 mW. These results demonstrate that accurate real-time activity recognition can be achieved on ultra-low-power wearable platforms without sacrificing autonomy, supporting the suitability of the proposed architecture for practical, field-deployable sport applications.

4.2 From Data Collection to On-Device Inference: Tennis Shot Recognition

After defining and validating the end-to-end edge-to-cloud workflow for data acquisition, we now focus on the ML stage of the framework. In this part, the collected datasets are used to train and deploy lightweight neural models on embedded hardware.

Figure 4.11 illustrates the overall process: starting from the previously described data collection through wearable sensors, the workflow proceeds with model training, conversion to the TensorFlow Lite format, deployment on a microcontroller, and real-time inference directly at the edge.

Parts of this chapter has been published in Dabbous, A., Fresta, M., Bellotti, F., Berta, R. (2024). Arduino Nano-Based System for Tennis Shot Classification. In: Ciofi, C., Limiti, E. (eds) Proceedings of SIE 2023. SIE 2023. Lecture Notes in Electrical Engineering, vol 1113. Springer, Cham. https://doi.org/10.1007/978-3-031-48711-8_43.

The material has been adapted and extended to fit the structure and scope of this thesis.

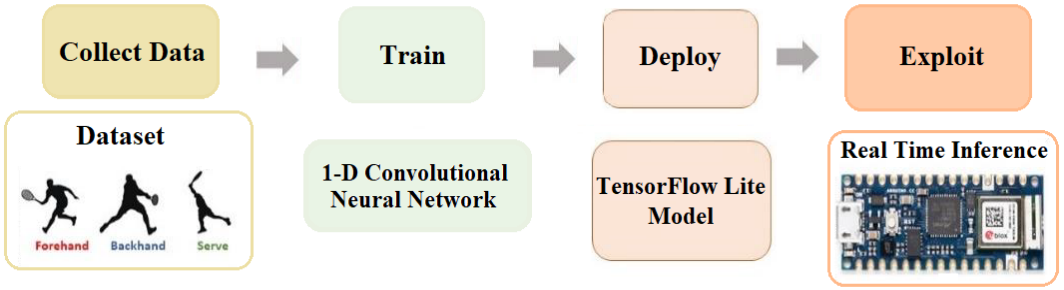


Figure 4.11: End-to-end workflow.

This section describes the complete ML experimental pipeline developed to validate the proposed architecture through sport and human activity recognition tasks. In particular, we present the dataset acquisition for tennis shots, the design of a compact 1D-CNN, its training and optimization, and finally its deployment on an Arduino Nano platform for real-time classification.

4.2.1 Experimental Setup

4.2.1.1 Dataset

The dataset employed in this work is made up of 6-axis time-series maps for three distinct tennis shots: forehand, backhand, and serve. To capture these shots, we asked an amateur tennis player to make several repetitions of the three types of shots, using his racket, to which an Arduino Nano IoT [126] device was connected. The player involved in this initial data collection was an amateur tennis player and was selected in order to provide sufficiently regular and well-defined executions of the considered shots, thus reducing the presence of ambiguous samples in the training set. During each trial, the time-series data from the IMU (LSM6DS3) sensor was recorded. This time-series has six axes: x, y, and z accelerometers and x, y, and z gyroscopes. The sensor was mounted on the racket because this position directly captures the motion dynamics of the shot and therefore provides a highly informative signal for discriminating between stroke types. The total number of samples collected for all shots is 213 divided into 80% for training and 20% for validation. Additionally, 45 separate time-series samples were recorded for the three shots for testing purposes.

4.2.1.2 1-D Classifier Network Architecture

Figure 4.12 shows the architecture of the proposed neural network, which consists of two 1D convolutional layers, interleaved by two max pooling layers, followed by a dense layer, and a three neuron softmax output layer. The convolutional layers aim at extracting the classification features, while the max pool layers, with non-overlapping receptive field, aim at reducing dimensionality. We discuss the actual hyperparameter values (e.g., number of neurons and of filters) later in the next section describing the training, which is where the actual values were set.

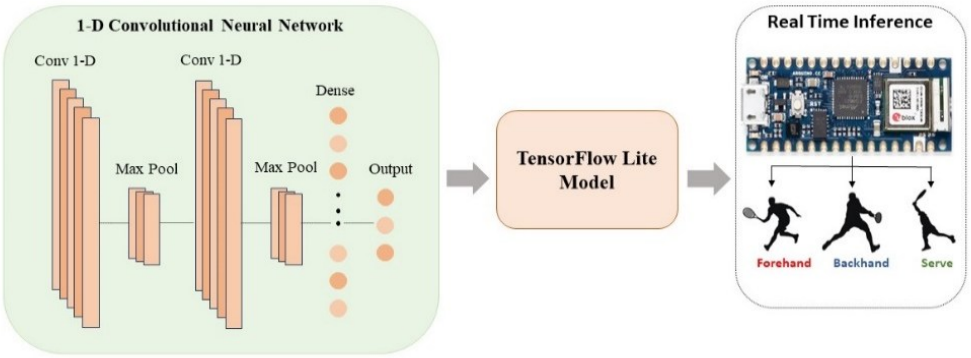


Figure 4.12: Tennis shot architecture. (Left) 1D-CNN. (Right) Arduino Nano 33 IoT for model deployment and real time classification.

4.2.1.3 Training Strategy

The first main challenge for the training consisted in maintaining consistency in the input shape of the 1D-CNN model, given the different lengths of the shot sequences. To this end, we employed Dynamic Time Warping (DTW) [127] techniques on the samples of each shot. This allowed us to clean up the dataset by identifying and excluding anomalous or poorly created samples. After applying DTW, we calculated the maximum number of time steps across all the samples, which was 20 in our case. For samples with fewer than 20 time-steps, we used padding techniques to maintain homogeneity. This entailed adding more time steps to ensure that all samples had the proper length. As a result, our 1D-CNN model's input shape was specified as (20, 6), where 6 denoted the number of

features recorded by each time step.

The actual training procedure then starts with fine-tuning the classifier architecture's hyperparameters, through cross-validation on the training set. A grid of potential candidates was investigated, with a focus on solutions with few parameters and high computing efficiency. The process finally selected a network architecture with two 1-D convolutional layers. The first layer has 8 filters, whereas the second 16. The kernel size in both layers is 3. The Dense layer comprised 32 units. The Rectified Linear Unit (ReLU) activation function is applied to both the convolutional and dense layers. Nevertheless, the total number of trainable parameters was equal to 2,219.

The training strategy's efficacy was assessed by measuring the model's accuracy on the testing set, after 50 epochs.

4.2.1.4 Embedded System for Real Time Classification

We deployed the 1D-CNN on an Arduino Nano 33 IoT, with the goal to classify shots in real-time. The Arduino Nano 33 IoT is a cost-effective device that can be easily installed into any tennis racket. It is powered by a SAMD21 Cortex®-M0+ 32-bit low power ARM MCU with a 48MHz CPU. The device includes 256KB of flash memory and 32kB of SRAM, and it operates at a voltage of 3.3V.

We were able to export the trained model in the float32 format suitable with the Arduino Nano 33 IoT after finishing the training of the 1D-CNN using Python and the TensorFlow library. The model was converted to the TFLite format, which is suitable for deployment on low-resource devices, including the Arduino Nano microcontroller. The embedded memory footprint of the proposed neural network is of 14KB.

4.2.2 Experimental Results

To evaluate the model's performance, we utilized a test set in which each one of the 45 samples represents an individual shot: 15 forehand strokes (red color in Figure 4.13), 14 backhand (blue), and 16 serves (green). The trained model could

correctly classify each shot in the test set.

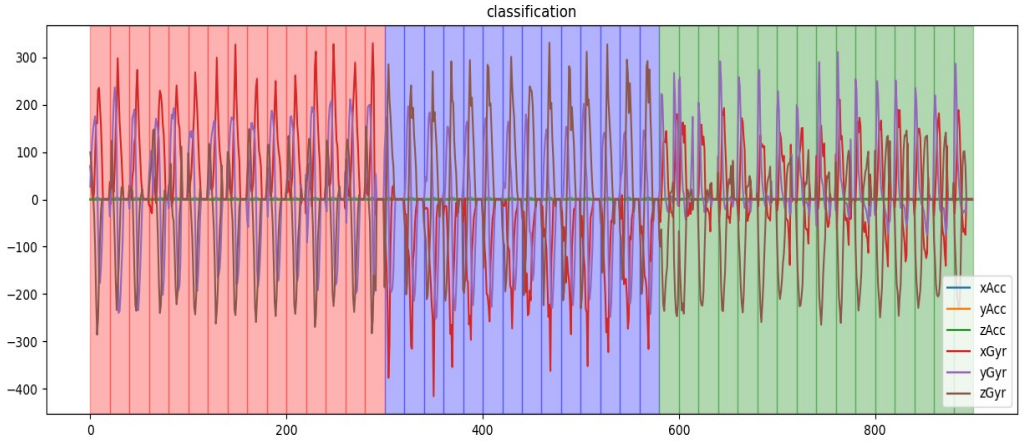


Figure 4.13: Tennis shots classification results. Red represents the Forehand shot, blue represents the Backhand shot and green represents the serve shot.

As a field-validation step, the model deployed on the Arduino Nano 33 IoT was evaluated with three distinct players, each of whom repeated three separate shots 15 trials. None of these players coincided with the subject used for the initial training data collection; the group consisted of one beginner, one regular player, and one amateur player. The complete system also included a threshold approach on the accelerometer measurements to assess whether there was movement and thus compute the classification.

Also in this case, the embedded model could detect and classify all the shots, as shown in Table 4.8. The inference time was 65ms, which is suited for real-time performance. These findings show that the proposed network is effective in terms of accuracy, and, advancing the state-of-the-art work [128], [129], has been deployed on an embedded device, executing real time inference.

Table 4.8: Real time tennis shots classification.

	Forehand	**Backhand**	**Serve**	**Inference Time**
Forehand	45	0	0	65 ms
Backhand	0	44	1	
Serve	0	1	44	

The obtained results confirmed that the trained model achieved reliable classification performance while maintaining a compact memory footprint and real-time execution capabilities on embedded hardware. These outcomes validate the proposed edge-to-cloud workflow as an effective solution for data acquisition, model training, and deployment directly on microcontroller-based platforms.

The experimental results demonstrate that the proposed model achieves reliable and fully deployable tennis shot classification on a microcontroller-class device. On the offline test set of 45 shots (15 forehand, 14 backhand, and 16 serves), the model achieved 100% classification accuracy. Field validation on three players, for a total of 135 real shots, confirmed robust real-time performance, with only two misclassifications observed.

When deployed on the Arduino Nano 33 IoT, the model achieved an inference time of 65 ms per sample, fully compatible with real-time operation. These results advance the state-of-the-art by demonstrating that accurate tennis shot classification can be executed entirely on a low-power microcontroller, without reliance on external computation, validating both the proposed model design and the end-to-end edge-to-cloud workflow.

4.3 Skiing Technique Classification on Embedded Devices

Following the validation of the workflow on the tennis shot recognition task, we extended the same methodology to a different sport characterized by more complex and continuous motion patterns, in order to further assess its flexibility and scalability. In this second use case, the framework was applied to the classification of skiing techniques, using wearable IMU data collected through the same edge-to-cloud architecture and processed according to the methodological principles previously described.

4.3.1 Experimental Setup

4.3.1.1 Dataset

For this study a dataset composed of 6-axis time-series data (x, y, and z from both the accelerometer and gyroscope) has been collected representing four specific skiing techniques: drifting, snowplow-steering, push off, and tuck. To focus specifically on the performance of a single athlete and assess the validity of the data collection method [130], an amateur skier performed multiple ski slopes equipped with a custom device featuring an Arduino Nano 33 BLE Sense with a custom battery board [131]. The IMU (LSM9DS1) sensor on this device captured the time-series data for the duration of the descent on the ski slope. The data collected by the Arduino was transmitted via BLE to a mobile phone and uploaded as CSV files to the Measurify cloud platform [132], [133]. To collect the dataset, the board was placed on the athlete's chest to gather IMU data on the movements during the descent. The data were collected towards two ski resorts with a total of 13 different ski slopes. In total, 39 slopes were gathered, with 85% used as the training set and 15% as validation set. Additionally, for testing purposes, 4 longer slopes were collected containing all the techniques in sequence. The test data were collected from the same amateur skier involved in the training, validation, and test phases, but from different descents and ski slopes, so as to keep the training, validation, and test sets separated and avoid direct contamination across splits. The choice of an amateur skier was motivated by the need to collect sufficiently regular and recognizable executions of the considered techniques under realistic field conditions. At the same time, the use of a single subject reflects the practical difficulty of data collection in this context, which had to be carried out on ski slopes, and should therefore be interpreted as a limitation in terms of inter-subject generalization.

Parts of this chapter has been published in Fresta, M., Bellotti, F., Capello, A., Cossu, M., Forneris, L., Berta, R. (2025). Classification of Skiing Techniques on Embedded System. In: Valle, M., Gastaldo, P., Limiti, E. (eds) Proceedings of SIE 2024. SIE 2024. Lecture Notes in Electrical Engineering, vol 1263. Springer, Cham. https://doi.org/10.1007/978-3-031-71518-1_58.
The material has been adapted and extended to fit the structure and scope of this thesis.

The dataset was collected with 20 Hz sampling frequency and for the classification we considered windows of 40, 60, and 80 data points respectively.

4.3.1.2 1D Classifier Network Architecture

To identify optimal hyperparameter values of the 1D-CNN (particularly: number of convolutional layers, number of filters, number of neurons for the first dense, and learning rate), we employed Bayesian optimization (BO) using the Keras tuner library to assess 20 combinations of hyperparameters over 40 epochs. Through this method, we reach good accuracy with 80 data points, and we obtained as optimal configuration a network with 3 convolutional layers, 32 filters, 32 neurons, and a learning rate of 10e-3.

Figure 4.14: shows the resulting structure of 1D Convolutional Neural Network (1D-CNN) designed for classification. This network is composed of three 1D convolutional layers, interlined by a max pooling layer. This sequence is followed by two dense layers and a softmax output layer with four neurons, corresponding to the four classes.

The primary function of the convolutional layers is to extract features relevant for classification, whereas the max pooling layers, which have non-overlapping receptive fields, are designed to reduce the dimensionality of the data.

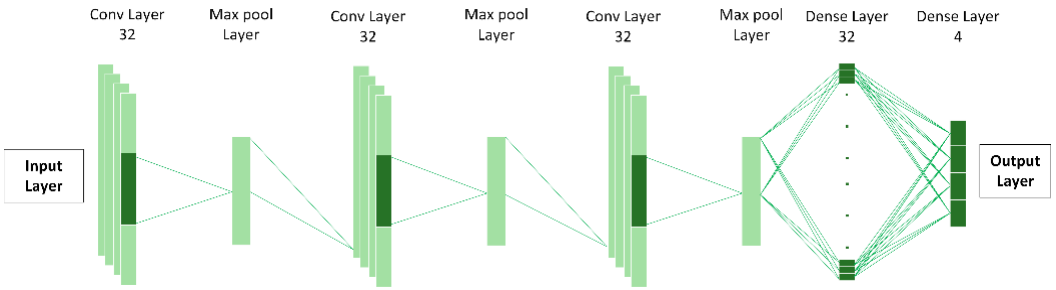


Figure 4.14:1D convolutional neural network.

4.3.2 Experimental Results

To evaluate the model's performance, we used the test set that includes more than 10 samples for each of the four techniques classified.

Figure 4.14: illustrates two key comparisons: on the left, it shows the comparison between the ground truth and the model's predictions for a segment of test sequence number 1; on the right, it presents the normalized confusion matrix of the best model.

The confusion matrix reveals that the drifting technique has the highest accuracy, while the tuck technique has the lowest.

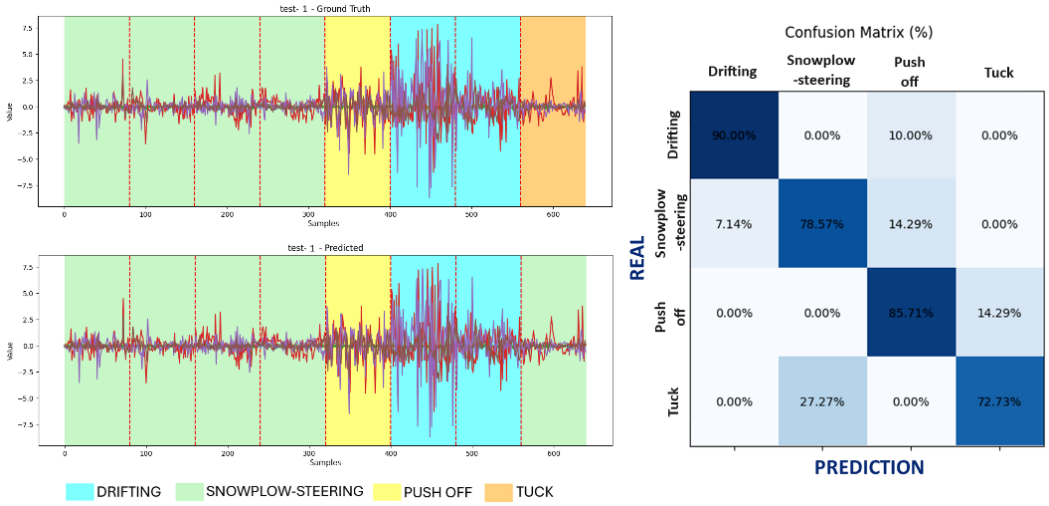


Figure 4.15: (left) Comparison of ground truth and prediction, (right) Normalized confusion matrix of the 1D-CNN model.

Table 4.9 shows the performance of the trained models over the test set and the size of the models. For the deployability, the model was converted to TFLite with a decreasing in model size without a loss of accuracy. Additionally, to further reduce model size we also used the TFLite quantization of weights to 8-bit integers, but this led to a loss in accuracy as shown in Table 4.9.

Table 4.9: 1D-CNN model performance and size.

Model	**Accuracy(%)**	**Precision(%)**	**Recall(%)**	**F1-score(%)**	**Size (KB)**
1D-CNN (Baseline)	85.58	87.86	85.58	86.71	232.46
TFLite	85.58	87.86	85.58	86.71	66.50
Quantized	80.92	85.63	80.92	83.21	26.04

For the performance studies, the two models were deployed on the Arduino Nano 33 BLE Sense.

Table 4.10 demonstrates good performance in terms of inference time, power and energy consumption. In particular, power consumption was measured by observing the power variations during inference using a JT-UM120 USB Multimeter.

Table 4.10: Model performance on Arduino Nano 33 BLE Sense.

Model	**Inference time (ms)**	**Power consumption (mW)**	**Energy consumption (mJ)**
TFLite	78.13	15.20	1.19
Quantized	26.37	14.10	0.37

The experimental results confirm that the proposed 1D-CNN model is able to accurately classify skiing techniques while remaining suitable for deployment on resource-constrained embedded hardware. On the test set, the baseline model achieved an accuracy of 85.58%, with the drifting technique being the most reliably recognized and the tuck technique the most challenging.

Model conversion to TensorFlow Lite reduced the memory footprint from 232.46 KB to 66.50 KB without any loss in accuracy, demonstrating the effectiveness of lightweight deployment formats for embedded inference. Further 8-bit quantization reduced the model size to 26.04 KB, at the cost of a moderate accuracy drop to 80.92%.

When deployed on the Arduino Nano 33 BLE Sense, the TFLite model achieved an inference time of 78.1 ms with an energy consumption of 1.19 mJ per inference, while the quantized version reduced inference time to 26.4 ms and energy consumption to 0.37 mJ. These results highlight the trade-off between accuracy and efficiency, and demonstrate that real-time, low-power skiing technique recognition is feasible on microcontroller-class devices, advancing the state-of-the-art in embedded sport activity recognition.

CHAPTER 5

5. Structural Health Monitoring

Structural Health Monitoring (SHM) aims at the continuous observation of civil infrastructures in order to detect, localize, and characterize damage before it compromises safety or serviceability [13]. Modern SHM systems rely on dense sensor networks (Figure 5.1), typically composed of accelerometers and other vibration sensors, which generate long, multichannel time-series under varying environmental and operational conditions [70].

Figure 5.1: Bridge Health Monitoring

These signals pose significant challenges for data management and analysis: they are high-dimensional, noisy, non-stationary, and often collected over extended time spans.

From a data-centric perspective, SHM shares many characteristics with the time-series scenarios discussed in the previous chapters, while introducing additional

constraints related to safety-critical operation, interpretability, and long-term monitoring. Effective SHM solutions must therefore combine robust data management, scalable time-series processing, and reliable ML models that can operate under realistic computational and energy constraints, potentially close to the sensing devices.

In this work, the Measurify platform is employed as the backbone for managing vibration datasets, storing raw and processed measurements as time-stamped entities that can be systematically retrieved for analysis and model training. To map the collected data to the Measurify resource model, the Thing was defined as "Z24Bridge", representing the monitored infrastructure, the Device was set to "SensorN", where N identifies the specific accelerometer used in the acquisition, and the Feature was defined as "Acceleration", corresponding to the vibration time series recorded by each sensor. The Model Registry introduced in Chapter 3 is used to organize datasets, algorithms, and trained models, enabling reproducibility and traceability across experimental configurations. This infrastructure allows SHM experiments to be conducted within the same end-to-end workflow adopted for other application domains, reinforcing the generality of the proposed methodology.

Building upon this foundation, the chapter investigates state-of-the-art time-series ML approaches for vibration-based SHM, with a specific focus on multi-class damage classification. We explore both deep learning architectures capable of learning discriminative representations directly from raw signals and lightweight methods designed for efficient deployment. Particular attention is given to WaveNet-based models and MiniRocket feature extractors, which have demonstrated strong performance on long and complex time-series, as well as to Binary Neural Networks, which enable extreme model compression and low-power inference on embedded platforms.

To enable a clear and reproducible comparison, the chapter focuses on the Z24 Bridge dataset [134], a widely studied benchmark for vibration-based SHM. This dataset provides vibration measurements collected under multiple structural states and environmental conditions and has become a reference in the SHM literature

[135]. Through this case study, we evaluate classification performance, deployability on edge devices, and interpretability aspects, highlighting the trade-offs between model complexity, accuracy, and resource consumption.

Overall, this chapter extends the end-to-end time-series methodology developed in the thesis to the SHM domain, demonstrating how advanced learning models, efficient feature extraction, and embedded-friendly architectures can be integrated within a unified data management and deployment framework for safety-critical monitoring applications.

5.1 Vibration-Based SHM on the Z24 Bridge

Our study relies on the Z24 Bridge dataset [134], which contains vibrational measurements collected from a bridge in the canton Bern, near Solothurn, Switzerland. This dataset is widely used in Structural Health Monitoring (SHM) to study the dynamics of the entire structure of the bridge under different damage and weather conditions. The bridge was built in 1963 to connect the villages of Koppigen and Utzenstorf, spanning the A1 highway between Bern and Zürich. The bridge's design featured a concrete two-cell box-girder structure with a principal span of 30 meters and two additional side spans of 14 meters each as shown in Figure 5.2. In 1998, the bridge was demolished. Before and during the demolition, the state of the bridge was monitored through a network of accelerometers to understand the correlation between the measured vibrations and the overall state of the bridge.

Parts of this chapter has been published in A. Dabbous, R. Berta, M. Fresta, H. Ballout, L. Lazzaroni and F. Bellotti, "Bringing Intelligence to the Edge for Structural Health Monitoring: The Case Study of the Z24 Bridge," in IEEE Open Journal of the Industrial Electronics Society, vol. 5, pp. 781-794, 2024, doi: 10.1109/OJIES.2024.3434341.

The material has been adapted and extended to fit the structure and scope of this thesis.

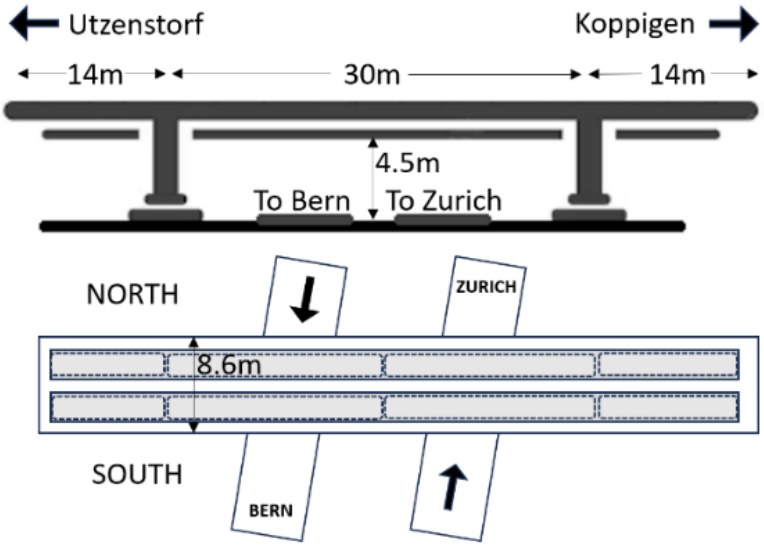


Figure 5.2 Lateral and top view of the bridge.

5.1.1 Dataset Structure

The Z24 Bridge dataset contains a comprehensive collection of structural and environmental data, specifically concerning bridge health monitoring and analysis of the dynamics of bridge behavior under various conditions. This dataset consists of two main types of data: long-term continuous EMS test data, capturing the bridge's response to natural weather conditions over one year (11 months before the demolition and 1 month during the demolition); and a short-term PDT, which records the bridge state under all the damage scenarios occurred during the demolition. The EMS data includes a collection of time-series taken hour-by-hour by 16 accelerometers placed along the entire bridge. However, the signal of only 6 of such accelerometers was included in the dataset as acceptable. The PDT collection records the signal of 291 accelerometers. The PDT data can be divided into two distinct subsets: the ambient vibration set, and the forced vibration set. The ambient vibration set measurements capture the bridge's response under environmental perturbations (e.g., wind). In contrast, the forced vibration set includes data obtained when the bridge was subject to deliberate vibrations obtained by placing two shakers on top of the bridge, allowing for a controlled study of the structure's response under these specific conditions. For our test, we

have utilized the PDT dataset ambient vibration test since we are interested in studying the bridge behavior in as natural as possible conditions. We have preferred PDT to EMS data because it includes much more sensors (291 instead of 6), which is key to more effectively classify the 15 classes of bridge conditions. Additionally, the EMS data focuses on the healthy state of the bridge over 11 months, while the separation between the different damage classes over a few days is not distinctly recorded. This makes the PDT data with its ambient vibration set more suitable for the classification task for many classes. Also, the dataset does not specify the nature and intensity of the environmental perturbation, requiring ML models to be generally robust with respect to them. Comprehensive information about the Z24 Bridge SHM program and tests can be found in [134].

The ambient vibration test dataset includes data from 291 uniaxial accelerometer sensors of Force Balance Accelerometers (FBA) types 11 and 23, distributed across the entire bridge's structure. The positioning was designed to maximize coverage and sensitivity, with sensors located at critical points where maximum stress and movement are expected.

As illustrated in Figure 5.3, the bridge was segmented into 9 sections and the accelerometers were placed along the length of the entire structure to ensure comprehensive data collection.

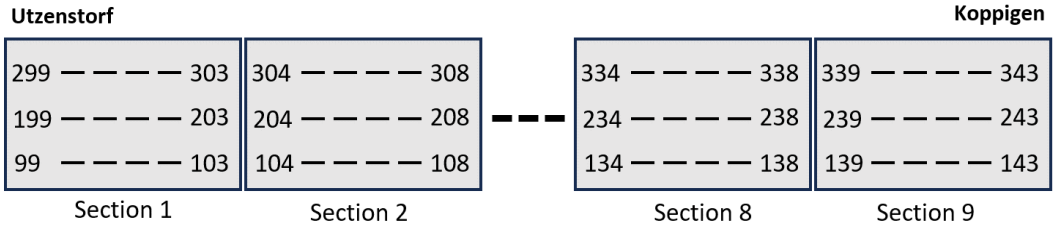


Figure 5.3: Positions of sensors across the entire bridge.

5.1.2 Demolition Scenarios And Data Sampling

The Z24 Bridge dataset comprises a total of 17 documented scenarios, each one carefully recorded to keep track of the dynamics of the bridge under different progressive damage conditions. Of these scenarios, we discarded the second one

because it concerns the deployment of the equipment to perform the lowering tests, and the eighth one because it is a reference scenario, not corresponding to any demolition progress. Table 5.1 presents the 15 selected scenarios, along with the corresponding number of accelerometer sensors per class. Each sensor provided a time-series of 65,536 steps, recorded at a frequency of 100 Hz.

Table 5.1: Scenarios specifications.

Scenario id	**Scenario Name**	**No. of Sensors**
1	Healthy state first reference	289
3	Lowering of Koppigen pier, 20 mm	258
4	Lowering of Koppigen pier, 40 mm	291
5	Lowering of Koppigen pier, 80 mm	291
6	Lowering of Koppigen pier, 95 mm	291
7	Lifting of Koppigen pier	291
9	Spalling of concrete at soffit, 12 m2	291
10	Spalling of concrete at soffit, 24 m2	291
11	Landslide of 1 m at abutment	291
12	Failure of the concrete hinge	291
13	Failure of 2 anchor heads	291
14	Failure of 4 anchor heads	291
15	Rupture of 2 out of 16 tendons	291
16	Rupture of 4 out of 16 tendons	291
17	Rupture of 6 out of 16 tendons	291

Like Santaniello and Russo [136], we removed 33 signals from the first section of the bridge (Figure 5.3) from the id 3 scenario because of the absence of more than 510 samples in each time-series from this section, and a divergence in the data patterns compared to those from other sections, as depicted in Figure 5.4.

The dataset does not specify a test split, and different studies have used different approaches. In light of common practices in ML, we advocate a rigorous approach by adopting a stratified split methodology and dividing the dataset into three distinct subsets: 70% for the training set, 10% for the validation set, and 20% for

the test set. Not only does this ensure an adequate sample size for training the model, but also facilitates robust assessment and fine-tuning of the hyperparameter through the dedicated validation set.

In this work, we adopt a distinctive approach by concentrating on refining our models through the utilization of the complete sequence dataset, eschewing any intermediary preprocessing steps. Notably, the input size for our models is configured as (65,536, 1), indicating that all the 65,536 time-steps of each accelerometer time-series are taken as univariate samples. This deliberate choice underscores our commitment to maintaining temporal fidelity and maximizing the exploitation of the information content inherent in the raw dataset. In particular, each accelerometer time-series is treated as an independent sample describing the structural state of the bridge under a given scenario. Therefore, the proposed ML pipeline does not aggregate measurements coming from different sensors into a single synchronized multichannel input. For this reason, no additional synchronization or resampling across sensors was required. Moreover, no further preprocessing was intentionally applied, in order to preserve an end-to-end setting and allow the models to extract the relevant features directly from the raw vibration signals.

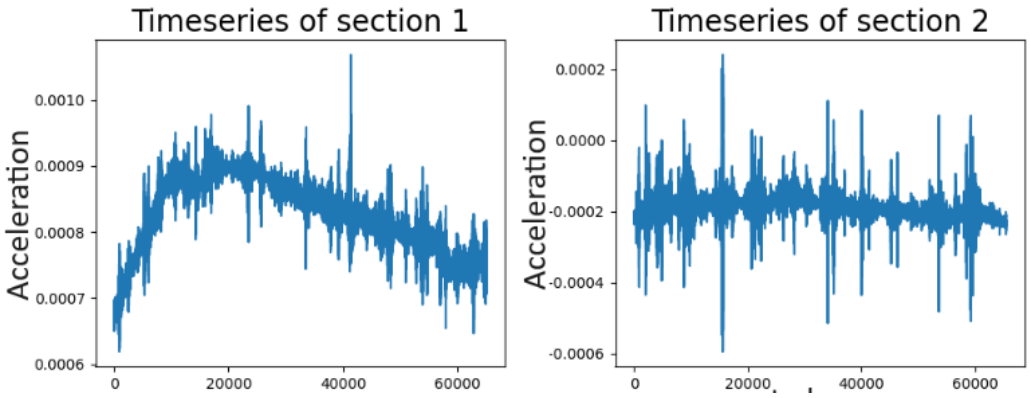


Figure 5.4:Comparison of acceleration data of section 1 (left) and section 2 (right) of the id 3 class.

5.1.3 Deep Learning Models

Our proposal on the assessment of the Z24 bridge state (i.e., classification of the damage scenarios reported in Table 5.1) relies on two state-of-the-art full-precision ML architectures, which we describe in the following.

5.1.3.1 WaveNet

WaveNet (WN) is a deep generative model of raw audio waveforms that has significantly improved the state-of-the-art operating on 1D data sequences [137]. We adapted this architecture to the damage classification task by customizing its output stack. One of the key features of WN is its use of dilated causal convolutions [138], which allows the network to reach a larger receptive field compared to regular convolutions, while ensuring that the output is only conditioned by past inputs, preserving the essential aspect of causality in time-series data. The network uses very efficient size 2 kernels only, with different dilation values (1 to 256).

The network begins with a causal convolution layer, with dilation 1 (as a regular convolution) (Figure 5.5). This setup captures the initial context of the input sequence. This layer is followed by a series of residual blocks [139], where the dilation rate doubles with each subsequent layer, allowing the network to aggregate information over larger temporal contexts without an exponential increase in the number of parameters and thus computation complexity. Each block extracts temporal features through gated activation [140]. A block starts with two dilated convolutions having the same dilation rate: the former employs a Hyperbolic Tangent (tanh) activation function; the latter a sigmoid (σ), whose output gates the previous values. This schema extracts features from the temporal dimension of the signal. The subsequent 1×1 convolution performs cross-channel operations [141]. The combination of dilated convolutions followed by the 1×1 convolution allows for a very efficient time-series convolution in terms of number of used parameters.

The transformed data is summed with the input of the residual block before going

into the next block, to allow the gradient to flow through the network more effectively during training, thus helping with the possible vanishing gradient problem and enabling deeper architectures. Additionally, the output of the 1×1 convolution serves as a skip connection, that connects the extracted information from each residual block to the output stack, ensuring that features identified at each block contribute to the final output.

After a ReLU activation, the output stack features two 1×1 convolution layers, that condense the learned features into a representation suitable for the next classification layers. Here, we replace WN's last layer (which was conceived to output a sequence) with two dense layers. The first one, with a ReLU activation, is followed by the last dense layer with SoftMax activation to compute the K class probabilities.

We implemented the WN using the TensorFlow [142] framework and its convenient Keras wrapper. To identify optimal hyperparameter values (particularly: the number of filters and dilation rates), we employed Bayesian optimization (BO) using the Keras tuner library. The BO adaptive procedure systematically assessed 20 combinations of hyperparameters, striking a balance between exploration of new regions and exploitation of promising ones. Through this method, we obtained as optimal configuration a network with 8 filters (reported as N in Figure 5.5) and 9 residual blocks, each one with a range of dilation rates from 1 to 256. Moreover, the number of units for the dense layer is 8. We trained the network from scratch, using the Adam optimizer with an initial learning rate of 0.0001, over 150 epochs.

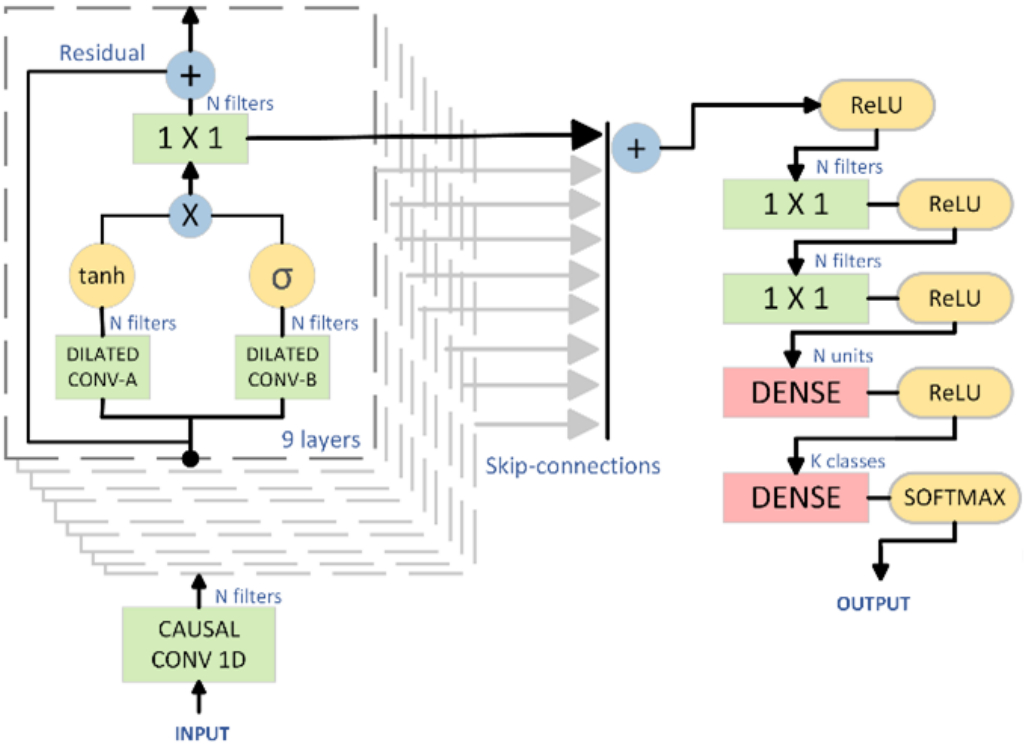


Figure 5.5:WaveNet model architecture.

5.1.3.2 MiniRocket

MiniRocket is known for its simplicity and outstanding training efficiency. According to the authors, it is the default version of the Rocket [143], which converts time-series data into features using random convolutional kernels. Such features are then utilized to train a linear classifier, as shown in Figure 5.6. In our implementation we used a Ridge classifier. MiniRocket (MR) reduces Rocket complexity, particularly its randomness so that it is a quasi-deterministic model [144]. For example, in Rocket, the length of the random kernel could be 7, 9, or 11, but is fixed to 9 in MR. Similarly, weights in Rocket are learned and initially sampled from a standard normal distribution, whereas in MR have fixed values of -1 or 2 only, for a total of 512 possible combinations, out of which only 84 fixed are used [144]. The values of biases are initially sampled from a (-1, 1) uniform distribution in Rocket, while are derived from the convolution output in MR, which is the only non-deterministic aspect of the model.

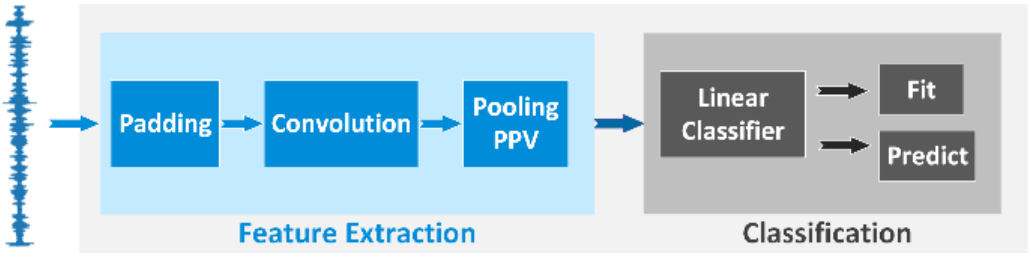


Figure 5.6: MiniRocket model architecture.

In the feature extraction phase, the kernels are convolved with the time-series. After convolutions, MR uses Proportion of Positive Values (PPV) pooling to reduce dimensionality and output the feature vector for the linear classifier.

MR exposes only two tunable hyperparameters: the maximum number of dilations per kernel and the total number of features. These hyperparameters are interrelated and we made an empirical study to select their best values for our case, whose outcomes are reported in Figure 5.7 and Figure 5.8, respectively. As candidates for the total number of features, we chose a set of multiples of the number of kernels (i.e., 84). Based on this tuning study, we selected the values of 64 as the maximum number of dilations per kernel, and 9,996 as the total number of features. We used the sktime implementation [145].

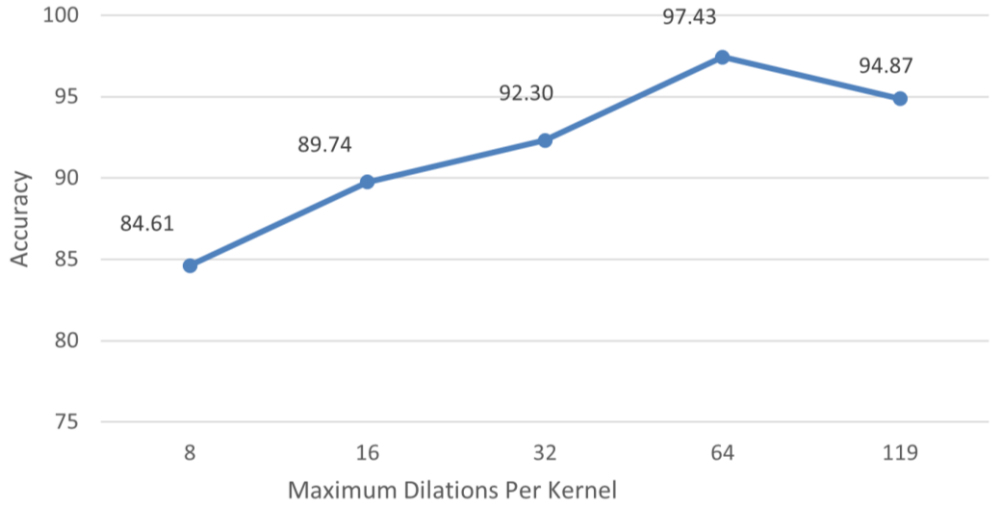


Figure 5.7: MiniRocket performance as a function of maximum dilations per kernel.

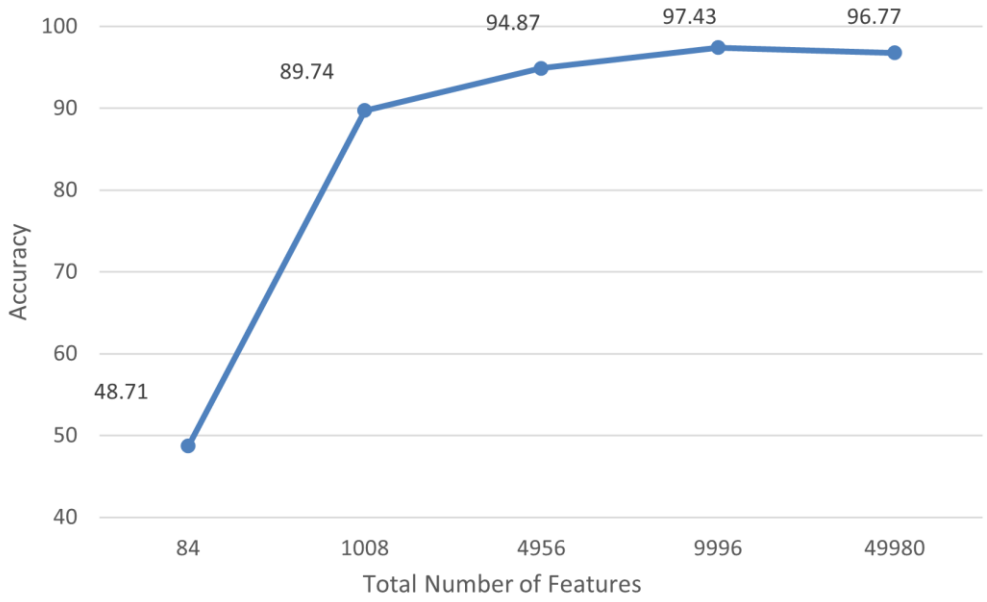


Figure 5.8: MiniRocket performance as a function of total number of features.

5.1.4 Experimental Results

As a first step of our assessment, we focus only on classes 1, 3, 4, 5, and 6 of the Z24 Bridge PDT dataset, which is the most targeted problem in literature ([136], [146]). These are the first classes of the dataset and are critical because they represent the initial phases of damage and, thus are the most critical ones in a progressive degradation scenario [136]. We trained the WN model on a laptop equipped with an NVIDIA RTX 4070 Laptop GPU, while the MR model was trained on the same laptop, exploiting its 2.20 GHz 13th Gen Intel® CoreTM i9-13900HX processor, without using GPU.

Results, obtained in the test set are reported in Table 5.2, and demonstrate the significant performance improvement brought by our models. The WN classifier achieves an impressive accuracy of 99.65% and largely outperforms the state-of-the-art (Santaniello and Russo [136]) also in terms of model size.

Table 5.2: 5-class problem performance comparison.

Models	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	MAC	# Parameters	Model size (MB)	Activation size (MB)	Training time (min)
SCWT+ RN50 [136]	97.50	97.77	97.34	97.51	N/A	40,105,093	> 98	N/A	N/A
WaveNet	99.65	99.66	99.65	99.65	215,482,383	2,624,781	31	125.95	50
MiniRocket	95.96	96.01	95.96	95.96	2,774,619,713	97,028	0.67	3.61	5

The MR confirms the mentioned peculiarities [144], achieving an accuracy comparable with the state-of-the-art but with minimal model size and training time. In Santaniello and Russo [136], the authors omitted details regarding the model footprint and training time, arguably because their work did not target an embedded deployment. However, we can observe that the pre-trained ResNet50 model they utilized is 98 MB, with 25M parameters according to the information provided in the Keras documentation. Notably, in [136], the total number of parameters is reported to be 40M, suggesting the incorporation of additional operations and layers, thereby contributing to an increased model size. Finally, it is important to stress that all our results were obtained without any preprocessing steps, in an end-to-end ML approach.

The second experiment concerned the 15-class problem, including also the other nine classes of the Z24 benchmark, up to the rupture of six tendons. These classes are relevant, particularly in case of a dramatic event (e.g., earthquake, land/rockslide), and their consideration could lead to a more complete analysis and tool. Only very recently has this full-dataset problem been addressed in literature, by Le-Xuan et al. [147]. Results in Table 5.3 show the neat superiority of both our proposed models.

Table 5.3: 15-class problem performance comparison.

Models	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	MAC	# Parameters	Model size (MB)	Activation size (MB)	Training time (min)
1DCNN-LSTM-RN [147]	81.50	82.01	81.51	81.41	N/A	5,117,808	N/A	N/A	N/A
WaveNet	98.97	98.99	98.97	98.97	220,725,293	7,867,671	92.50	146.63	80
MiniRocket	91.29	91.46	91.29	91.30	2,774,719,683	196,998	1.45	4.01	21

The WN architecture reaches an outstanding accuracy of almost 99%, achieving results close to the 5-class problem. [147] has a lower number of parameters but does not report the corresponding model size. The test reports a 4.7% accuracy drop for the MR, instead, hinting at a lower capability of this model to deal with the increased complexity. Arguably, the complexity of the problem calls for learnable kernels, like the WN's ones. Compared to the 5-class problem the WN augmented its size by a factor of 3, while MR more than doubled its size. It is important to notice that the number of multiply and accumulate (MAC) operations for the MR is one order of magnitude higher than for the WN, which may be attributed to the MR's much larger kernel sizes. MR confirms its extreme training efficiency, even if the increment in training time with respect to the 5-class problem is not dramatic for WN either. For better evidence, Figure 5.9 depicts ML performance metrics and resource utilization for both the 5- and 15-class problems.

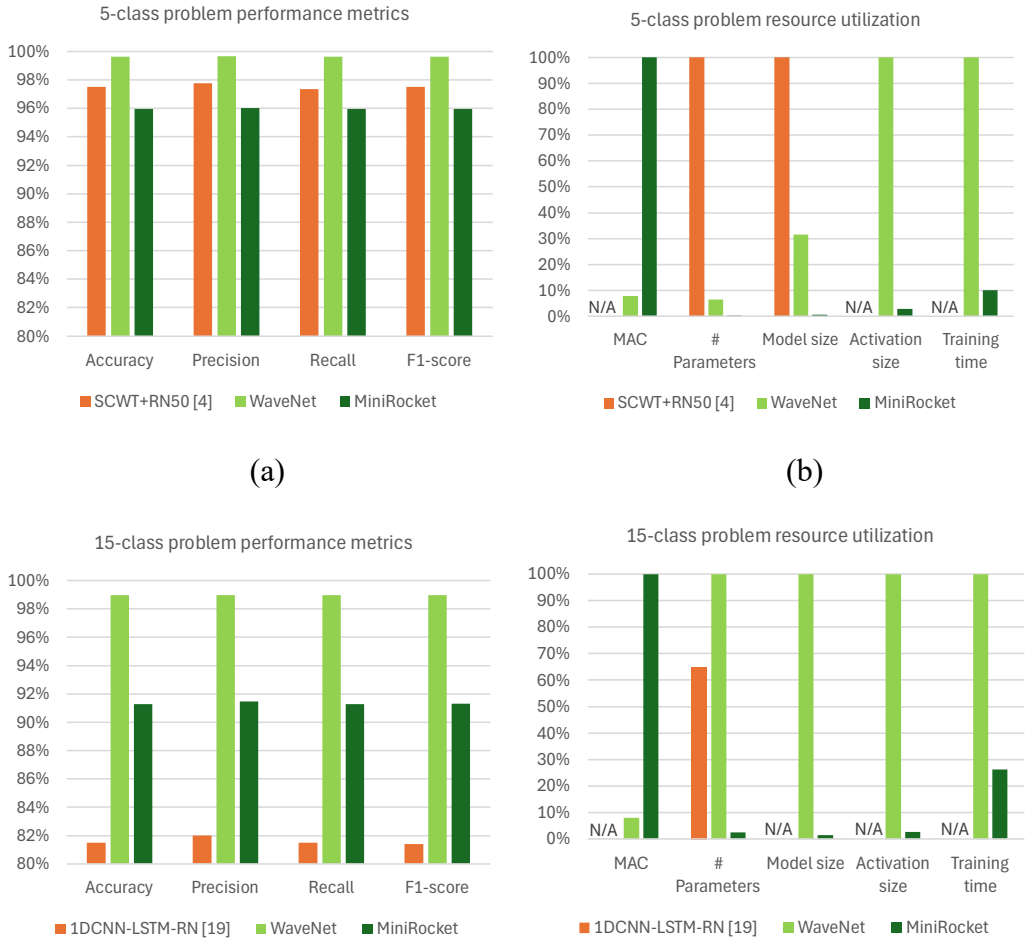


Figure 5.9: Performance and resource utilization comparisons for the 5 and 15 class problems. Resource utilization is relative to the highest value.

Resource utilization is expressed in percentage, relative to the worst model's value. Tables 5.2 and 5.3 report the comparison with the state-of-the-art only (i.e., Santaniello and Russo, and Le Xuan et al., respectively). Comparisons with lower-performance models can be found in [136] for the 5-class problem, while [147] is the only one paper addressing the whole 15-class problem, to the best of our knowledge.

The third part of our experimental analysis concerned the deployment of our models on two target devices employable in the field: a Raspberry Pi 4 model B, and a Jetson AGX Xavier. As a reference, we measure performance also on a laptop equipped with NVIDIA GeForce RTX 4070 Laptop GPU (Table 5.4).

Table 5.4: Edge Devices specifications.

Device	RAM (GB)	Processor	Frequency (GHz)	Max Power (W)
Raspberry Pi 4	4	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit	1.50	8
Jetson	32	512-core NVIDIA Volta architecture GPU with 64 Tensor Cores	1.37	30
Laptop	8	NVIDIA GeForce RTX 4070 Laptop	2.47	200

Despite its compact size and affordable price, the Raspberry Pi 4 can execute deep learning tasks efficiently, through a quad-core ARM Cortex-A72 processor. It includes built-in Wi-Fi and Bluetooth protocols for data communication. NVIDIA's Jetson Xavier series is known for powerful system-on-module designs tailored for edge computing applications and excels in deep learning tasks. It features high-performance GPUs, multi-core CPUs, and dedicated AI accelerators. While providing faster precision and decision-making, the Jetson series consumes more power than the Raspberry Pi and is much more expensive.

Table 5.5 reports the execution time, power consumption, and energy consumption for the proposed models across the mentioned edge devices.

Table 5.5: Deep Learning model performance on hardware devices.

Model	Raspberry Pi 4 Model B			Jetson AGX Xavier GPU			NVIDIA GeForce RTX 4070 Laptop GPU		
	Inference time (ms)	Power consumption (mW)	Energy consumption (mJ)	Inference time (ms)	Power consumption (mW)	Energy consumption (mJ)	Inference time (ms)	Power consumption (mW)	Energy consumption (mJ)
WN	575	650	374	118	364	43	79	450	36
MR	6,276	1,000	6,275	4,251	500	2,125	2,287	551	1,260

As a reference, the total idle power consumption of the considered platforms is on the order of about 1.8 W for the Raspberry Pi, 5 W for the Jetson, and 20 W for the PC. These values are reported only to better contextualize the inference-related measurements, and should therefore be interpreted as approximate

platform-level baseline figures rather than as strictly comparable experimental results.

To highlight the comparison between the different models, Figure 5.10 depicts the same quantities, in percentage, relative to the worst model's value.

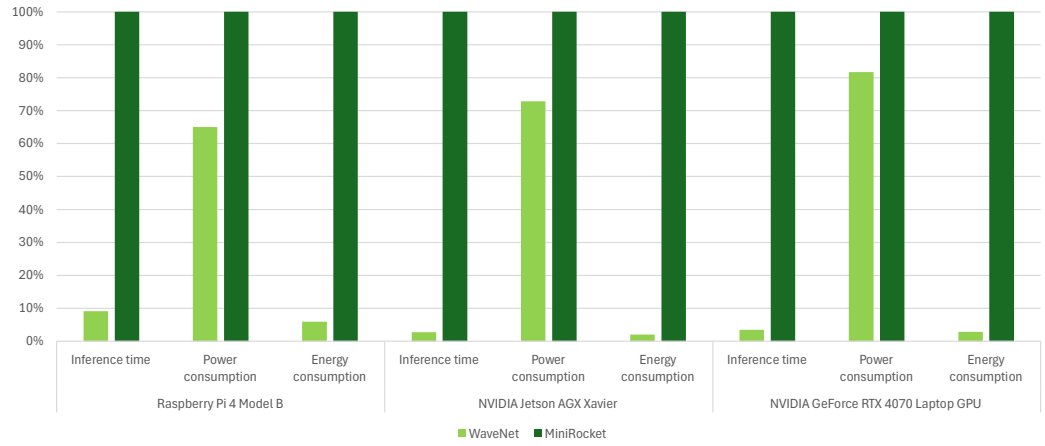


Figure 5.10: Deep Learning model performance on hardware devices. Charts show values relative to the highest.

We employed different methods to measure power metrics. On the Jetson board, we used the Jetson power logging feature. On the Raspberry Pi, we employed a USB multimeter connected between the power supply and the device to track power variations. On the laptop, we utilized the "nvidia-smi" utility. Accordingly, power is measured at GPU level for laptop and Jetson, but at whole board level for the Raspberry Pi. Energy is then computed by multiplying power by inference time. Reported values are averages over 100 test runs.

Results stress the relatively low energy consumption of the WN (0.04 Joule on Jetson vs. 0.1 Joule of ResNet on RTX 2080 Ti [148], that we use as a reference given the lack of this kind of measurements in the Z24 Bridge literature). On all the platforms, MR has a much higher power and energy consumption than the WN. This corresponds to the higher computational complexity we have discussed above. The laptop outpaces the Jetson device in inference time by approximately two times, for both models. This may be attributed to the laptop's more powerful GPU. However, this speed advantage comes at the cost of higher power consumption. On the other hand, the Jetson device exhibits greater overall energy consumption, because of the longer execution time.

In the Arm-based Raspberry Pi's architecture, MR has longer execution times and higher power and energy consumption. This trend aligns with the observations made on the Jetson Xavier platform.

Compared to the Jetson and laptop devices, the Raspberry Pi has considerably higher inference times for both our models. Measured power and energy is higher as well, but these measurements for the Raspberry concern the whole board, not only the GPU. On the other hand, Raspberry Pi's price is two orders of magnitude lower.

These findings underscore the intricate relationships among model complexity, CPU and GPU capabilities, and power and energy consumption. Thus, the development of effective deep learning-based edge applications requires a nuanced understanding of their requirements and the capabilities of the underlying processing and communication architecture.

The presented results can be obtained through a system architecture, as depicted in Figure 5.11.

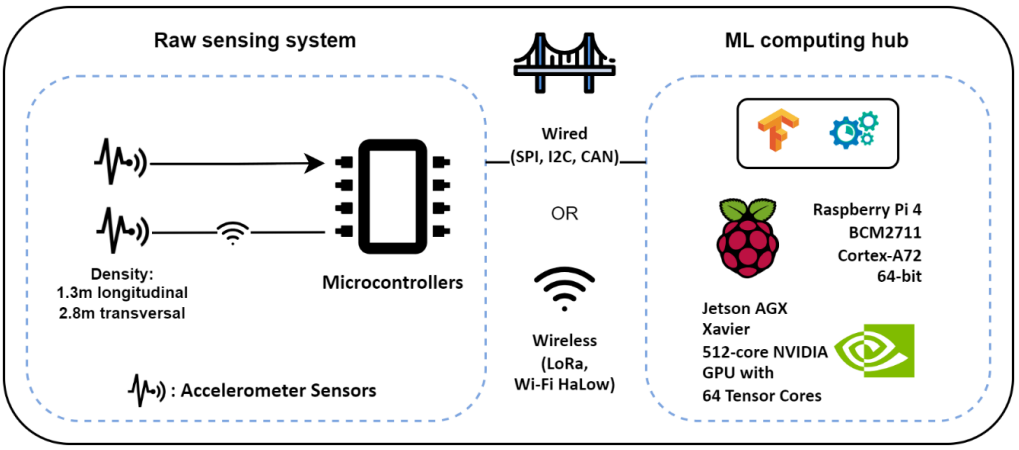


Figure 5.11: Architecture blocks for a bridge health monitoring system.

The sensing sub-system involves a set of microcontroller units (MCUs) equipped with accelerometer sensors, able to capture the vibrational signatures. Based on the Z24 Bridge data [134], a density of 1 tri-axial accelerometer each 1.3 meters longitudinal to the bridge and 2.8 meters transversal to the bridge could be sufficient to achieve the accuracy presented results. Each microcontroller could be connected to multiple accelerometer sensors, depending on the number of general

purpose input/output (GPIO) pins and the power [149].

Raw signals collected by the microcontroller have then to be delivered to the computing hub, to execute in real-time state-of-the-art ML models (e.g., WN and MR). This task is computationally intensive and resource demanding, overcoming the capabilities of mainstream microcontrollers. Particularly, the WN model footprint (31 and 92.5 MB for the 5 and 15 class problem, respectively) by far exceeds the typical MCU Flash size of 2 MB. The models' activation size is even larger, exceeding the RAM capacity of typical MCUs also in the MR case. However, our analysis has shown that the classification task can be effectively undertaken also by field-deployable edge devices. Particularly, the hardware devices in Table 5.4 are representative, respectively, of a low-cost single-board computer (Raspberry Pi 4) and a high-end ML-oriented embedded computing board (Nvidia Jetson AGX Xavier).

It is important to highlight that this architecture deals with runtime inference only, while the training of the ML models is performed offline, on desktop/cloud hardware, as we did for the training of the models presented in the previous section. The architecture is modular and generic and may be easily updated (e.g., with new classification algorithms) and customized (e.g., for pylons, masts, gates, industrial machines).

5.1.5 Feature Analysis

The experimental results have shown the outstanding ability of leading-the-edge deep convolutional neural networks to automatically extract features that allow effective and efficient handling of damage detection problems, by seamlessly processing raw signals output from accelerometers. This overcomes some of the most powerful state-of-the-art models used for SHM and advances significatively in creating accurate, affordable and low-power embedded SHM tools.

However, neural network models are black boxes, and an interpretation of the meaning of the extracted features is awkward, differently from the features defined by domain experts (e.g., modal features). According to the end-to-end ML

approach, this section presents an analysis of the features extracted by the convolutional layers of the two models for the most challenging 15-class problem. These are the 10K-dimensional input to the Ridge classifier for the MR (Figure 5.6), and the 524K-dimensional vector concatenated by the WN before its classification stack (Figure 5.5).

We performed the analysis using the SHapley Additive exPlanations (SHAP) method [150]. SHAP, which has a solid theoretical foundation in game theory [151], estimates the influence of each feature on each model's prediction. Assigning each feature an importance measurement is regularly used to quantitatively explain the output of black-box ML models (e.g., [152]). The analysis is performed on the test set.

Figure 5.12 shows that the distribution of absolute SHAP values across WN's 524,288 features is relatively uniform. Specifically, the absolute SHAP values are within 2.37e-09 and 1.70e-04, with an average of 1.79e-05 (average normalized per number of features: 9.38), and a standard deviation of 1.4e-05. This indicates that all the features have a similar importance for the classification, which hints at an ability of the WN's convolutional layers to extract useful information and effectively build a robust classifier.

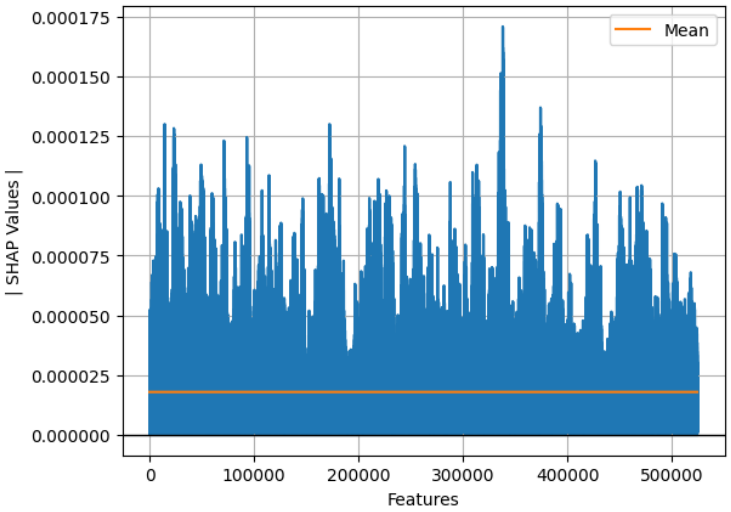


Figure 5.12: Absolute SHAP scores for each feature in WaveNet

On the other hand, the same analysis on the MR's 9,996 features shows a clear trend (Figure 5.13). Here, the absolute SHAP values vary between 0 and 1.34e-02, with an average of 1.51e-03 (average normalized per number of features: 15.09) and standard deviation of 9.54e-04. This suggests that some features are very important for the classification, while others are much less relevant. We argue that this is due to the almost deterministic nature of the MR's kernels, while a trainable system could learn a larger number of more effective features, which, in this case, is needed, as we can see looking at the WN's higher performance.

In Figure 5.13, features are in groups of increasing dilation size. Figure 5.14 plots the average absolute SHAP value against dilation size, which makes the trend apparent, thus suggesting the importance of features computed on a large perception field.

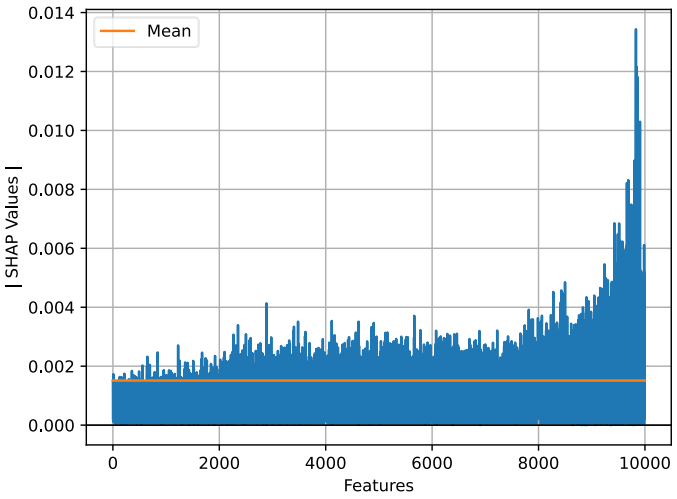


Figure 5.13: Absolute SHAP scores for each feature in MiniRocket.

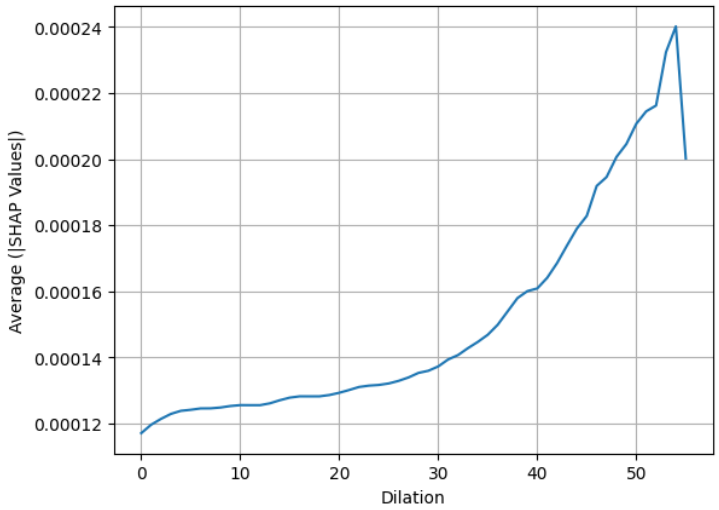


Figure 5.14: Average of the Absolute SHAP scores for dilation size in MiniRocket.

The WN deep convolutional stack can extract potentially all its features by exploiting a large perception field, which could be a motivation for its superior performance. Moreover, these findings highlight the importance of feeding the model with the whole time-series. In fact, in MR the highest SHAP values are measured for features extracted by kernels with high dilations, whose perception field spans over the whole input sequence (or at least large parts of it). While state-of-the-art approaches split each sensor's trace (e.g., [136], [146], [147]), we directly employ the entire sequence, which could contribute to explain the better accuracy achieved by both our methods.

Figure 5.15 (a) and (b) show the average importance (absolute SHAP value) for each class of the 12 most important features, for WN and MR, respectively. In WN, importance appears more evenly distributed across the features. The relative importance for each class (within each feature) looks more evenly distributed in WN, as well. Plotting normalized values (Figure 5.15 (c) and (d)) strengthens the impression.

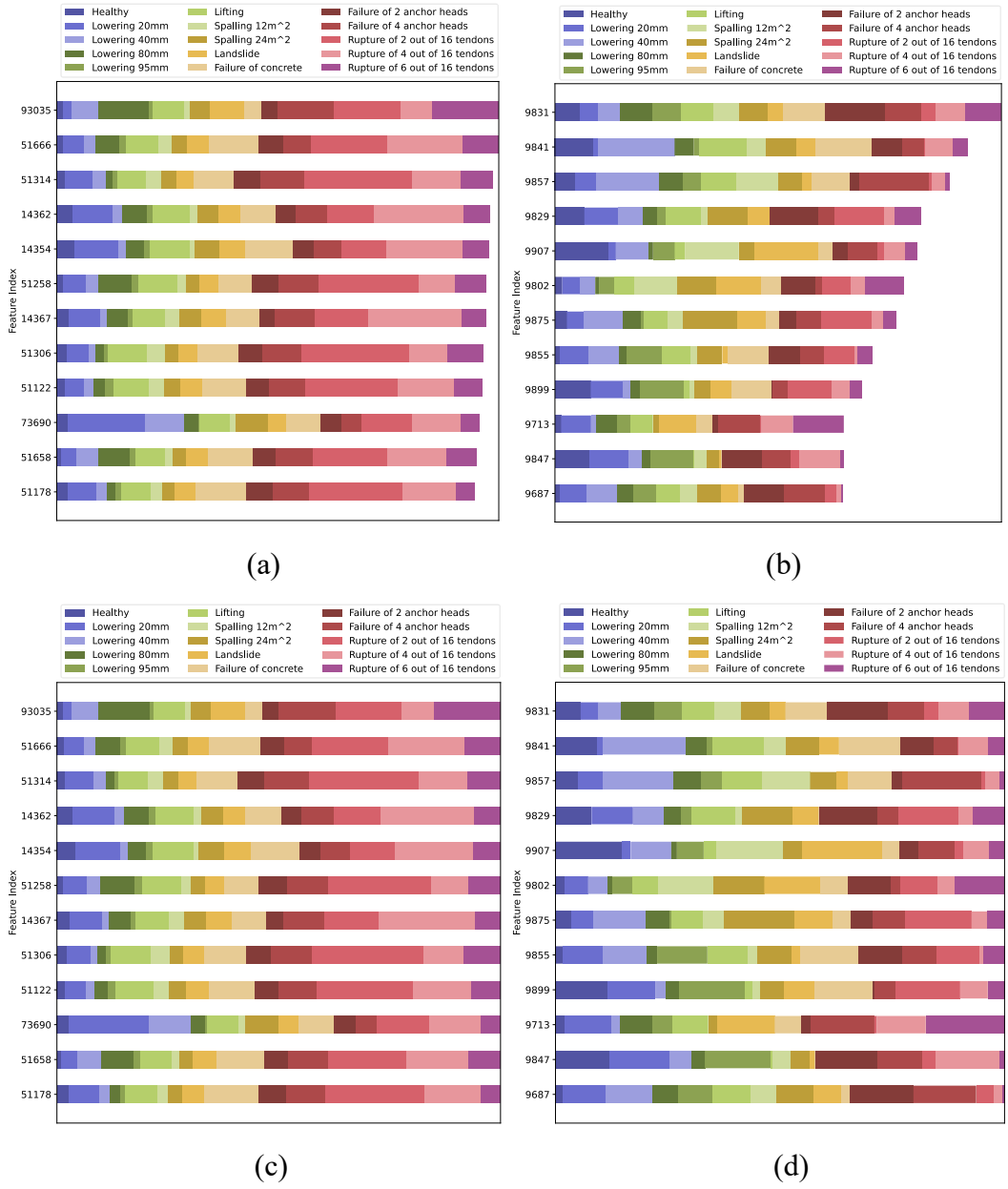


Figure 5.15: Absolute SHAP scores for each class of the 12 most important features in WaveNet (a) and MiniRocket (b). Values in (c) and (d) are normalized to better show the distribution of importance across classes.

Quantitatively, the mean across classes of the standard deviation among features is lower in WN than in MR 12 top features (2.1% vs 4.2%). Considering all features, the difference remains (3.3% vs 4.9%). The mean across all the features of the standard deviation among classes is lower in WN than in MR (4.3% vs 5.0%), but not for the top-12 case (5.0% vs 4.4%).

All these findings suggest that trainable convolutional layers learn features that

can influence decisions evenly across classes, which, according to the presented results, seems to lead to a more robust model, with better performance than MR's almost deterministic features. However, more work should be done to better understand the point.

The experimental results on the Z24 Bridge dataset show that end-to-end time series models can achieve state-of-the-art performance for vibration-based structural health monitoring without relying on handcrafted features or signal preprocessing. WaveNet reached 99.65% accuracy on the 5-class benchmark and 98.97% on the full 15-class problem, outperforming previous approaches while reducing model size from more than 98 MB to 31 MB. MiniRocket achieved competitive accuracy with extremely small model sizes (below 2 MB) and very short training times, at the cost of higher computational complexity during inference.

Deployment on Raspberry Pi 4 and Jetson AGX Xavier confirmed that full-sequence SHM classification is feasible on field-deployable edge devices. In particular, WaveNet showed a favorable balance between accuracy and energy consumption, reaching inference energies as low as 0.04 J on Jetson, while MiniRocket exhibited higher latency and energy usage due to its large number of operations. Overall, these results demonstrate that carefully designed time-series models can meet both accuracy and deployability requirements in practical SHM scenarios.

To support reproducibility and facilitate further research, we publicly release the implementation and the pre-trained models at: https://github.com/Elios-Lab/WaveNet_MiniRocket_Z24_Bridge_Structural_Health_Monitoring .

Building on these results, the next section moves toward stricter efficiency requirements by investigating Binary Neural Networks, targeting ultra-compact models suitable for low-power embedded deployment.

5.2 Binary Neural Networks for Embedded SHM

Building on the results obtained with full-precision WaveNet and MiniRocket models, an additional research direction concerns the further reduction of model size, memory footprint, and computational cost to enable deployment on even more constrained embedded devices. While the previous study demonstrated the feasibility of running multi-class SHM classifiers on platforms such as Raspberry Pi and Jetson Xavier, the growing interest in ultra-low-power edge computing motivates the exploration of more aggressive forms of model compression.

Binary Neural Networks (BNNs) represent one of the most extreme and promising approaches in this regard [51], [52], [153]. In BNNs, both weights and activations are quantized to a single bit, enabling inference through bitwise operations rather than floating-point arithmetic. This results in drastic reductions in model size, memory bandwidth, and energy consumption, making BNNs particularly attractive for systems that must operate continuously on battery-powered or highly resource-constrained hardware.

In this section, we investigate whether BNNs can effectively classify vibration-based SHM data while preserving sufficient discriminative capability for practical use. We first summarize the principles of binary neural networks and their associated training challenges, then describe the adopted architecture, training strategy, and evaluation methodology. Finally, we present experimental results that highlight the trade-offs between accuracy and efficiency when applying BNNs to the same Z24 Bridge classification task.

5.2.1 Binary Neural Networks

Binary Neural Networks (BNNs) approximate both weights and activations using binary values, significantly reducing memory usage and computational cost compared to full-precision neural networks [153]. BNN training follows the

Parts of this chapter has been published in A. Dabbous, L. Lazzaroni, F. Sakr, R. Berta, M. Fresta and F. Bellotti, "Binary Neural Networks for Real-Time Structural Health Monitoring on Edge Devices," 2025 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2025, pp. 1-4, doi: 10.1109/ICCE63647.2025.10930047.
The material has been adapted and extended to fit the structure and scope of this thesis.

standard forward and backward propagation paradigm, but introduces a binarization step for activations A and weights W during the forward pass. This binarization is commonly implemented using the sign function:

$$Sign(x) = \{+1, \quad if\ x \geq 0; \quad -1, \quad otherwise\}$$

By constraining A and W to the set {−1, +1}, inference can be implemented using bitwise XNOR operations followed by population count, replacing computationally expensive floating-point multiplications.

During backpropagation, standard gradient descent cannot be directly applied because the derivative of the sign function is zero almost everywhere. To address this issue, the Straight Through Estimator (STE) [154] is commonly adopted. The STE approximates the gradient of the sign function by treating it as an identity function during the backward pass, allowing gradients to propagate through the binarization operation. In addition, a clipping function is applied to the underlying full-precision weights to limit their update range:

$$clip(x,-1,1) = max(-1, min(1,x))$$

The main limitation of BNNs lies in their reduced representational capacity, which typically results in lower accuracy compared to full-precision models on complex tasks [155]. This loss of performance is primarily due to the severe information reduction introduced by binary quantization. To mitigate this effect, several optimization strategies and architectural refinements have been proposed in recent literature, achieving promising results and motivating the application of BNNs to challenging domains such as vibration-based structural health monitoring.

5.2.2 BNN Architecture and Training

As in Section 5.1, the experiments in this section are conducted on the Z24 Bridge dataset [134], [135], focusing on the same 15-class vibration-based damage classification task and adopting an end-to-end learning approach without intermediate preprocessing.

The objective here is to evaluate whether Binary Neural Networks can preserve

sufficient classification performance while drastically reducing model size and computational cost.

The architecture of the BNN model is illustrated in Figure 5.15.

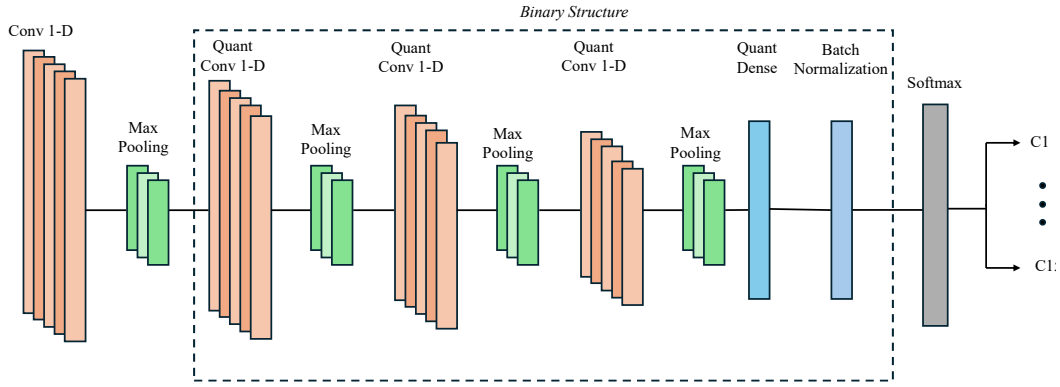


Figure 5.16: BNN architecture.

The network begins with an 1D convolutional layer with 32 kernels, which is kept in full precision (float32) to ensure precise feature extraction. Three convolutional layers follow, each one with a kernel size of 3 and containing 16, 16, and 8 kernels, respectively. Each convolutional layer is followed by a max-pooling layer, which reduces the spatial dimensionality of the output. After the convolutional and pooling layers, the model includes a binarized fully connected layer with 8 neurons. Batch normalization is applied after the fully connected layer to enhance convergence. The final layer contains 15 neurons, corresponding to the 15 scenarios in the classification task. The final layer uses a softmax function to output class probabilities. All layers apart from the first one are fully binarized (weights and activations), leveraging the computational efficiency and reduced memory usage of binary operations. This design balances the need for accurate feature extraction with the benefits of binarized neural networks.

We used the Larq framework [156], an open-source Python library for training neural networks with low-precision weights and activations (BNNs), to model and train our BNN on tensorial time series bridge scenario data. The network is trained for 300 epochs with 5-fold cross-validation and a batch size of 32. An Adam optimizer with a learning rate of 0.0001 is employed, using "ste_sign" for input and kernel quantization and "weight_clip" for kernel constraint.

5.2.3 Experimental Results

On the Z24 Bridge 15-class problem, the best results reported in the literature are those by Le-Xuan et al. [157] and the full-precision models presented in this thesis in Section 5.1.4.

We trained the proposed BNN model on a laptop equipped with an NVIDIA RTX 4070 GPU. The results, obtained on the test set, highlight the strong performance of the proposed approach under severe model compression constraints.

Table 5.6 summarizes the comparison between the proposed BNN model, the state-of-the-art solution by Le-Xuan et al. [157], and the full-precision architectures discussed in Section 5.1.4.

Table 5.6: Models performance comparison

Model	**Accuracy (%)**	**Model size (MB)**	**Energy consumption (mJ)**	**Inference time (ms)**
Our BNN	98.04	1.04	25	46
1DCNN-LSTM-ResNet [157]	81.50	--	--	--
Our WaveNet full-precision	98.97	92.50	374	575

The proposed BNN classifier achieves an accuracy of 98.04%, significantly outperforming the results reported by Le-Xuan et al. [157], while approaching the performance of the full-precision WaveNet model. In addition, while Le-Xuan et al. [157] reported only the number of parameters, the proposed BNN reduces the parameter count by approximately 98.4%, from 5,117,808 to 265,000. The resulting model size further demonstrates the efficiency gains achieved through binarization when compared to full-precision architectures.

After training, we converted the model to TensorFlow Lite format for deployment on edge devices. We utilized the Larq Compute Engine (LCE) library to perform this conversion. The resulting model size is 1.04 MB, making it suitable for

deployment on microcontroller units (MCUs) and other embedded systems with very limited resource availability. However, we encountered issues with implementing the model on an MCU due to its RAM usage, which is affected by the input size of (65,536, 1). To address this, we experimentally assessed our proposed BNN model on a Raspberry Pi 4 edge device. On such platform, we measured inference time, power consumption, and energy consumption for the proposed model. To track power variations, we used an USB multimeter connected between the power supply and the device. Energy consumption was then calculated by multiplying the power usage by the inference time for a single sample. The reported values are averages over 100 runs.

The results underscore the BNN model's relatively low energy consumption on the Raspberry Pi 4, measured at just 25.30 mJ. In terms of inference time, the proposed model classifies a single sample in only 46 ms, making it highly suitable for real-time SHM. Our findings indicate that the BNN model surpasses the state-of-the-art in both inference time and energy consumption. These results highlight the BNN's effectiveness and efficiency as an edge model for real-time applications, requiring significantly fewer resources compared to other deep learning solutions, with a small drop in accuracy compared to the state of the art.
Overall, these results demonstrate that Binary Neural Networks can effectively extend vibration-based SHM toward ultra-low-power embedded deployment. While a modest reduction in classification accuracy is observed compared to full-precision models, the proposed BNN achieves a drastic reduction in model size, inference latency, and energy consumption, enabling real-time operation on low-cost, field-deployable edge devices. This complements the full-precision solutions presented earlier in the chapter and highlights the trade-off between accuracy and efficiency when pushing SHM inference closer to the sensing layer.

CHAPTER 6

6. Driver Distraction Detection

Driver distraction is a major contributing factor to road accidents [158], [159] and represents one of the most critical challenges in Advanced Driver Assistance Systems (ADAS) and automated driving [160]. While modern vehicles increasingly rely on perception and planning modules to interpret the external environment, the driver's cognitive and attentional state remains a decisive element for safety, particularly in semi-automated or shared-control scenarios [161]. Reliable detection of distraction in real time requires models capable of interpreting heterogeneous, high-frequency signals under the strict latency, robustness, and resource constraints imposed by in-vehicle embedded platforms [162], [163].

In line with the theme of this thesis, this chapter investigates driver distraction detection as a time-series-driven, safety-critical application where lightweight deep learning models and deployable system architectures are essential [164], [165]. The problem shares several characteristics with the previous case studies, including noisy and non-stationary signals, ambiguous or imbalanced annotations, and the need for real-time inference. At the same time, it introduces additional challenges that are particularly relevant in human-centered systems, such as strong inter-subject variability, multimodal data fusion, and the difficulty of defining reliable ground truth for cognitive states.

The experiments presented in this chapter are conducted within the end-to-end data management and ML workflow introduced earlier in the thesis. Measurify is employed as the backbone for storing, organizing, and retrieving time-stamped multimodal time-series data. To map the collected data to the Measurify resource model, the Thing was defined as "UserN", where N identifies the subject

associated with the recording session, the Device was set to “Smart-sensors”, and the Feature was defined as “Sensors-data”, corresponding to the multimodal signals acquired from the different sensing sources during the driving experiment. The Model Registry introduced in Chapter 3 supports systematic tracking of datasets, model configurations, and trained artifacts. This infrastructure enables reproducible experimentation across multiple learning paradigms and deployment scenarios.

The chapter is structured around two complementary studies. The first addresses a binary classification problem, distinguishing attentive from cognitively distracted driving, and focuses on real-time detection under tight latency constraints. The second extends the problem to a multi-level setting, discriminating among attentive driving, distraction, and inattention, reflecting more realistic driver monitoring requirements. Across both studies, we analyze the impact of model choice, window length, data modalities, and evaluation protocols, with particular attention to cross-user generalization and on-board deployability.

Through these investigations, the chapter aims to assess how time-series-dedicated deep learning models can support robust, real-time driver state monitoring, and to highlight the trade-offs between accuracy, generalization, and resource efficiency that must be considered when transitioning from laboratory studies to real-world automotive deployment.

6.1 Binary Cognitive Distraction Detection

This section describes the design of the user experiments and the procedure adopted for data collection, also including the driving simulation settings. The study presented in this paper was part of a broader study on level 3 Automated Driving within the Hi-Drive project [166] (in particular, how the Human-Machine Interaction of AD, integrated with the driver's state, can improve a safe transition from automated to manual driving). It is important to precise that this data collection was done preliminarily, thus avoiding any bias from the subsequent ML processing.

Parts of this chapter has been published in M. Fresta et al., "Deep Learning-Based Real-Time Driver Cognitive Distraction Detection," in IEEE Access, vol. 13, pp. 26589-26607, 2025, doi: 10.1109/ACCESS.2025.3539392.

The material has been adapted and extended to fit the structure and scope of this thesis.

6.1.1 Experiment

6.1.1.1 Participants

Our sample was quite various and balanced, consisting of 42 participants (23 females, 19 males) of age between 22 and 62 years (mean age of 35.64 ± 9.78 years). Participants had on average their driving license for 15.80 ± 10.29 years and drove for about 11,404.76 ± 8,705.37 kilometers a year. The study was implemented in accordance with the principles of the Declaration of Helsinki and the ethical laws applied in France at the time of data collection. All participants signed a written informed consent form and received 40 euros for their participation.

The number of 42 users has been chosen based on the results of a statistical power analysis done with G*Power 3.1.9.7 [167]. This choice overcomes the state-of-the-art, as the number of participants in similar works is 9, 24, 26, 40, 41 in [168], [169], [170], [171], and [172], respectively. We argue that the higher number of subjects (68) chosen by Gjoreski et al. [173] is due to the broader extent of their study, which, beside cognitive distraction, also analyzed emotional, sensorimotor and mixed distractions.

6.1.1.2 Apparatus

The medium-fidelity static driving simulator set up at the VEDECOM labs consists of three screens of 140×230 cm allowing 230° of vision and includes a two-seat cabin, a steering wheel, a set of pedals, three mirrors (i.e., left, rear, and right), and a dashboard. The driving simulator ran under SCANeR studio 2023.1 (Avsimulation®), which was synchronized with a 5-camera eye-tracking setup (Smart Eye Pro 6.0). Eye-tracker camera positions are shown in Figure 6.1. The driving simulator and camera signals were sampled at 60 Hz. A BioPac Bionomadix setup with a 3-electrode electrocardiogram (ECG) and Electrodermal activity (EDA) was used to record and sample physiological signals at 2,000 Hz (Biopac MP150 with AcqKnowledge 5).

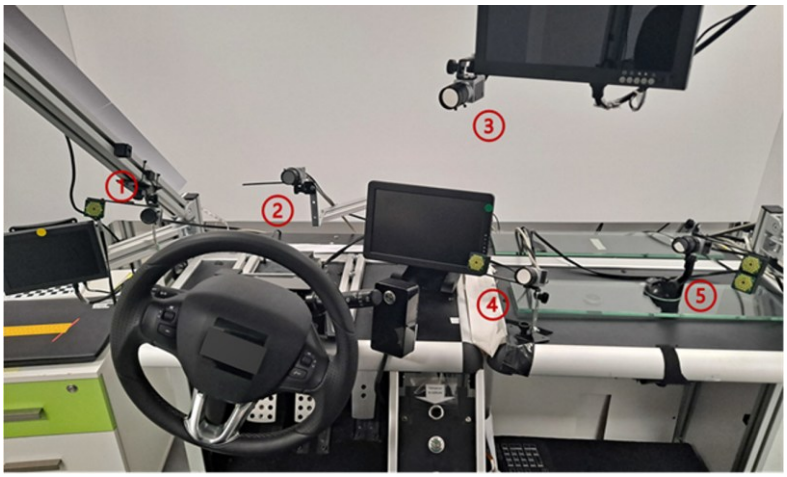


Figure 6.1: In the 5-camera Smarteye Setup, each number indicates the placement of a camera.

6.1.1.3 Cognitive Distraction Task

The cognitive load was induced by the Twenty Question Task (TQT) [174], which aims at emulating a conversation with a passenger or a phone call. This task consists of guessing something within a category, in our case an animal or a public figure, by asking a maximum of twenty questions that can only have responses that are “yes” or “no”. This task is known to use cognitive resources such as working memory, problem-solving, and visualization, which is very close to a phone conversation. This task was preferred over the n-back task, even if the n-back task is commonly used in driving studies (for a meta-analysis on the n-back task, see von Janczewski et al. [175]) because the TQT presented more advantages due to its ecological aspect, better mimicking a real-world distractive activity.

6.1.1.4 Procedure

Prior to using the driving simulator, participants received information regarding the goals of the study and completed the informed consent form.

The experimenter applied the ECG and EDA electrodes on the participant, verifying that a good signal quality was obtained from the electrode’s placement. The ECG electrodes were placed according to ECG placement standards following Einthoven’s triangle on the chest with one electrode near V1, one near

V5, and earthing below floating ribs, which was found to be the best placement for the QRS complex with a small number of electrodes [176]. The EDA electrodes were placed on the left wrist due to the driving constraints.

At the beginning of the experiment, participants completed a short evaluation of their trust in automated driving by means of a slider ranging from 0 (lowest trust) to 100 (highest trust) and a questionnaire of acceptability [177]. At the end of the experiment, participants completed a technophilia questionnaire as well [178].

Participants had to drive on a 2-lane highway with ongoing traffic, being instructed to stay in the right lane (Figure 6.2).

Figure 6.2: The driving simulator during the highway drive.

During the study, the experimenter ensured that participants followed instructions to be looking at the road. Additionally, the surrounding environment was intentionally kept quite simple, to minimize any potential distractions.

Participants started with a 10-minute training session, in which they could familiarize themselves with the driving simulator. After the end of the training, participants seamlessly continued driving the car for two consecutive 90-second periods, which constitutes our data sample. After the first period with no cognitive distraction, the TQT cognitive distraction task started.

This design was very important to prevent potential bias from alternating multiple distraction and non-distraction periods. We deemed that the cleanest approach was to start without inducing any cognitive distraction for a not too long period, so to prevent the risk of mind-wandering, boredom, or drowsiness. Also, the second period, during which participants were engaged in a task promoting cognitive

distraction, had a duration that prevented the induction of stress, which could have impacted the collected data. The equal time length of the two experimental conditions also allowed the creation of a balanced dataset. Overall, we aimed at minimizing the number of possible factors, thus resulting in a “simple” set-up, so to better control the experiment and allow a cleaner analysis.

6.1.1.5 Data Structure

For the sake of the study, we extracted the following data from the driving simulator: Speed (m/s), Steering wheel speed (rad/s), Steering wheel angle (rad), Steering Wheel Torque (Nm), Gas pedal activation (%), Brake pedal activation (N). All these signals were sampled at 60Hz. For the eye-tracking data, real-time and estimated eye-gaze position, head rotation, pupil diameter and the eyelid opening were collected, sampled at 60Hz. For the physiological data, we used the cardiac-rhythm (mV) and EDA response (S), sampled at 2,000Hz.

6.1.1.6 Signal Post-Processing

Raw data from the driving simulator and the ECG were collected with no issues nor need for post-processing. The collection of EDA data was more complicated. A significant number of participants lost the electrodes placed on their wrists due to the movement of the steering wheel and the temperature (about 26-28 °C) in the driving simulator. As an example, Figure 6.3 shows a divergence in the EDA and heart rate (HR) values towards the end of the series.

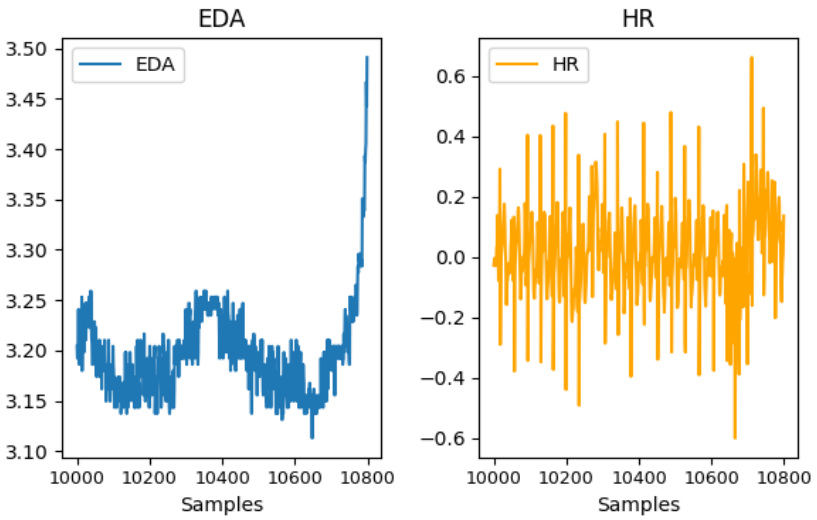


Figure 6.3: Example of EDA and HR signal divergence.

This can be seen as a limit of our experimental setup. However, it also reflects an objective invasiveness, also in terms of privacy, of physiological data collection. Moreover, it allows to test the distraction detection system in sensor failure conditions as well. Thus, we argue that it represents a realistic challenge towards real-world deployment. In this perspective, we chose to keep the raw input with no signal post-processing. On the other hand, we applied post-processing to the eye-tracking signal because of a significant data loss (21.28% ± 5,56%) due to the system, not the user interaction. A linear interpolation was performed on missing data and applied on any missing segment of no more than 150ms, above which data was considered lost. This threshold was chosen according to the literature to be able to interpolate during a complete saccade fixation and saccade process [179]. Following interpolation, eye-tracking data was smoothed with a small low-pass Butterworth filter. Such post-processing allowed us to reach a 92.90% ± 5.45% quota of usable data. BioPac physiological signals, collected at 2,000 Hz, were downsampled to match the frequency of vehicular and eye-tracking measurements (60 Hz), which is close to that of commercial fitness monitors, representing a more realistic approach for on-board deployment.

6.1.1.7 Dataset

The dataset includes 12 features for 162,800 samples, each one labeled as a distracted or not-distracted state (in a 50%-50% proportion, as per the data collection procedure). Featured data types are reported in Table 6.1, where X and Y are used for features representing spatial coordinates.

Table 6.1: Data types

Data type	**Features**
Vehicular driving data (D)	Speed X
	Speed Y
	Acceleration X
	Acceleration Y
	Steering angle
	Steering speed
	Steering torque
	Accelerator
	Head heading
	Head pitch
	Head roll
Eye-tracker data (E)	Eyelid Opening
	Left Eyelid Opening
	Right Eyelid Opening
	Robust Eyelid Opening
	Left Robust Eyelid Opening
	Right Robust Eyelid Opening
	Pupil Diameter
	Left Pupil Diameter
	Right Pupil Diameter
	Estimated Eye Position X
	Estimated Eye Position Y
	Estimated Eye Position Z
	Eye position X
	Eye position Y
Physiological data (P)	Electrodermal activity (EDA)
	Heart rate (HR)

As for eye-tracking data, eye coordinates positions X and Y were processed to create a new categorical feature that identifies the object on which the eye's gaze focused. The identified categories include left side, left mirror, road, dashboard, steering wheel, rear mirror, right mirror, right side, outside, and eye-sight not recognized. Categorical values were then processed with OneHotEncoding [180], so the single eye position identification feature was shaped into a sparse matrix

consisting of 10 new features.

In this work, we adopted an end-to-end ML approach, considering as sample to be classified a whole multivariate time-series within a fixed-length time window (e.g., 1 second), thus using the raw signals directly, as commonly done in state-of-the-art deep learning applications. This choice was motivated by two main reasons. First, we wanted the model itself to learn and optimize the most relevant discriminative features directly from the temporal data, rather than relying on handcrafted statistical descriptors. Second, from a deployability perspective, avoiding an explicit feature-extraction stage reduces the amount of preprocessing required at runtime, and therefore limits additional computational cost, latency, and energy consumption on the target embedded platform. Other state-of-the-art studies (e.g., Misra et al. [171]) instead represent each sample as a set of data points enriched with statistical summaries computed over the interval.

For targeting a real-world vehicular application, we applied an overlapping moving windowing technique to the training and validation data, initially employing a stride of 10% of the total data points for the given window size. For simplicity, and given the very similar results obtained, testing windows were finally separated without any overlap, thereby generating multiple independent test samples.

As anticipated in the Related work section, we partitioned the dataset into three distinct subsets, selecting 70% of the users for training (30), 15% for validation (6), and 15% for testing (6). Other state-of-the-art studies (Gjoreski et al. [173], Misra et al. [171]) have not adopted this solution. But, in view of a real-world system deployment, having the same users in the training, validation and testing sets may lead to an over-optimistic assessment, since it would assume that the DDD system would have been trained also on data from its actual users, before its deployment (or after-sale fine-tuning would be needed). Thus, we decided for a between-subject design, to ensure integrity and generalizability of the model across different subjects.

6.1.2 Model Tuning

For our analysis, we optimized and tested various state-of-the-art ML models dedicated to timeseries classification to assess their performance in the task. This section briefly introduces them and describes their hyperparameter tuning.

6.1.2.1 1D-CNN

1D-CNN [181], [182] are well-suited for timeseries analysis as they are able to capture temporal features through local connectivity and shared weights, which makes them robust to shifts and suited to handle multivariate data. 1D convolution is performed by 2D kernels that aggregate values on the temporal and on the channel axes. For simplicity, Figure 6.4 shows an example with a 1D signal (i.e., single input channel).

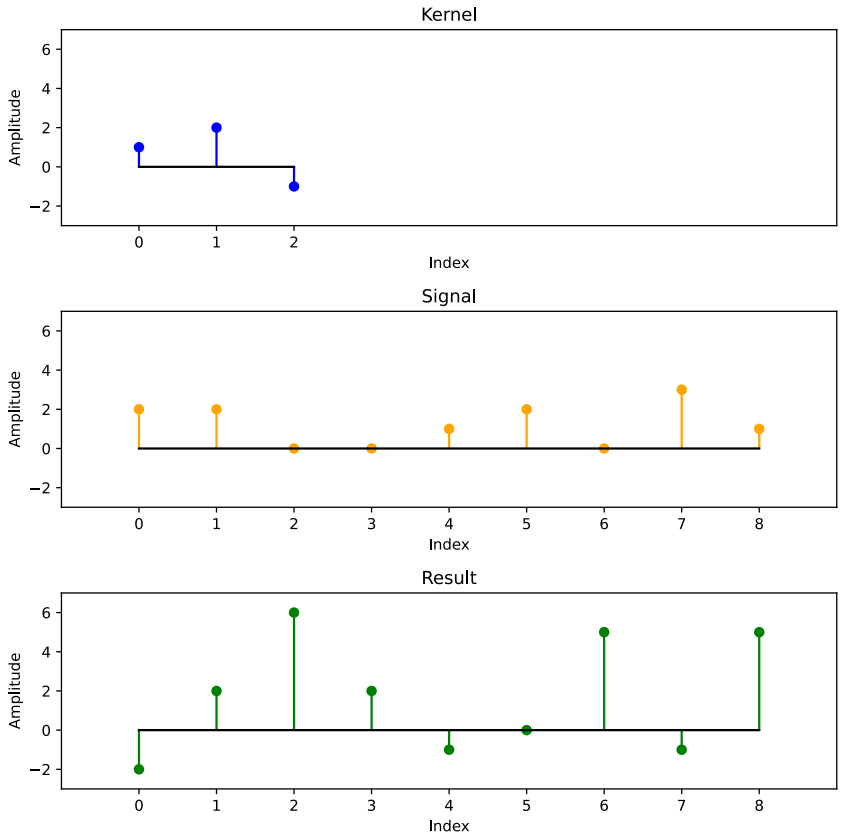


Figure 6.4: 1D convolution with a 1D input signal.

To optimize the network, we performed a grid search exploring various kernel sizes (3, 5, 7), batch sizes (16, 32, 64, 128, 256), learning rates (10-4, 10-3), dropout rates (0.3, 0.5, 0.7), and number of kernels in the two convolutional layers (16-32, 24-48, 24-48-96, 64-128). The combination that resulted in the best accuracy was a kernel size of 3, batch size of 32, learning rate of 10-3, dropout rate of 0.3, and 16-32 kernels (Figure 6.5).

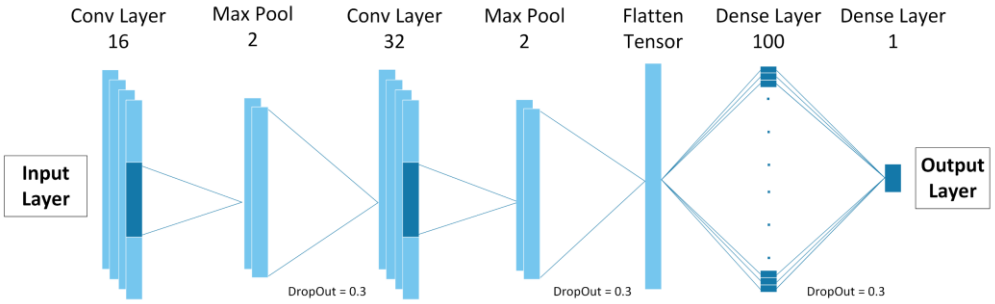


Figure 6.5: 1D-CNN model architecture.

Our network uses the ReLU activation function, valid padding, and has a two-layer dense head, with the final sigmoid activation for extracting the distraction class probability.

6.1.2.2 MiniRocket

MiniRocket [183] [184] was already introduced in Section 5.1.3.2, where its main methodological principles were discussed in detail. In the present chapter, it is reused as an efficient time-series classification baseline for the driver distraction detection task. Figure 6.6 recalls the computation of the PPV-based features, while Figure 6.7 shows the adopted MiniRocket pipeline used in this application.

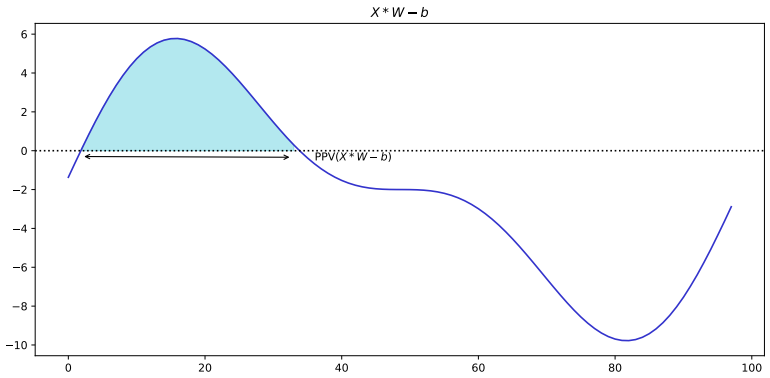


Figure 6.6: Example of PPV computation for a feature vector.

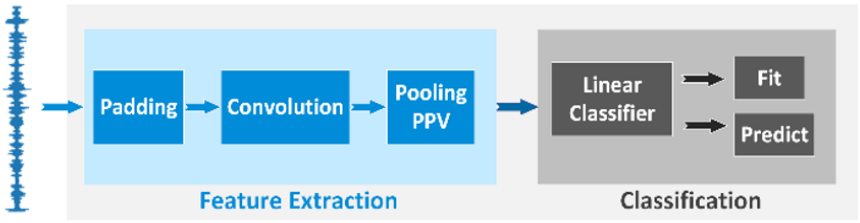


Figure 6.7: MiniRocket model architecture.

This approach minimizes hyperparameter tuning, which allows limiting the grid search to the number of kernels (we tested: 420, 504, 1008, 2688) and max number of dilations per kernel (5, 6, 12, 32). We obtained the best validation results for 420 kernels and 5 max dilations. MiniRocket involves just one convolutional layer (but with a large number of combinations of kernels and dilation, i.e., extracted feature vectors). However, we consider it a deep learning model, as it combines the convolutional layer with a shallow ML model such as the Ridge classifier.

6.1.2.3 WaveNet

WaveNet [185] was already introduced in Section 5.1.3.1, where its main architectural principles were discussed in detail. In the present chapter, the model is reused as a representative deep learning architecture for time-series classification in the driver distraction detection task. Figure 6.8 recalls the adopted WaveNet structure, while Figure 6.9 illustrates the stack of dilated convolutional layers that enables the model to capture temporal dependencies over progressively wider receptive fields.

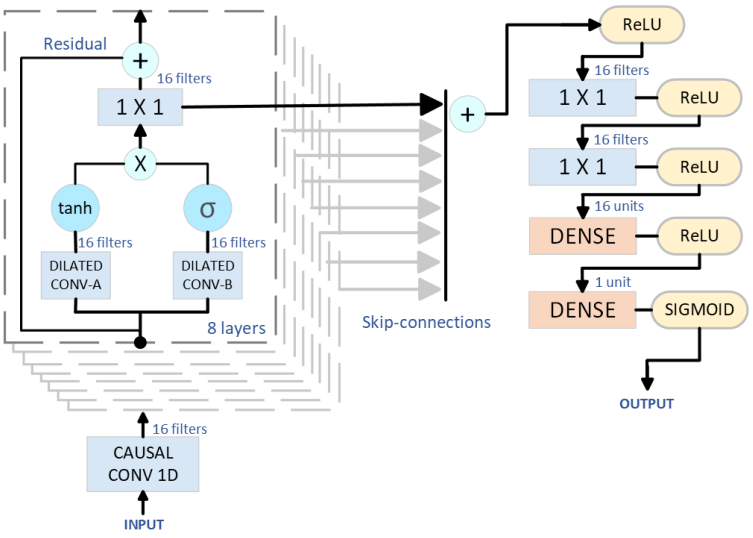


Figure 6.8: WaveNet model architecture.

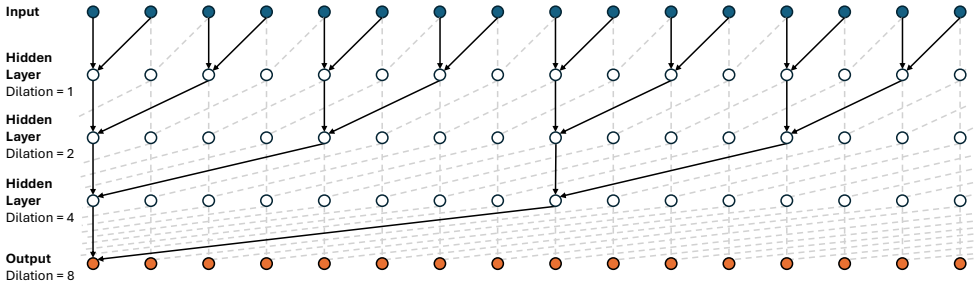


Figure 6.9: Stack of dilated convolutional layers.

Our grid search involved learning rates (10-4, 10-3), number of filters (both in blocks and head) (16, 32, 64), number of units in the head's dense layer (16, 32, 64), number of stacked layers (7, 8, 9), and batch size (32, 64, 128), finding the best values for learning rate 10-4, filters 16, units 16, layers 8, and batch size 32 (Figure 6.8).

6.1.2.4 Transformers

Transformers [186] have been used for timeseries classification, leveraging the well-known attention mechanism. We employed TensorFlow's implementation, where a transformer block with multiple attention heads enables the model to focus on different aspects of the data. Specifically, we utilized the MultiHeadAttention layer with 8 attention heads and embeddings' dimension of

128, followed by a feed-forward network (ffn) with 512 units. The model architecture consists of a decoder with 4 transformer blocks, each one containing multi-head attention (Figure 6.10), layer normalization, and a dropout mechanism for regularization.

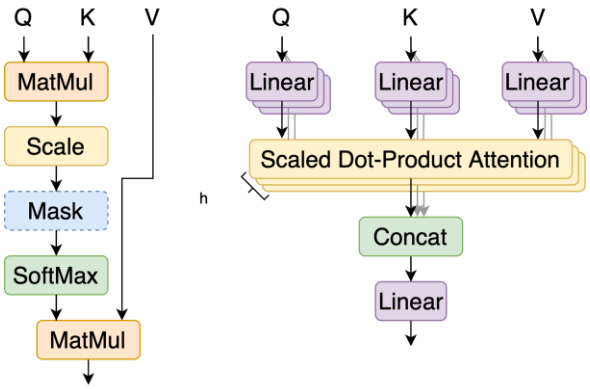


Figure 6.10: Scaled dot product attention (left) and multi-head attention (right).

Positional encoding was added to account for the sequence order in the timeseries data. Finally, after global average pooling, a dense layer with sigmoid activation is used for binary classification (Figure 6.11).

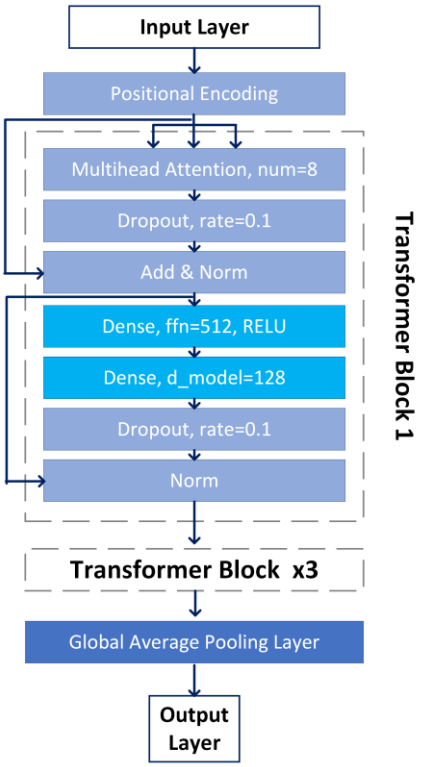


Figure 6.11: Transformer model architecture.

The parameters were computed with a grid search conducted on number of attention heads (4, 8, 16), embedding dimension size (64, 128, 256) and feedforward network size (128, 256, 512). The values for the best configuration are reported in Figure 6.10.

6.1.2.5 ResNet-18

Residual Networks (ResNets) [187] are popular in image recognition, thanks to their skip connections (Figure 6.12), which allow the building of deep hierarchical structures by mitigating the vanishing gradient problem. In adapting ResNet for timeseries data, we configured a 1D ResNet-18 version. Each residual block of the model consists of two 1D convolutional layers followed by batch normalization and ReLU activation (Figure 6.12), with an optional convolutional shortcut if the input and output dimensions differ.

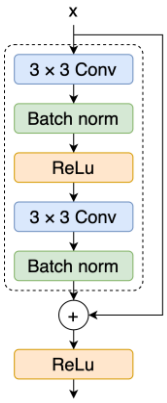


Figure 6.12: ResNet block.

The network in Figure 6.13 begins with an initial convolution, followed by four groups of residual blocks with increasing number of filters (8, 16, 32, and 64) and occasional downsampling through stride adjustments.

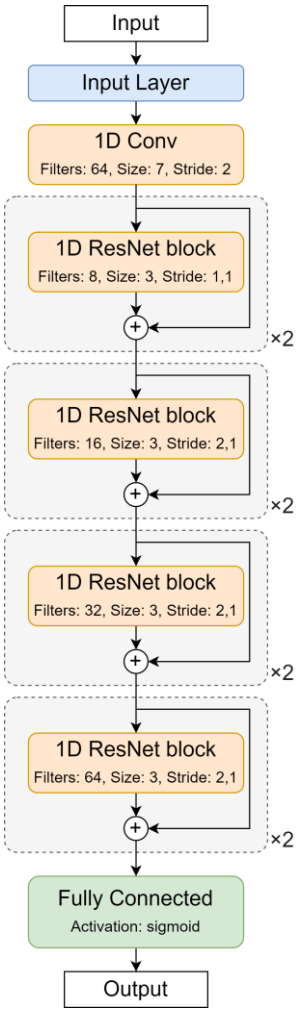


Figure 6.13: ResNet-18 model architecture.

After the final block, a global average pooling layer eliminates the temporal dimension, and a fully connected layer with a sigmoid activation outputs the binary classification. The model is compiled using the Adam optimizer and binary cross-entropy loss. Grid search involves kernel size (3, 5, 7), finding the best option of 3.

6.1.3 Experimental Results

This section presents and discusses our experimental results, which we compare with two key reference works in the area, namely Gjoreski et al. [173] and, particularly, Misra et al. [171]. We must preliminarily highlight that the three works (including ours) use three different proprietary datasets, which makes the comparison arbitrary. However, the goal of our work is to advance the state-of-

the-art by answering the research questions proposed in the Introduction, trying to assess different experimental methods and ML models, also leveraging the most relevant knowledge and results published in the literature, such as the two above-mentioned papers.

The experimental analysis is structured to progressively evaluate model performance under increasingly realistic constraints, including reduced time windows, cross-user generalization, limited sensing modalities, and on-board deployability.

6.1.3.1 20-Seconds Window Size

As a first experiment, we used the same time-window size as Gjoreski et al. [173], i.e., 20 seconds. Table 6.2 reports a comparison of their experimental results with ours, first on the same set of signals (i.e., eye-tracker (E) + physiological (P)) then adding also driving (or vehicular) data (D), which is quite straightforward for a system targeted at on-board deployment.

Table 6.2: F1 score for 20 second time-windows (D=Driving, E=Eye-Tracker, P=Physiological data)

Dataset	**Classifier**	**F1 score (%)**
Gjoreski et al. [16] (E+P)	RF	67
	GB	73
	XGB	72
	eLSTM	67
	STRNet	67
Ours (E+P)	RF	62
	XGB	69
	1D-CNN	82
	MiniRocket	71
	WaveNet	73
	Transformer	81
	ResNet-18	81
Ours (D+E+P)	RF	74
	XGB	79
	1D-CNN	85
	MiniRocket	**95**
	WaveNet	85
	Transformer	84
	ResNet-18	91

Considering shallow learning techniques, such as RF and eXtreme Gradient Boost (XGB), our RF performed worse than the same model in [173] (62% vs 67%), and XGB alike (69% vs 72%), which allows us to argue that our dataset is not intrinsically easier than that in [173]. On the other hand, our investigated deep learning models (i.e., 1D-CNN, MiniRocket, WaveNet, Transformer, and ResNet-18) perform substantially better than those in [173] , reaching the highest F1 score with the 1D-CNN (82%).

These results align with established literature (e.g., [188]), which highlights that shallow models work well with smaller datasets - particularly having a limited number of features, that are frequently selected through accurate manual feature engineering. DL models, on the other hand, can have a much larger number of learnable parameters, and are thus better suited to process high-complexity data, being able to perform and optimize feature extraction directly from raw signals, through end-to-end ML training. As we will see from the last group of results in Table 6.2, this also allows us to effectively process the vehicular signals, further boosting performance. Moreover, differently from the shallow learning models, all the proposed DL models are dedicated to timeseries (i.e., they intrinsically exploit the sequential nature of the input data) [189].

Considering our whole dataset (i.e., including vehicular driving signals beside eye-tracking and physiological data (D+E+P group in Table 6.2), MiniRocket outperforms all the other models, achieving an outstanding F1 score of 95%, followed by ResNet-18 (91%). This outcome confirms the known validity of the model (e.g., [190]) but is quite surprising given the fact that MiniRocket uses pre-determined convolutional weights, requiring very limited training time, which is essentially due to the linear classifier head. Accordingly, the MiniRocket's much lower performance on the reduced dataset (71% in E+P vs 95% in D+E+P) could be attributed to the deterministic, non-learnable nature of its convolutional kernels, which penalizes its performance in challenging datasets. This limit also raises concerns about the performance of this model in further and more various driving scenarios than those tested in the present experiment.

To a lesser extent than for MiniRocket, however performance of all our models

has a significant surge in the complete dataset, which strongly hints at the importance of driving data for tackling the task. Particularly, we argue that the significantly lower F1 score in the (E+P) dataset than in the whole dataset could be due to the larger differences between subjects for physiological and eye-tracker signals rather than for vehicular signals. In fact, as we will discuss later, our results concern a test set with different users than the training set, while [173] does not specify such differentiation.

6.1.3.2 Window Size Reduction

These results spurred us to investigate a reduction of the time window size, in view of an on-board deployable, real-time system, which needs much lower latencies. We thus consider samples of 20, 5, 2, 1, and 0.5 seconds, corresponding to 1,200, 300, 120, 60, and 30 data frames, respectively, within the 60 Hz dataset. Figure 6.14 shows the performance of our ML algorithms with the different window sizes.

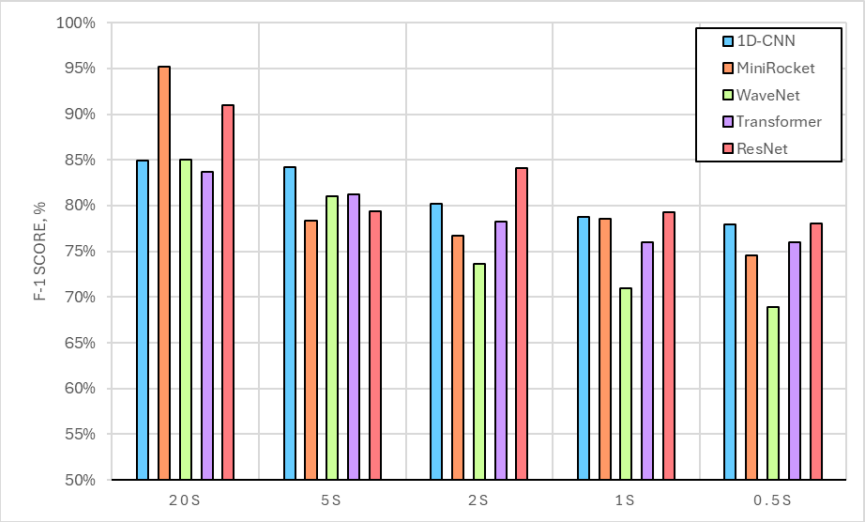


Figure 6.14: F1 scores for different window sizes.

Results indicate a decrease, which smooths down at around 1s. The chart also shows that ResNet-18 (and 1D-CNN alike, but with slightly worse results) has more consistent performance than the others across the different sizes, which may suggest its utilization also in a multi-resolution system. The performance loss is limited for the ResNet-18, which achieves a 78.04% F1 score with a 0.5-time window (not much lower than 79.32% obtained with 1s) and also for the 1D-CNN

(78.71% for 1s, vs 77.92% for 0.5s). This result is very significant, as it suggests the feasibility of a real-time system, processing a short time span at each shot.

6.1.3.3 Cross-User Generalization

A third experiment concerned the comparison with Misra et al. [171], who achieved a remarkable accuracy of almost 99% (on a D+E+P dataset). They use an RF model which samples the input signals every second and also incorporates four secondary features computed in the one-second window, namely: longitudinal velocity, standard deviation of the steering wheel angle, standard deviation of lateral position, and mean of the steering error (i.e., the deviation between the expected steering movement and the one performed by the driver). In contrast, our RF model achieves an accuracy of 78.15% and ResNet-18 of 83.00% (Table 6.3).

Table 6.3: Influence of generalization, signal type and timeseries-oriented neural models on accuracy for 1s windows

	D+P+E	**D+E**	**D+P**	**E+P**	**D**	**E**	**P**
RF in Misra et al. [171] (without generalization)	98.75%	95.08%	98.13%	99.37%	71.57%	96.32%	99.58%
RF with generalization	78.15%	76.16%	69.92%	69.40%	71.42%	71.05%	66.37%
ResNet-18 with generalization	83.00%	80.68%	77.90%	76.05%	78.60%	78.19%	69.01%
RF without generalization	97.84%	94.70%	93.65%	93.47%	94.98%	94.22%	86.46%
ResNet-18 without generalization	99.45%	99.06%	97.85%	97.09%	97.67%	97.28%	92.09%

We argue that such a performance gap can be attributed to the different dataset split approaches employed. While Misra et al. [171] trained and tested their models on a mixed dataset (i.e., the training, validation, and testing subsets were split after random shuffling the samples from all the users), we opted for a stricter approach by splitting the dataset between subjects, reserving 30 users for training, 6 other users for validation and 6 others for testing. This is because we aim to

evaluate the generalization capabilities of the models across new, unseen subjects [173], [191].

In order to give a novel, quantitative measure of the overfitting on the dataset subjects, Table 6.3 also report the results we obtained in an approach without generalization, (accuracy of 97.84% and 99.45%, for RF and ResNet-18, respectively), that are comparable with those in Misra et al. [171]. This approach implies training a system with data also from its final users, which may lead to an over-optimistic performance assessment in view of a real-world deployment.

Extending the comparison to the different data types, Table 6.3 shows that the performance of our models always benefits from the presence of D and E data, which is not the case in [171], where the best result is obtained by processing physiological data only. The between and within-subject design comparison clearly shows that P signals alone are much more prone to inter-individual variability. RF and ResNet-18 achieve similar results on D and on E separately, both with and without generalization. Combining D and E generally increases the models' performance, while adding P to D or E causes a small drop in accuracy, which corresponds to the problematic collection of P data, as reported in sub-section III.F (Signal post-processing).

The introduction of secondary features (as in [171]) leverages domain knowledge to enhance the input, while the deep-learning approach we follow synthetizes features by directly processing the raw input signals as part of the model training process. Full timeseries processing preserves the temporal dependencies and exploits the sequential nature of timeseries data, allowing models to learn complex patterns and interactions over time. This approach requires more sophisticated, deep learning models, such as 1D-CNN, ResNet-18 or MiniRocket, which are less likely to miss important temporal information. The ability to recognize complex patterns and still avoid overfitting appears from the superior performance of ResNet-18 with respect to RF in all the tasks with our dataset.

6.1.3.4 Signal Types

A key factor for on-board system deployment concerns the type of signals to be processed. Particularly, vehicular data are easily accessible to original equipment manufacturers (OEMs), while collecting the others is more expensive and invasive for the driver. On the other hand, vehicular signals alone may not be sufficient to reach the necessary accuracy. We thus analyzed several combinations of data types (D, E and P), considering different types of deep learning models as well. As time-window size, we chose 0.5 s. This reduces performance compared to Table 6.3 (1 s), but can be considered a stress test, halving the system latency, which may make the difference, especially at high vehicle's speed.

A comprehensive analysis, reported in Table 6.4, confirms the fundamental importance of vehicular signals.

Table 6.4: Results for Different Types of Input Data for 0.5 s window

Dataset	Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Training time (s)	Model size (KB)
D	1D-CNN	72.03	74.44	67.12	70.60	201	329
	MiniRocket	67.51	68.00	66.11	67.04	18	86
	WaveNet	69.77	70.23	69.77	69.99	85	793
	Transformer	70.69	74.50	65.72	69.84	119	44
	ResNet-18	72.35	67.79	75.62	71.49	219	1,100
D+E	1D-CNN	75.12	75.36	71.57	73.41	155	339
	MiniRocket	72.36	70.99	74.54	72.72	13	92
	WaveNet	71.95	72.01	71.89	71.95	89	833
	Transformer	74.68	72.40	71.30	71.84	120	60
	ResNet-18	75.40	75.23	75.05	75.14	218	1,159
D+P	1D-CNN	71.93	74.30	63.98	68.76	98	331
	MiniRocket	69.04	65.40	72.94	68.96	12	84
	WaveNet	69.03	69.03	70.75	69.88	90	997
	Transformer	71.20	72.32	69.34	70.79	94	49
	ResNet-18	70.75	62.28	76.27	68.57	240	1,130
E	1D-CNN	69.31	71.98	67.71	69.78	73	310
	MiniRocket	67.68	66.40	68.33	67.35	19	101
	WaveNet	69.53	70.13	69.84	69.99	87	803
	Transformer	71.12	73.28	69.23	71.20	129	47
	ResNet-18	71.99	63.00	78.77	70.01	239	950
E+P	1D-CNN	68.17	70.48	63.31	66.70	95	311
	MiniRocket	65.42	66.58	64.52	65.53	12	43
	WaveNet	66.93	67.06	66.93	67.00	91	829
	Transformer	69.25	68.94	68.82	68.88	128	56
	ResNet-18	68.36	70.07	67.05	68.53	187	1,061
D+E+P	1D-CNN	77.13	76.58	78.55	77.55	188	340
	MiniRocket	75.10	75.88	74.01	74.93	20	188
	WaveNet	69.94	71.22	69.90	70.55	97	835
	Transformer	75.34	74.55	75.43	74.99	120	69
	ResNet-18	**78.08**	74.38	81.61	77.83	236	1,233

With our experimental data, the added value of P signals looks limited, as we have commented in the previous sub-section for a slightly larger time-window. We verified that the P signal performance is not penalized by our 60 Hz sub-sampling,

by assessing performance also at 120, 180, and 240 Hz, according to the Biopac guide [192] (page 8), obtaining similar results. However, notably, the best performance is obtained by combining all the signal types together (D+E+P), which indicates that the investigated models are able to identify additional general patterns while learning to be robust also to possible sensor failure, as it happened for P. The ResNet-18 model trained on the full dataset achieves the highest accuracy of 78.08%, with the 1D-CNN reaching a close value of 77.13%.

The (E+P) dataset is particularly important as in the automated driving condition where vehicular signals do not depend on the driver activity. At high levels of driving automation, distraction detection is very useful to achieve an effective take-over request (TOR) [193], which is issued for instance in case of an ADF system failure or exit from its operational design domain (ODD). Results from the (E + P) dataset, where Transformer achieves a 69.25% accuracy, indicate a lot of room for research in this direction, particularly to mitigate the inter-individual variability issue.

Considering a more comprehensive metric, such as the area under curve (AUC) for the precision-recall curve, we obtain results substantially consistent with the F1-score (e.g., 0.73 and 0.74 in D for 1D-CNN and ResNet, respectively; and 0.85 and 0.86 in D+E+P for 1D-CNN and ResNet, respectively). Figure 6.15 presents the precision-recall curves for ResNet-18 in the different data type combinations (always with a 0.5s window). The chart confirms the overall superiority of the complete dataset, followed by the D+E combination, while addition of physiological data looks useful at low recall. Significantly, the complete system seems to be able to keep relatively high precision at very high recall levels.

As predictable, given its well-known training efficiency [184], MiniRocket has the lowest training times, between 12 and 20 seconds. But the training time is limited for all the other models as well (up to less than 4 minutes).

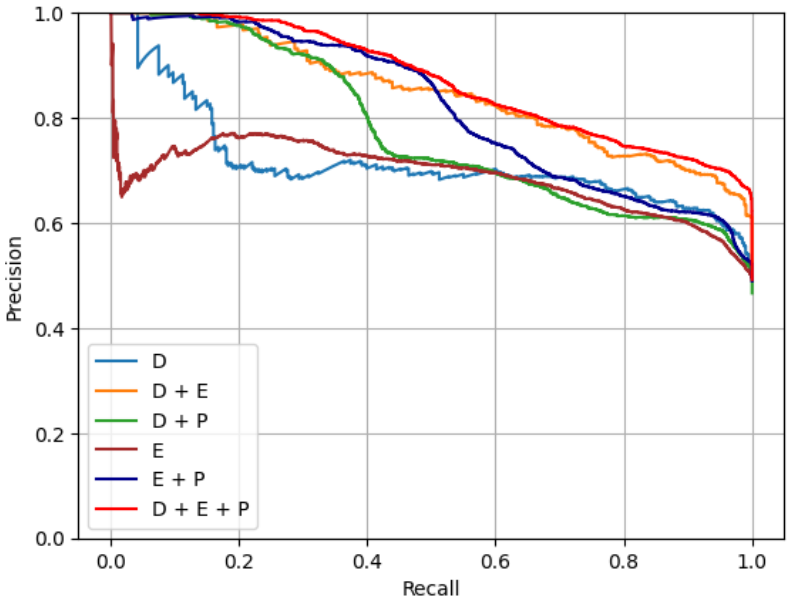


Figure 6.15: ResNet-18 precision-recall curves for different types of input data for the 0.5 s. window.

Inference times depend on the target platform, and we will discuss them later in the sub-section dedicated to embedded system deployability.

The size of the MiniRocket and Transformer models is very small (in the order of tens of KB), while 1D-CNN, ResNet-18, and, overall, WaveNet are one order of magnitude larger.

6.1.3.5 Error Analysis

Table 6.5 reports the confusion matrices of the 0.5s ResNet-18 for two sample users for whom the system performs quite less than the average (accuracy of 71.67% and 76.95%, respectively).

Table 6.5: Prediction confusion matrix for two sample users

	User 1		**User 2**	
Prediction / Label	**Positive**	**Negative**	**Positive**	**Negative**
Positive	33.30%	16.70%	49.94%	0.06%
Negative	11.23%	38.37%	22.99%	27.01%

Figure 6.16 shows the evolution of the predictions during their 180 s test.

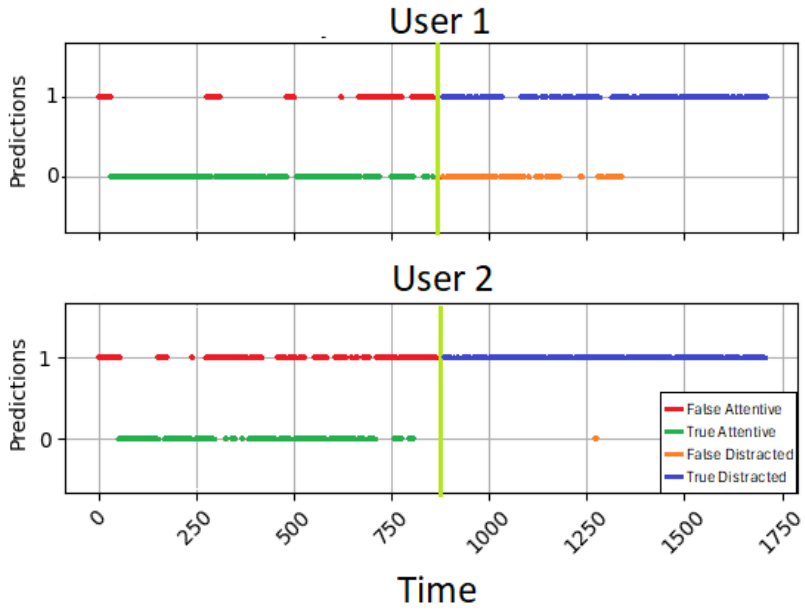


Figure 6.16: ResNet-18 binary predictions over time for two sample users.

The first sample clearly indicates the difficulty of our system in the transition phase (i.e., at the beginning of the TQT, which happens in the middle of the timeline). This is a clear challenge, essentially because actual distraction may not start immediately at the start of the task. On the other hand, the second user shows distracted behavior, at least according to our model, also without the cognitive task. For this user the system is very sensitive, but also very imprecise. These examples highlight the significant difference in subjects' responses and attitudes, and hints at future research work on better feature extraction, user segmentation, and custom fine-tuning of ML models. In this sense, it is useful to add that we tried to cluster the test users according to their metadata (e.g., total real-world driven km, age, sex), but could not achieve better results. This is worth more investigation in future work, since these factors may affect acceptance and effectiveness of ADAS (e.g., Son et al. [194]).

6.1.3.6 Explainability

We then employed SHAP analysis to interpret the predictions of our best-performing model, namely the ResNet-18 trained on the full dataset, applied to the most critical, 0.5-s time window size. The SHAP method was already introduced in Section 5.1.5. In the present chapter, it is used as an interpretability tool to

analyze the contribution of the different input features to the driver distraction detection task [195], [196].

Figure 6.17 shows the ResNet-18's top 10 features in terms of mean absolute SHAP values.

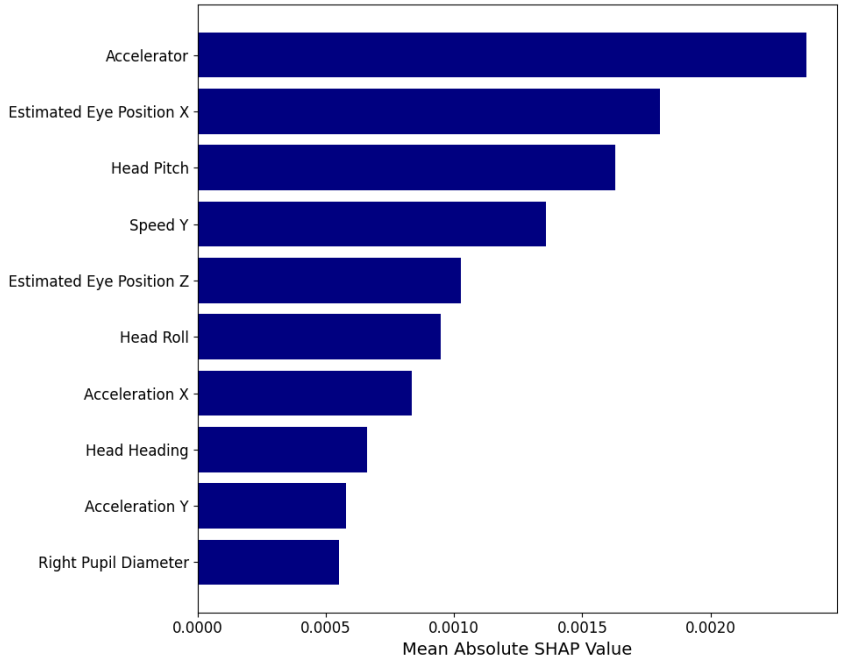


Figure 6.17: Top 10 features by highest absolute SHAP value for the ResNet-18 model on the full dataset.

Such values provide a global measure of feature importance and are thus useful to understand what features are more relevant, on average, for the decision making process of the analyzed model. It can be seen that six eye-tracker and four vehicular data have the largest values.

Particularly, the strongest contribution to the model's predictive power comes from the accelerator signal, the estimated eye position, the head pitch and roll, and the lateral velocity.

These findings align with and deepen the results presented in Table 6.4, where vehicular and eye-tracker data allowed reaching an accuracy of more than 75%. The SHAP top 10 feature ranking (Figure 6.17) does not include the HR and EDA physiological signals, consistently with the fact that P signals seem to contribute to overall performance less than the other two types of signals (Table 6.4). A specific result of our experiment analysis is given by the importance of the

accelerator signal as a strong indicator of a driver's cognitive "presence". Comparing the distribution of the normalized signal values under the two conditions confirms that distraction leads to a much more uniform driver behavior in terms of pedal activity (Figure 6.18).

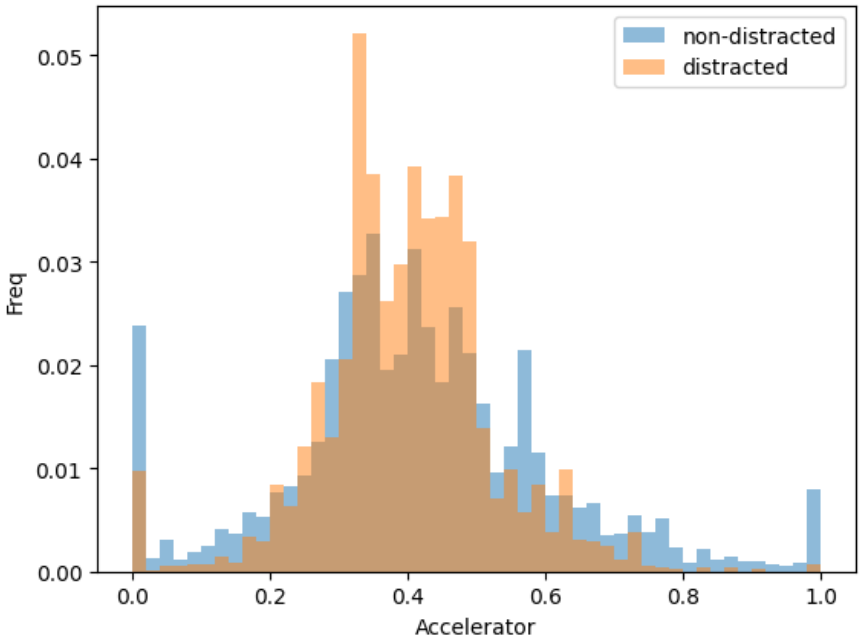


Figure 6.18: Accelerator signal profile in the two conditions

The average and standard deviation values are 0.42 (0.19) vs. 0.40 (0.13), in the non-distraction and distraction cases, respectively.

Going to a local analysis, Figures 6.19 and 6.20 show, as an example, the SHAP values for a correct "attention" and "distraction" sample classification, respectively.

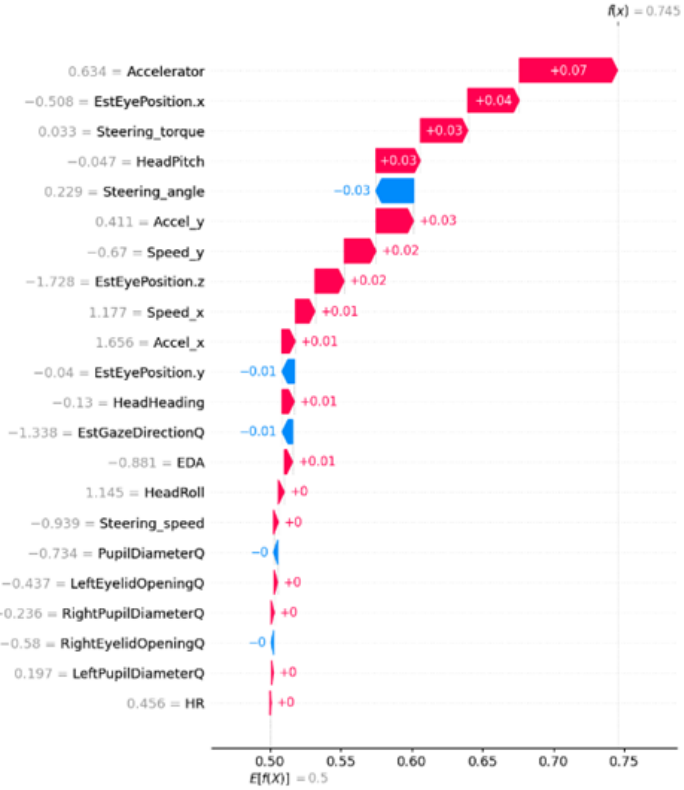


Figure 6.19: SHAP waterfall diagram of a correct “distraction” classification

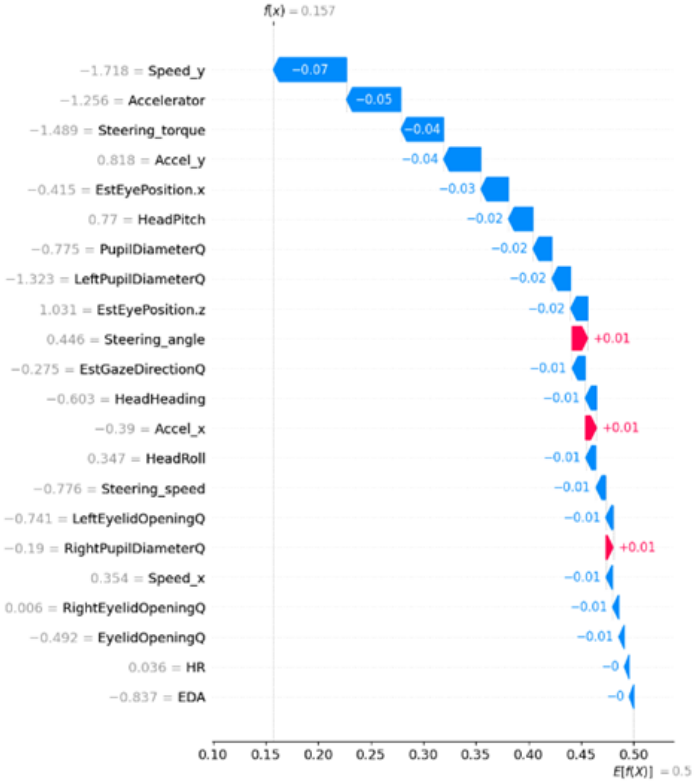


Figure 6.20: SHAP waterfall diagram of a correct “attention” classification

The charts confirm that signals such as Accelerator, estimated eye position, speed and accelerations (y and x) are main motivators for the decision. Looking at their specific (normalized) values, it is interesting to note, as above, that attention seems related to control (higher values for lateral speed, accelerator and steering torque), opposed to distraction (longitudinal acceleration, steering angle different from 0, but torque almost null). Eye position and head pitch seem to be more relevant to classify distraction than attention. Physiological signals (HR and EDA) provide a small, decision-reinforcing contribution.

Misra et al. [171] made a feature importance analysis on their ExtraTree classifier, and highlighted the relevance of eye-tracking data. In our case, the SHAP analysis reveals that the largest contribution comes not only from eye-tracker signals, but also from the vehicular ones, which is a novelty in literature and an important outcome of our experiment, since such data is easily available in vehicles. We argue that this advancement is due to our proposal of processing vehicular signals with deep learning models specifically dedicated to time series processing. From an industrial perspective, this can contribute to reduce of dependency on external (i.e., non-vehicular) systems, that are costly and invasive. Beyond the technical detail of the SHAP analysis, knowing that vehicular signals are an important factor for the decision making process by DL models in the driver cognitive distraction detection task is an arguably relevant novelty for non-expert stakeholders as well.

6.1.3.7 Fine-Tuning

In our experiment, the DDD has a user-generalizable performance of less than 80%. However, better results could be achieved with subsequent user-specific adaptation. To assess adaptation to an unseen user, we explored two methods: a single-user fine-tuning and a user-extended fine-tuning. In both cases, we adapt the fully trained ResNet-18 model by using 50% of the target user samples for the fine-tuning, while leaving the other 50% for the testing.

In the single-user case, we fine-tuned the model exclusively on a single new user (taken from our test set). Such fine-tuning was performed for 30 epochs with a

reduced learning rate (10-4, one decade less than the original), resulting in significantly improved metrics (Table 6.6, presenting the average over all the test users). However, this approach causes the model to lose its generalization capabilities, as it appears from a ~15% loss in all the metrics, averaging over all the other test users.

Table 6.6: Single User Fine-Tuning and User-Extended Training Methods

Method	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Original ResNet-18	78.08	74.38	82.08	77.97
Single user fine-tuning	98.52 (+20.44)	99.19 (+24.81)	97.78 (+16.17)	98.43 (+20.60)
User-extended training	96.04 (+17.96)	94.78 (+20.40)	95.86 (+14.25)	95.32 (+17.49)

In the user-extended fine-tuning experiment, instead, we incorporated the new user into the training set, fine-tuning the model using a 10-4 learning rate, but for only 10 additional epochs. We iterated the approach for each user in the test set, achieving a 17% average improvement in F1 score (Table 6.6). While this result is lower than that of the single-user case, this second method maintains the model's ability to generalize, showing for the other users in the test set a ±2% performance variation compared to the original model.

6.1.3.8 Deployability

In view of on-board deployment, Tables 6.7 and 6.8 report specific metrics (namely: model size, inference time, power consumption, and energy consumption per sample) measured on a system prototype deployed on a Raspberry Pi 5 microcontroller and a Jetson Nano microcomputer.

Table 6.7: Deployability Metrics on a Raspberry Pi 5

Dataset	Classifier	Model size onnx (KB)	Execution Time (µs)	Power Cons. (mW)	Energy Cons. (µJ)
D	1D-CNN	92.3	45	850	38
	ResNet-18	306.9	250	1,100	277
D+E+P	1D-CNN	97.1	50	950	48
	ResNet-18	350.4	270	1,200	314

Table 6.8: Deployability Metrics on a Jetson Nano (CPU)

Dataset	Classifier	Model size onnx (KB)	Execution Time (µs)	Power Cons. (mW)	Energy Cons. (µJ)
D	1D-CNN	92.3	175	1,550	271
	ResNet-18	306.9	860	1,600	1,376
D+E+P	1D-CNN	97.1	195	1,650	322
	ResNet-18	350.4	890	1,650	1,469

The Raspberry Pi 5 has compact dimensions and an affordable price, and can execute deep learning tasks efficiently, through a quad-core ARM Cortex A76 processor. The processor runs at a speed of 2.4 GHz, and the device includes 8 GB of RAM and a 32 GB SD card for storage and booting. The Jetson Nano, on the other hand, includes a quad-core ARM Cortex A57 processor that runs at a speed of 1.43 GHz, a NVIDIA Maxwell GPU, 4GB of RAM and a 32 GB SD card. These ranges of values are significant for automotive deployment, while other optimizations would be needed for boards having lower memory and computational capacity (e.g. STM 32), that we leave as future work.

To measure power metrics on the Raspberry PI 5, we employed a JT-UM120 USB multimeter connected between the power supply and the device to track power variations. For the Jetson Nano, instead, we used the “jtop” integrated NVIDIA software. Energy consumption is computed on a single 0.5s time window. We have limited our analysis to the ResNet-18 and 1D-CNN models, because their ML performance and memory footprint make them particularly promising for

embedded deployment. Results in Tables 6.7 and 6.8 show that the memory footprint of the models is under the 0.4 MB value, which is a typical limit for microcontrollers' ROM. Due to their higher complexity, the ResNet-18 models are 3.3x bigger than the 1D-CNN models. In all cases, the inference time is extremely short, making it compatible with real-time execution. We highlight that we reduced the latency by two orders of magnitude by converting the model from the .h5 to the Open Neural Network Exchange (ONNX) format, which has a significant impact also in terms of energy consumption. The difference in execution time between the Raspberry Pi 5 and the Jetson Nano can be attributed to the fact that the Raspberry Pi 5's CPU is a newer-generation quad-core ARM Cortex A76 processor, offering higher clock speeds (2.4 GHz) and improved architectural efficiency compared to the older quad-core ARM Cortex A57 processor in the Jetson Nano (1.43 GHz). Cross-referencing data from Table 6.4 and Tables 6.7 and 6.8, we note that while ResNet18 has a slightly better classification performance, its energy consumption is significantly higher than 1D-CNN, which highlights an important trade-off to be considered by designers.
Notably, for the Jetson Nano, Table 6.8 reports results for the CPU, because the GPU had longer inference times and higher energy consumption. We attribute this to the overhead of utilizing the GPU and the inefficiency of the parallelism for small batches and lightweight tasks. Particularly, compared with the same inference task on CPU, the inference time and energy consumption on the GPU are ~37x and ~9x higher for 1D-CNN and ResNet-18, respectively.

Overall, the results presented in this section demonstrate the feasibility of real-time cognitive distraction detection using time series-dedicated deep learning models under realistic deployment constraints. Using a strict split-by-user evaluation protocol, the proposed models achieve up to 78.1% accuracy in the binary setting with time windows as short as 0.5 s, confirming their ability to operate under low-latency requirements. The analysis quantitatively highlights the importance of vehicular and eye-tracking signals, which consistently outperform physiological signals and enable robust performance even in the presence of

sensor noise or partial data loss.

From a deployment perspective, the selected models meet the constraints of on-board embedded platforms, with model sizes below 0.4 MB, inference times in the order of a few milliseconds, and energy consumption compatible with continuous operation on devices such as Raspberry Pi 5 and Jetson Nano. These results confirm that time-series–specific deep learning models can achieve a practical balance between accuracy and resource efficiency when evaluated under realistic conditions.

While the binary classification setting provides a meaningful baseline for cognitive distraction detection, real-world driver monitoring systems require a finer-grained representation of driver state. In particular, distinguishing between attentive driving, distraction, and inattention is essential for advanced driver assistance and automated driving functions. Motivated by these quantitative findings, the next section extends the analysis to a multi-level driver distraction framework.

6.2 Multi-Level Driver Distraction States

While the previous study focused specifically on binary cognitive distraction detection, real-world driver monitoring systems must discriminate among a wider range of driver states, including attentive driving, distracted behavior, and full inattention. Addressing this broader problem is particularly relevant for advanced driver assistance and automated driving functions, where different driver states may require different system responses. As a result, this section extends the analysis to a multi-level classification framework that goes beyond the distracted/not-distracted paradigm.

To support this expanded task, a controlled experimental study was conducted, integrating multiple data sources to capture driver behavior under various distraction and inattention scenarios. The experiment was designed to approximate real-world driving conditions while ensuring reproducibility, high-quality data collection, and a safe protocol for all participants.

While there is a solid theoretical basis [197] and empirical data confirmation

[198] about the importance of distinguishing between inattention and distraction, no study with data collected from real-world (i.e., not simulated) driving has used more than two classes, to the best of our knowledge. To address this gap, we developed the 3-classes Driving while Distracted (DWD) dataset, accounting for attention, distraction, and inattention. The DWD data has been collected driving a specially equipped vehicle in a closed track. As in [197], we consider distraction as an intermediate state that includes cases where the driver is engaged in a secondary activity but still maintains some level of attention to driving. Thus, the distraction class accounts for scenarios where the transition to full inattention is not fully established, capturing momentary lapses or partial engagement in distractions. When addressing the 3-class data, we will use the following terms: (i) attention, when the driver is fully focused on driving; (ii) distraction, when the driver is distracted by some secondary activity but still keeps paying enough attention to driving; (iii) inattention, when the driver is strongly distracted and does not pay enough attention to driving. Distinguishing between these two levels of insufficient attention is key not only for designing a proper driver warning and information management strategy (e.g., [199], [200]), but also for properly managing the transition between different automation levels (also including level 0) [158], [201].

Table 6.9 proposes a synoptical picture to synthetize our state-of-the-art analysis. The table shows the number of open issues and highlights the comprehensive approach towards vehicular deployment that is the main characteristic and motivation of our work.

Table 6.9: Comparison of our proposed work with state-of-the-art solutions.

Article	Distract. types	Real world	Vehicular signals	3-class	Label by inspect.	<= 5 s. samples	Split-by-user	Shallow models	Deep learning	Expl. analysis	Depl. analysis
[202], [203]	Man/cog	✓			✓	✓			✓		
[204]	Man/cog	✓			✓	✓	✓		✓		
[205]	Vis/cog		✓	✓		✓		✓			
[206]	Generic	✓	✓			✓			✓		
[207]	Cognitive		✓					✓			
[208]	3 types		✓				✓		✓	✓	
[209]	3 types		✓			✓			✓		
[210]	3 types						✓	✓	✓		
[211]	Cognitive		✓			✓		✓	✓	✓	
[212]	Cognitive		✓			✓	✓	✓	✓	✓	✓
ours	3 types	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

6.2.1 Data Collection

To develop a comprehensive dataset for driver distraction and inattention detection, a controlled experimental study was conducted, integrating multiple data sources to capture driver behavior under various distraction scenarios. The experiment was designed to simulate real-world driving conditions while ensuring reproducibility and high-quality data collection, as well as a safe procedure.

All tests were conducted in a closed test track in Galizia, Spain (Figure 6.21), specifically designed to simulate urban driving conditions while eliminating external hazards such as traffic, pedestrians, or environmental uncertainties. This controlled environment helped minimize potential risks during the tests.

Parts of this chapter has been published in M. Fresta et al., "On-Board Deployability of a Deep Learning-Based System for Distraction and Inattention Detection," in IEEE Open Journal of the Industrial Electronics Society, vol. 6, pp. 1351-1369, 2025, doi: 10.1109/OJIES.2025.3601982.
The material has been adapted and extended to fit the structure and scope of this thesis.

Figure 6.21: The CTAG's test track, with distraction areas highlighted.

Each session lasted approximately 30 minutes, including a structured driving session of about 10 minutes, divided into a baseline segment (attentive driving) and multiple distraction scenarios.

6.2.1.1 Subjects

A total of 29 participants were recruited for the experiment, selected to ensure a various and representative sample of drivers. Participants were required to be between 18 and 65 years old, hold a valid driver's license with at least two years of driving experience, and be in good physical health, free from any conditions that could affect their driving performance. This ensured that all participants had sufficient driving experience and were capable of handling distraction tasks safely. We did not make any a priori grouping by age, speed preference, or other demographic criteria. Participants span from their late teens through their sixties, with most clustered in their mid-twenties to early thirties and a substantial segment in their mid-thirties to mid-forties, while fewer sit at the youngest and oldest extremes. The sample is predominantly male, though women represent a meaningful minority. In terms of education, the majority hold undergraduate degrees, vocational qualifications and postgraduate degrees are also well represented, and only a few have stopped at high school. Most participants reported many years of driving experience.

To maintain objectivity and prevent bias, participants were not informed of the specific distraction protocols being tested. This approach ensured that their responses to distractions were natural and reflective of real-world driving

behavior. All participants provided informed consent before participating in the study, in compliance with ethical research standards, including informed consent, risk mitigation, and data anonymization.

6.2.1.2 Distraction Tasks

Distraction scenarios were carefully designed to mimic real-life cognitive, visual, and physical distractions commonly experienced by drivers. Distraction tasks were performed only in two specific areas of the test track (Figure 6.21). The detail of the distraction tasks, taken at each round, is as follows:

1st Round - Reconnaissance tour: The driver conducted a reconnaissance tour around the track to familiarize with the environment and the testing conditions, providing a baseline for subsequent rounds.

2nd Round – Question answering: The driver was given a question to read and answer, testing the ability to focus on text while driving.

3rd Round - Picking up a coin: The driver was instructed to pick up a coin placed at the vehicle's door, simulating a situation where the driver needs to divert the attention outside the vehicle.

4th Round - Walkie-Talkie conversation: The driver was asked to interact with a co-pilot, engaging in a short conversation using a walkie-talkie, testing the ability to multitask.

5th Round - Key retrieval: The driver had to retrieve a key from a box placed inside the car (close to the gear shift), simulating a scenario where the driver needs to reach and handle objects inside the vehicle.

6th Round - Address searching: The driver was asked to find a specific portal number within the track's visual elements, requiring him/her to shift focus from the road to the track's surroundings.

7th Round - Mathematical operation: The driver was asked to solve a simple mathematical operation, testing the cognitive ability to perform mental tasks while driving.

8th Round - Visual hieroglyph: A visual puzzle in the form of a hieroglyph was presented to the driver, requiring him/her to use visual perception and problem-

solving skills during the driving task.

9th Round - Counting barriers: The driver was asked to count a set number of barriers placed along the side of the track, introducing a task that requires visual scanning and counting while maintaining control of the vehicle.
In summary, the experiment involved a structured set of Non-Related Driving Tasks (NRDT), including visual-mechanical distractions (e.g., picking up a coin, retrieving a key) and cognitive distractions (e.g., answering questions, solving mathematical operations, and counting barriers).

6.2.1.3 Safety Measures

To ensure participant safety, an expert co-driver was present in the vehicle during all test sessions. The inclusion of a secondary steering wheel and additional control pedals in the co-pilot's position ensured safety throughout the experiments, allowing the expert driver to take control of the vehicle in case of an emergency or unforeseen event. All tests were conducted under controlled conditions on a closed track, minimizing potential hazards. Prior to each session, participants were screened for any conditions or circumstances that could pose a risk during the driving sessions.

6.2.1.4 Data Acquisition

A specially equipped vehicle (Figure 6.22) was used to collect multimodal data, including vehicle dynamics, eye-tracking metrics, and driving lane information.

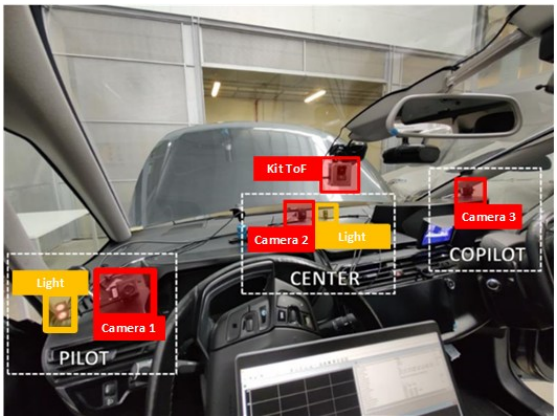


Figure 6.22: Interior of the vehicle equipped with cameras, illuminators and ToF sensor.

The vehicle recorded signals such as speed, acceleration, steering angle, and pedal pressure. A Smart Eye Pro eye-tracking system was employed to monitor gaze direction and blinking patterns. The eye-tracking features inserted in the dataset concern the estimated state of the left eye, right eye and of both eyes. Such features have three possible categorical values: unknown, closed, or open. An additional feature reports the percentage of time that the eyes were estimated as open in the last 15 seconds.

The EVK75027 evaluation kit (including the MLX75027 VGA time-of-flight (ToF) sensor and VCSEL illumination, with 110° FOV at 940 nm) was integrated into the cabin as well, to have precise spatial and motion tracking, even under low-light conditions. The ToF VCSEL video feed provided visual insights into head pose, hand movements, and secondary task engagement, which was essential for the labeling process (Section III.F), allowing annotators to verify instances of distraction and inattention with high accuracy. ToF VCSEL sensors differ from ordinary video cameras in that they record distance geometry (a per-pixel depth map and amplitude of the reflected IR pulse), rather than capturing full-color, high-resolution imagery. Since the resulting maps contain no texture nor color information, they inherently carry fewer personally identifying details.

Notably, the ToF VCSEL feed was used for labeling purposes only, and not included in the final dataset, according to our on-board deployment goal, trying to balance needs in terms of performance, cost, energy efficiency, privacy.

6.2.1.5 Data Synchronization

A signal synchronization module was implemented, in order to align each ToF VCSEL video frame with the relevant eye-tracking data and vehicular signals extracted from the vehicle Controller Area Network (CAN) Bus. This approach guarantees that all data streams remain temporally consistent, allowing researchers to analyze driver behavior across modalities. Data were stored in Hierarchical Data Format version 5 (HDF5) format, keeping precise synchronization. The data was structured into 29 session folders (one for each participant). The recorded data has a sampling frequency of 24 Hz (frame distance: 41.7 ms), ensuring a sufficient temporal resolution for capturing driver behavior dynamics. All timestamps were converted to seconds with seven decimal precision, enabling frame-by-frame alignment across all modalities.

6.2.1.6 Attention State Labeling

The labeling process was a critical step to ensure that the dataset provided accurate and reliable annotations for driver distraction detection. Labeling was performed by experienced annotators, who carefully reviewed the frame-aligned data recordings of each driving session. Using a proprietary, specialized labeling tool providing a comprehensive set of labels, the annotators analyzed posture, head movements, eye gaze, hand interactions, engagement with secondary tasks, and overall driver state. Given the inherently subjective nature of distraction and inattention cues, we set up a multi-step procedure to ensure consistency and reliability of the labeling, The dataset was initially annotated following an established protocol to minimize subjectivity and guide the annotation decisions. Then, we conducted a thorough internal review of the entire annotation set, identifying inconsistencies and edge cases. Based on this, a full re-annotation of the dataset was performed to improve consistency. A last round further refined and finalized the annotations. In our experience, this multi-step process could significantly enhance the quality and reliability of the labels.

Synchronization between data and events resulted in a well-structured and high-quality dataset, minimizing errors and maximizing the dataset's usability for

developing and validating machine learning models in real-world driver distraction applications.

At the end of the process, the data involved 279,486 timestamps (i.e., frames) labeled as attention, 17,486 as distraction and 62,471 as inattention. Figure 6.23. reports three samples from the 2nd Round “Question answering” distraction task (wrt. sub-section III.B), that exemplify some major differences among the classes. First, the driver is driving, with both hands on the steering wheel (a). Then, she receives a paper from a passenger, while still partially looking at the road (b). Finally, she is engaged in reading the question (c).

Figure 6.23: Examples of the three attention states. (a) Attention; (b) Distraction; (c) Inattention.

The overall pipeline of the proposed DDD system is shown in Figure 6.24, as further detailed in the next sections.

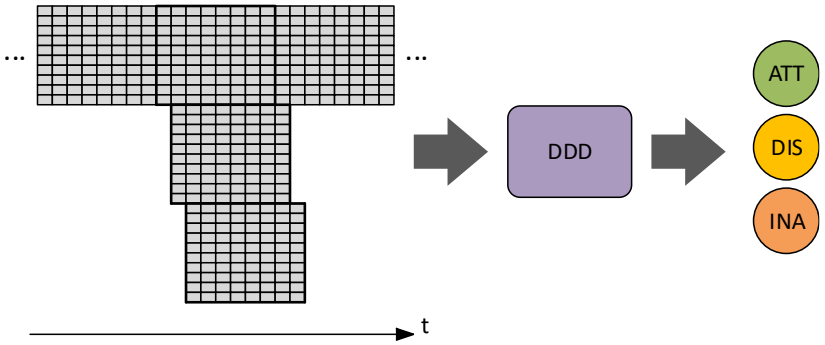


Figure 6.24: The DDD system classifies overlapped sliding windows.

6.2.2 Methodology

As per the latest literature (e.g., [209], [208], [212]), we framed the detection problem as a timeseries classification task, exploiting all the information present in the dataset signals. Thus, samples are given by the original signals windowed with a pre-defined length to capture sufficient temporal patterns related to distraction and inattention (Figure 6.25). The corresponding DDD system could work at the maximum frequency (i.e., the 24 Hz signal sampling rate), classifying overlapped sliding windows, with a 1 frame stride (41.7 ms).

The first ML data pre-processing step concerned standardization. Then, categorical states from the eye-tracker (e.g. eye open, closed or unknown) were one-hot encoded. The final dataset has a total of 24 features containing vehicle dynamics, eye-tracker-extracted high-level features, and information on vehicle position in the lane.

The main pre-processing step concerned the preparation of the samples. Mimicking normal conditions, the data collection campaign covered longer periods of attention than distraction. This resulted in an imbalanced number of samples across the three classes. To address it, we generated samples as sliding windows with a different stride (i.e., number of skipped frames between

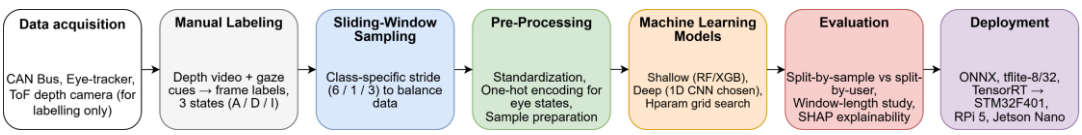


Figure 6.25: Proposed DDD system pipeline.

consecutive samples) for each class, to compensate for the imbalance. Sliding-window sampling is a well-established method for processing time-series [213], [214], and the use of adaptive strides to balance class representations was adopted in other studies as well [215], [216]. In the binary classification set-up, we used a stride of 6 for the attention class and a stride of 2 for the merged distraction classes. For the ternary case, we used stride values of 6 for the attention class, 1 for distraction, 3 for inattention. Figure 6.26 shows the applied stride for each class. This approach ensured a more balanced dataset, with an aim to reduce the potential bias towards the most represented class (i.e., attention).

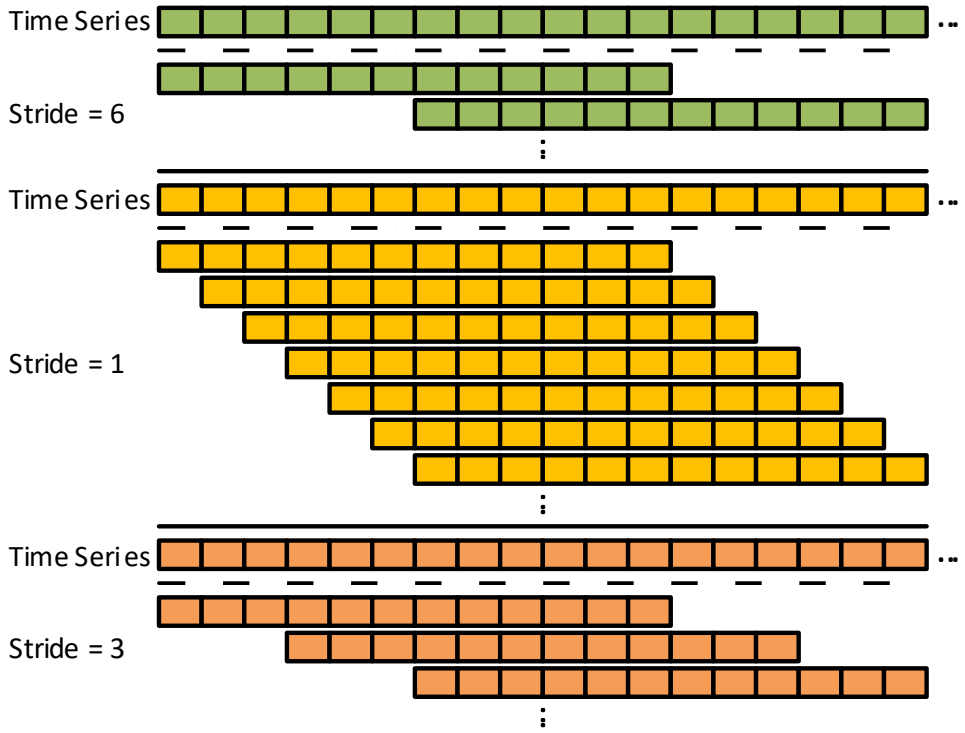


Figure 6.26: Sample extraction with different strides for Attention class (green, stride 6), distraction (yellow, 1), inattention (orange, 3).

Figure 6.27 shows the dataset class distribution for the 3-class case and time window length of 0.5 seconds, for a total of 84,711 samples.

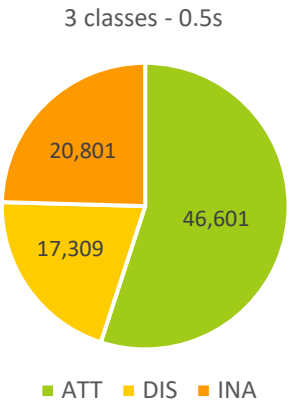


Figure 6.27: Dataset sample distribution (number of samples per class) for the 0.5 second window size.

For the 2-class task, distraction and inattention samples are simply merged. The distribution for other window sizes is similar, since windows are overlapped with stride 1 frame, with small differences at the borders.

To address the remaining class imbalance, we used oversampling-based data augmentation (with and without additive white Gaussian noise) and class-weighted loss, but saw no improvement.

The final dataset distribution after sample extraction is reported in Figure 6.27 for the 0.5 s window, and the distributions for the other window lengths are similar. Different stride values were used as a practical strategy to mitigate the limited number of original samples in some classes. However, small stride values generate highly overlapped windows and therefore more correlated samples, which may increase the nominal size of the dataset without providing a proportional increase in its effective variability. We therefore acknowledge this aspect as a limitation of the present study due to the limited dataset and the difficulty to collect driving time with distracted and inattention. Nevertheless, the split-by-user protocol prevents direct leakage between training and test subjects, so these correlated windows do not transfer across the evaluation split. A major methodological aspect concerned the comparison of two data split strategies: split-by-sample and split-by-user. The former is more common in literature (e.g., [205],

[206], [207], [209], [211]), but tends to provide an over-optimistic estimation, overfitting on individual, user-specific patterns, since it allows data from the same user to appear in both training and testing. The latter separates users among training, validation and testing ([208], [210], [212]), to provide a more realistic assessment of the model's ability to generalize across different individuals (i.e., predict correctly for unseen users), and guarantee better compliance with real-world deployment requirements [204].

6.2.3 Machine Learning Models

Developing the DDD system, we compared various state-of-the-art machine learning (ML) models. We studied both shallow and DL models dedicated to timeseries processing. For each model type, grid search was performed for hyperparameter optimization, to select the configuration able to reach the highest validation accuracy. The chosen model was then evaluated on the test set to determine its final performance. This section briefly introduces all the used model types and their hyperparameter tuning.

6.2.3.1 Shallow ML Models

Shallow models, particularly tree-based, such as Random Forest and XGBoost, are common in tabular classification tasks for their simplicity and efficiency (e.g., [211]).

1) Random Forest

Random Forest (RF) is an ensemble learning method based on decision trees, aggregating predictions from multiple trees to reduce variance and improve generalization [217]. We trained the model with a varying number of estimators (50, 100, 200), maximum tree depths (5, 10, 20), and maximum leaf nodes (50, 100), to find the best hyperparameter configuration.

2) XGBOOST

Extreme Gradient Boosting (XGBoost) is a gradient-boosted decision tree method, well established in the shallow ML state of the art because of its efficiency and scalability [218]. We optimized it using a grid search on learning

rates (0.3, 0.01), and maximum depths (3, 6).

6.2.3.2 Deep Learning Models

DL models are well-known for their ability to learn hierarchical feature representations from raw data, offering a more advanced and complex alternative to shallow networks. DL architectures can automatically extract relevant patterns from large amounts of data, making them particularly effective for sequential and high-dimensional inputs. We analyzed recurrent and convolutional architectures, along with hybrid combinations integrating both types, trying to capture both short-term and long-term dependencies, which is critical for accurate classification.

1) LSTM and Bidirectional LSTM

Long Short-Term Memory (LSTM) networks are a class of recurrent neural networks (RNNs) designed to handle mid-term dependencies by mitigating the vanishing gradient problem [219]. To further enhance temporal context, we also implemented bidirectional LSTM layers, which process the data in both the forward and reverse directions. Considered hyperparameters included numbers of layers (1, 2, 3), numbers of units (32, 64, 128), dropout rates (0.2, 0.3), and learning rates (0.001, 0.005).

2) GRU and bidirectional GRU

Gated Recurrent Units (GRU) provide a simpler architecture compared to LSTMs, reducing computational complexity while preserving performance in modeling temporal dependencies [220]. Due to their structural similarity with LSTMs, we adopted the same set of hyperparameter combinations in the grid search optimization.

3) 1D-CNN

1D-CNN are well-suited for time-series analysis due to their ability to capture local temporal patterns through shared weights and local connectivity and hierarchically combine them in the longer term [221]. The architecture we explored consists of different number of convolutional layers (1, 2, 3), kernel sizes (3, 5, 7), filter sizes (32, 64, 128), and dropout rates (0.3, 0.5). Max pooling layers

were used to reduce the resolution, and ReLU for activation. Grid search was performed also over batch sizes (16, 32), and learning rates (0.001, 0.005).

4) Hybrid combinations

To exploit the strengths of both convolutional and recurrent architectures, we evaluated also a small set of hybrid models combining 1D-CNN with LSTM layers. These models first extracted features using convolutional layers and then processed sequential dependencies through recurrent layers. Architectures were tested with different configurations in terms of number of CNN layers (1, 2, 3), number of LSTM layers (1, 2, 3), CNN filters (32, 64, 128) and LSTM units (32, 64, 128).

6.2.4 Experimental Results

The experimental analysis is structured to assess the impact of evaluation protocol, time-window length, data modalities, and model architecture on multi-level driver state classification under realistic deployment constraints.

6.2.4.1 Dataset Complexity and Shallow ML vs DL

In the first experiment, we adopted a split-by-sample approach, to compare the results with the literature. Thus, the samples from all the different subjects were shuffled together and the dataset randomly divided into training, validation, and testing sets, with a split ratio of 70%, 15%, and 15%, respectively. The raw signals were then windowed using a 1-second window size to allow direct comparison with the results reported in [211] and [212], that are the only works studying so short time-windows. To ensure a fair comparison, we processed both vehicular data and eye-tracker signals training a RF model for the binary classification task (merging the distraction and inattention classes). However, it is important to highlight that the datasets used in these studies differ significantly among each other. Our dataset is collected from real-world driving conditions, whereas both [211] and [212] rely on simulations. Moreover, both the studies focus on cognitive distraction, while our dataset includes a mix of distraction types, which increases classification complexity and may contribute to lower

overall accuracy.

Results in Table 6.10 (compared in terms of accuracy, the only one performance metric reported in [211] and [212] for RF) indicate lower results for our case.

Table 6.10: Comparison for the 2-class task, split-by-sample, 1 s. window.

Models	**Accuracy**
RF in [171]	95.08%
RF in [173]	94.70%
RF ours	81.48%

This may be due to the higher complexity of the mixed distraction tasks with respect to the cognitive distraction task and anyways indicates that our dataset is not easier to address, but presents a realistic and challenging benchmark for distraction detection.

On the other hand, Table 6.11 shows that a DL approach (all rows apart from the first two) allows reaching similar accuracy and F1-scores.

Table 6.11: Accuracy for 2 and 3 classes, split-by-sample, 1 s. window.

Models	**2 classes**		**3 classes**	
	Accuracy (%)	**F1(%)**	**Accuracy (%)**	**F1(%)**
RF	81.48	81.63	76.43	75.98
XGB	89.62	89.55	88.12	88.24
LSTM	93.45	93.49	92.85	92.67
BI LSTM	95.59	95.74	94.38	94.35
GRU	93.78	93.21	92.17	92.33
BI GRU	95.42	95.41	94.29	94.31
1D CNN	95.55	95.59	94.33	94.11
CNN+LSTM	95.62	95.53	94.52	94.69
CNN+BI LSTM	**96.34**	**96.32**	**94.84**	**94.86**

The results align with findings in the literature (e.g., [212]), highlighting that DL outperforms shallow models, demonstrating their ability to detect complex patterns from raw signals. This is confirmed by the fact that all the timeseries-

dedicated DL models reach similar results in the 3-class problem. For simplicity, Table 6.11 omits precision and recall values, as they are very similar to the reported accuracy and F1 scores. The full version of these tables is provided in Appendix A.

6.2.4.2 Split-By-User vs Split-By-Sample

Then, we analyzed the impact of the difference between the two dataset split designs: split-by-sample (seen in the previous sub-section) and split-by-user. For the split-by-user configuration, the dataset was divided into 21 users for training, 4 for validation and 4 for testing. To mitigate potential bias in user selection within the test set, we adopted a 5-fold cross-validation strategy, following the best practices suggested in [208]. In each iteration, 3 folds were used for training, 1 for validation, and 1 for testing, with the mean and standard deviation of the evaluation metrics reported as final results. Results were consistent across the various DL models.

The outcomes confirm our hypothesis of a discrepancy between the two experiment designs. For instance, the 1D-CNN model achieved an accuracy of 76.21% and 69,91% in the two and three class task respectively, indicating that the split-by-sample procedure (results reported in Table 6.11 in the 1D CNN row) overestimates performance by 25,4% in the 2-class problem. The gap grows to 34,9% in the more complex 3-class problem. This result is even more critical than that presented in [212], who measured a 19,8% increase in the 2-class problem using simulation data, with a ResNet-18 model. These findings underscore the critical importance of adopting a split-by-user design to properly assess model generalization. It provides a more stringent and realistic evaluation setting, as it avoids data leakage across users and tests the model's ability to extract subject-independent patterns. Thus, the remainder of our experiments were conducted using the split-by-user methodology, differently from the most common practice in literature.

6.2.4.3 Model Selection

We selected the model comparing different alternatives using a time window length that allowed a comparison with state-of-the-art real-time driver monitoring solutions that use split-by-user evaluation. We thus chose the 0.5 second length employed by [212], as solutions in [208] and [210] worked with window lengths incompatible with real-time performance (30 s. and 20-80 s. respectively).

We evaluated only DL models, given their superior performance known in literature and confirmed in our first, split-by-sample experiment.

Results in Table 6.12 show a mean accuracy of 73.36% for the 2-class problem, indicating a slight performance gap in a real-world scenario (vs. 75.15% in [212], using a 1D-CNN on simulated data).

Table 6.12: Accuracy for 2 and 3 classes, split-by-user, 0.5 s windows.

Models	**2 classes**		**3 classes**	
	Acc. (%) (std (%))	**F1 (%) (std (%))**	**Acc. (%) (std (%))**	**F1 (%) (std (%))**
LSTM	72.59 (2.42)	72.71 (2.38)	67.24 (2.66)	67.11 (2.60)
BI LSTM	72.75 (2.61)	72.68 (2.55)	67.04 (2.75)	67.22 (2.71)
GRU	73.16 (2.67)	73.04 (2.59)	66.94 (2.96)	67.08 (2.89)
BI GRU	73.27 (2.68)	73.42 (2.61)	67.35 (2.88)	67.28 (2.83)
1D CNN	**73.36 (2.42)**	**73.35 (2.44)**	**67.46 (2.74)**	**67.45 (2.70)**
CNN+LSTM	73.10 (2.64)	73.12 (2.58)	67.21 (2.92)	67.08 (2.85)
CNN+BI LSTM	73.15 (2.71)	72.71 (2.38)	66.96 (2.65)	67.11 (2.60)

In Table 6.12, the best performance is achieved by a 1D-CNN model, for which the grid search optimization identified hyper-parameter values effective for both the binary and 3-class tasks. The resulting architecture consists of two Conv1D layers, each one with 64 filters and a kernel size of 5, followed by a 1D max-pooling layer and a 30% dropout layer to prevent overfitting. The output dense layer has one neuron for binary classification and three neurons for 3-class classification. The model was trained for 30 epochs using a learning rate of 1e-3.

Since the DL models in Table 6.12 showed very similar performance, we

conducted an Analysis of Variance (ANOVA) test to check if there were any significant differences between them. The test was conducted on all seven models, resulting in an F-statistic of 0.197, meaning the differences among model performance were very small compared to the variations within each model. The p-value of 0.975, well above the standard 0.05 threshold, confirms that there is no statistical evidence supporting the hypothesis that any model is significantly better than the others. This suggests that, despite differences in architecture, the models perform similarly on this dataset, substantially confirming the prevalence of data and features over models [222], [223], [224].

Also, the literature indicates that 1D-CNN models are more efficient than LSTM and GRU for deployment [225].

Based on these considerations, we proceed for the rest of the experiments using the identified 1D-CNN model only.

6.2.4.4 Distraction Levels

A main feature of the proposed DDD system concerns the introduction of an intermediate distraction level, whose impact we investigate in this sub-section. Figure 6.28 shows the confusion matrices for the two- and 3-class tasks, at 0.5 s. and 5 s. time window length.

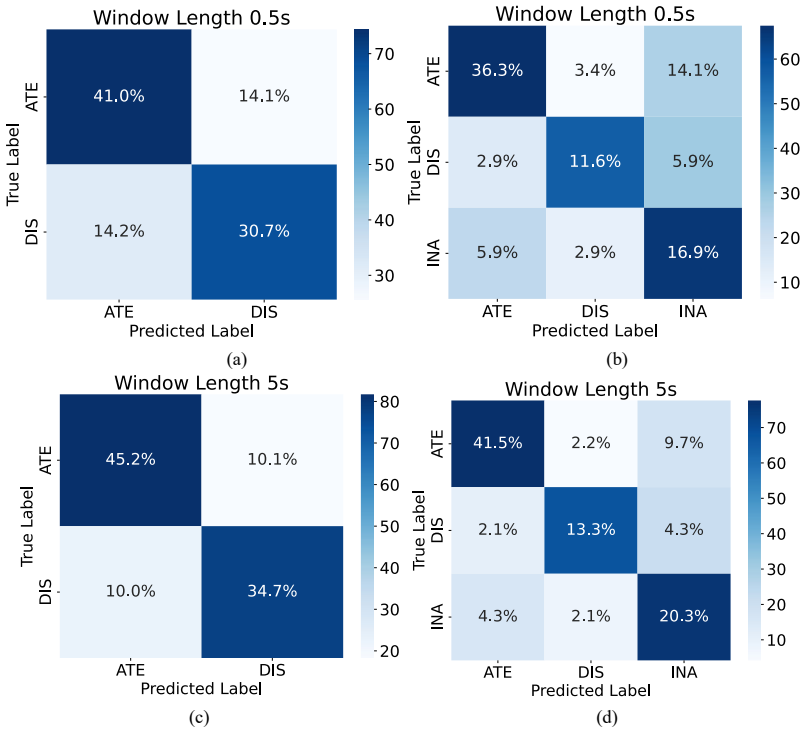


Figure 6.28: Confusion matrices for the 2-class (a, c) and 3-class tasks (b,d), with window lengths of 0.5s (a, b) and 5s (c, d). Color intensity is normalized row-wise.

For each matrix, color intensity is normalized row-wise, to make the error rates visually apparent, independent of the dataset unbalance. Focusing on the 0.5 s. window, the binary case shows the greater difficulty in classifying distraction/inattention samples, reflecting the dataset unbalance. In the 3-class case, instead, we observe that the intermediate class is the most difficult one to recognize and is frequently confounded with inattention. Surprisingly, attention samples are frequently classified as inattention. The reverse is also true, although to a lesser extent. Differently from the binary case, the ternary case shows that attention is under-predicted while inattention is over-predicted (particularly for the more critical 0.5-second window length), which may hint at a possible bias introduced by the stride-differentiation procedure presented in Section IV to augment the dataset. However, this is not true for the distraction samples, that were more heavily oversampled indeed (Figure 6.26). In general, care should be paid in future data collection to have enough original samples for each class.

To deepen the topic, we considered three variants of the binary classification task: (i) merge distraction with inattention, as already seen in Tables 6.11-6.12 and

Figure 6.28(a); (ii) entirely exclude the distraction class, thus only comparing attention versus inattention; (iii) merge distraction with attention. Results in Table 6.13 (0.5 s. window) confirm the challenge.

Table 6.13: Accuracy for variants of the 2-class task.

	Distraction in inattention	**Distraction in attention**	**No distraction class**
Accuracy (%) (std(%))	73.36 (2.42)	75.63 (2.60)	77.53 (2.44)

Interestingly, a higher accuracy is achieved when distraction is merged with attention (wrt. the case in which the intermediate distraction level is merged with attention), suggesting a significant overlap between these two states.

On the other hand, the 3-class confusion matrix (Figure 6.28(b)) shows the ability of the 1D CNN in distinguishing between distraction and attention, but also the challenge introduced by the intermediate level.

6.2.4.5 Time Window Length

As distraction and inattention is estimated on time windows, their length is an important hyperparameter of the model. While extending the window size tends to improve classification accuracy, excessively long windows risk missing short-duration distractions, which are important for detecting temporary losses of attention. Consequently, the optimal window length should achieve the right balance between capturing sufficient context and maintaining the resolution needed to detect brief distractions.

To assess the issue, we trained 1D CNN models on samples of varying window sizes from 0.5 to 7.5 seconds, that we consider an upper bound for real-time detection. Classification metrics are reported in Tables 6.14 and 6.15, for 2 and 3 classes respectively.

Table 6.14: Performance on different window lengths, 2-class task.

Window length (s)	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
0.5	73.36	73.32	73.38	73.35
1	76.21	76.22	76.20	76.21
1.5	77.08	77.10	77.06	77.08
2	78.36	78.44	78.33	78.38
2.5	78.99	79.14	78.96	79.05
3	80.14	80.13	80.10	80.11
3.5	80.48	80.69	80.41	80.55
4	80.88	80.93	80.83	80.88
4.5	81.13	81.26	81.11	81.18
5	81.76	81.86	81.72	81.79
5.5	81.93	81.94	81.80	81.87
6	82.17	82.29	82.14	82.21
6.5	82.38	82.50	82.37	82.43
7	82.51	82.64	82.48	82.56
7.5	82.65	82.79	82.63	82.71

Table 6.15: Performance on different window lengths, 3-class task.

Window length (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
0.5	67.46	67.50	67.41	67.45
1	69.91	67.11	66.85	66.98
1.5	71.01	71.21	70.95	71.08
2	72.68	72.75	72.57	72.66
2.5	74.15	74.14	74.20	74.17
3	75.41	75.42	75.37	75.39
3.5	75.94	75.99	75.90	75.94
4	76.38	76.45	76.31	76.38
4.5	76.82	76.89	76.58	76.73
5	77.62	77.58	77.67	77.62
5.5	77.98	77.99	77.96	77.97
6	78.23	78.29	78.17	78.23
6.5	78.44	78.41	78.52	78.46
7	78.68	78.62	78.72	78.67
7.5	78.79	78.70	78.85	78.77

Longer windows allow improving classification performance. However, the improvement decreases after 3 seconds, and further drops after 5 seconds, hinting at a saturation. After an initial reduction, the gap between the two and three class problem, seems to remain constant. Figure 6.29 shows the accuracy values reported in Tables 6.14 and 6.15. F1-scores are omitted, as they are very similar to the accuracy.

Figures 6.30 and 6.31 further support this trend by illustrating the values for the area under curve for the precision-recall curve (AUC PRC) and receiver operating characteristic (AUC ROC) values for the binary and ternary classification task, that measure model performance across different decision thresholds, providing a more comprehensive assessment. The results confirm all the previous considerations and make it apparent (particularly the AUC PR) that the window length should be at least of 5 seconds.

[212] showed a similar trend for the binary task on simulated data and cognitive-only distraction. The small differences concern the fact that we have slightly lower values, and the performance gap is slightly bigger for smaller windows. This suggests that our settings (real-world environment and three types of distractions) introduce additional complexity, making distraction detection more challenging, for which a 0.5 second window appears insufficient for robust classification. This is even more cogent for the 3-class problem.

Enlarging the window increases time resolution issues, particularly due to presence of inhomogeneous samples (i.e., samples including in their time window frames labeled with more than one value, as shown in Figure 6.32). In a first approach, we decided to use homogeneous samples only. For instance, dashed samples in Figure 6.32 would be discarded. This led to consistently improve model performance for longer time windows. However, this was due to the fact that, increasing the size, we were also discarding an ever-increasing number of samples, oversimplifying the task and overfitting the model on the most represented class(es). We thus decided to assign each sample to the class with the majority of frame labels. In this way, no samples are discarded, increasing the training task generality (and complexity).

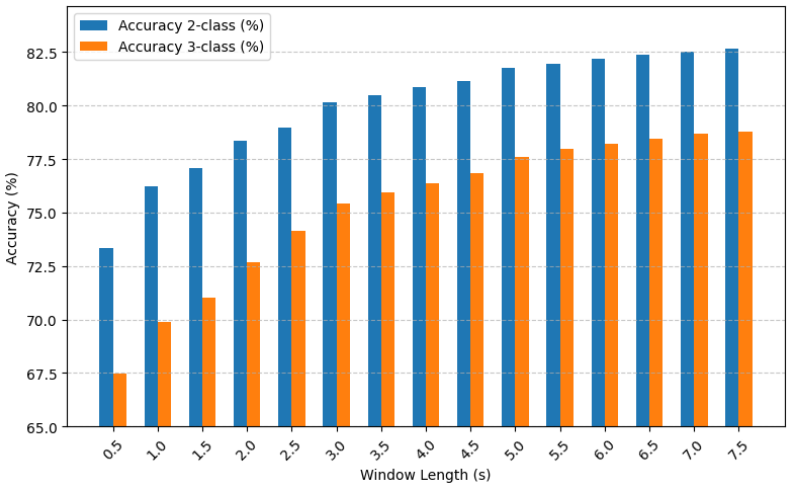


Figure 6.29: Accuracy per different window lengths (2 and 3 classes).

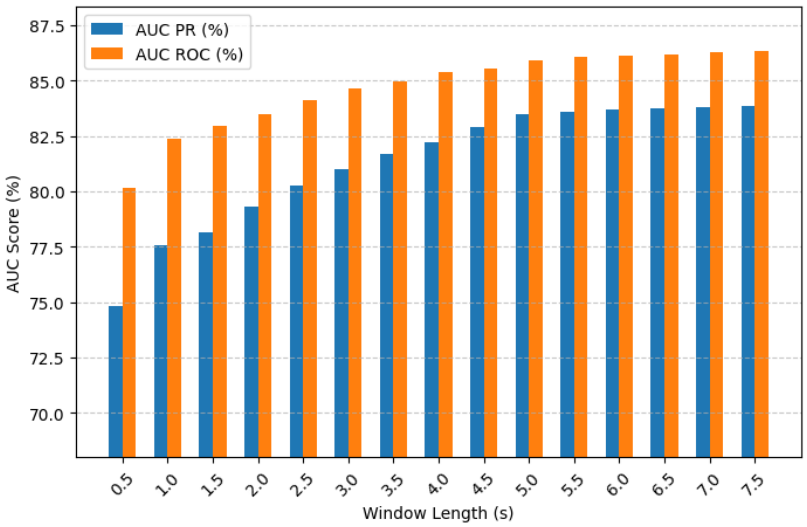


Figure 6.30: AUC PR e AUC ROC per different window lengths, 2-class task.

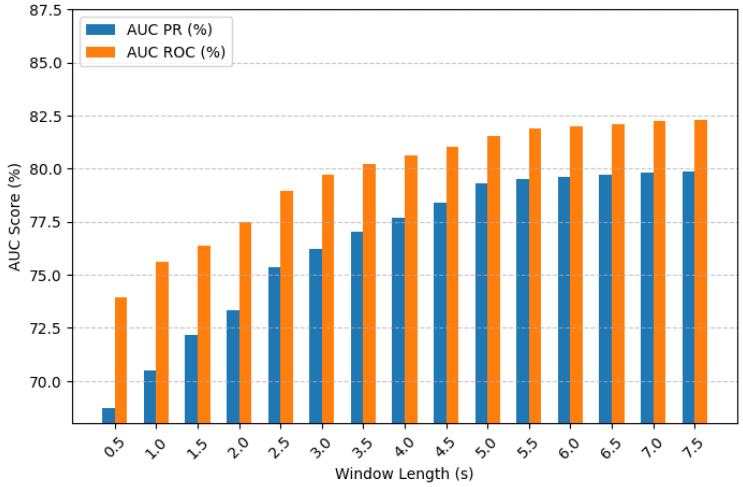


Figure 6.31: AUC PR e AUC ROC per different window lengths, 3-class task.

Figure 6.32 (which depicts two extreme cases in a 5 second period in which frames are labeled with two different classes) highlights also another aspect of the time resolution trade-off. Using a 5 second window length leads to missing the (short) distraction event in both case (a) and (b). Lowering the window length prevents this issue and reduces the percentage of inhomogeneous samples from 100% to 20% in case (a) and 40% in case (b). Case (b) also shows that shorter windows are subject more significant to border effects. For simplicity, the figure does not show the overlap of the samples.

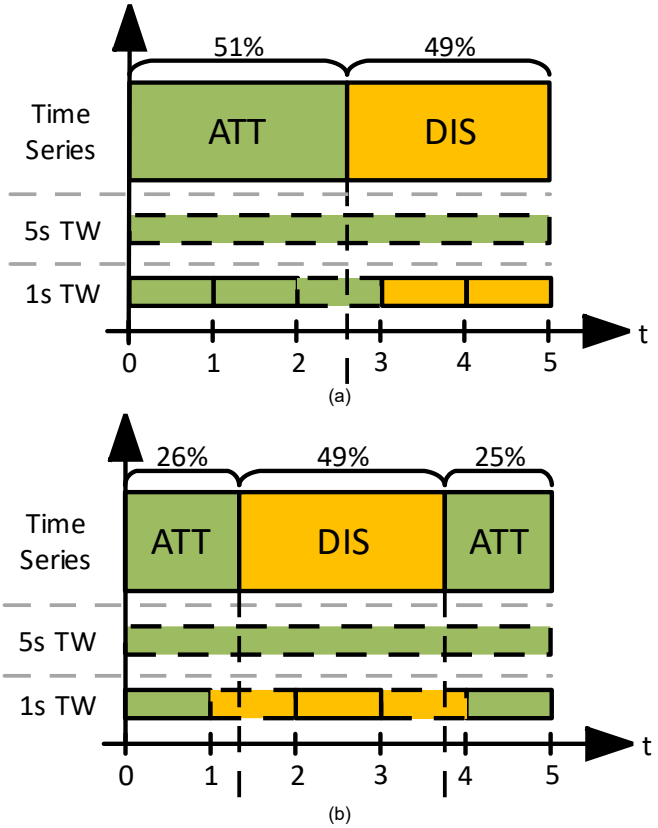


Figure 6.32: Comparison between 5s and 1s time window resolutions in two extreme example cases. Dashed samples include frames with multiple classes.

Concerning time resolution, we also mention that the Long_term_eye_state eye tracker's feature records the percentage of time that the eyes_state was estimated as open in the last 15 seconds. Thus, each sample brings information also from some time before its actual window length.

Other useful information for deciding the window size comes from analyzing the errors. The second row of Figure 6.28 ((c) and (d) for the 2 and 3-class task, respectively) shows the confusion matrices for the 5 second length. We can generally observe similar patterns as for the 0.5 second case (first row: (a) and (b)), with the significant difference of a scaling to better performance values. Notably, in the binary case the improvement is slightly more apparent for the attention class, while in the ternary case the improvement looks equally distributed across all the classes.

We highlight that, beside the mentioned dataset unbalance and time resolution issues, misclassifications (Figure 6.28) may be attributed also to another factor,

namely the feature set reduction between the labeling and the modeling phases. The labeling process, in fact, leveraged full-context visual data, including ToF video, to annotate behavioral cues such as head pose, hand actions, and posture. On the other hand, these features were excluded from the model inputs, introducing a semantic gap. As a result, the model is tasked with inferring cognitive states without direct access to the observable cues that informed the annotations.

Using a 5 second classification window yields the best detection accuracy. An automated system relying on these predictions may reduce the margin for safe takeover. However, a 5-second response time is broadly in line with the thresholds defined by Commission Delegated Regulation (EU) 2023/2590 [226], which mandates that Advanced Driver Distraction Warning (ADDW) systems alert drivers within 3.5 seconds of sustained gaze off the road at speeds above 50 km/h and within 6 seconds between 20 km/h and 50 km/h, with the possibility to extend these thresholds by up to 1.5 seconds under non-nominal conditions such as low lighting or sensor occlusions.

Based on all these findings, we selected the 5-second window 1D CNN model for further experiments, as it seems to provide the best trade-off between latency and time resolution on the one hand, and need for capturing sufficient temporal information to detect distraction on the other.

6.2.4.6 Explainability

Another aspect of our research concerned getting insights into the internal decision-making process of the DDD system, particularly to assess which input features contributed most to the model's predictions. We employed SHapley Additive exPlanations (SHAP) analysis to interpret the feature importance of the 5-second window 1D CNN models trained on the binary and 3-class classification tasks. Figure 6.33 plots the mean absolute SHAP values of the binary classification model. The results indicate that three vehicular signals (namely: longitudinal acceleration, vehicle speed, and gas pedal value) are by far the most influential features, followed by four eye-tracker features particularly the long-

term eye state.

Figure 6.34 plots the mean absolute SHAP values of the 3-class model, where feature contributions are also split across the classes. We can observe that gas pedal value is important for all the classes, particularly inattention. Acceleration is important for attention and inattention, with its lateral component is important for distraction. Notably, all the most important features are given by vehicular signals, with the exception of the long-term eye state. Moreover, steering wheel angle rate and lateral acceleration gain importance wrt the 2-class model, stressing the greater relevance of steering behavior and lateral vehicle control for a finer-grain detection. This seems to suggest that multi-class settings may require

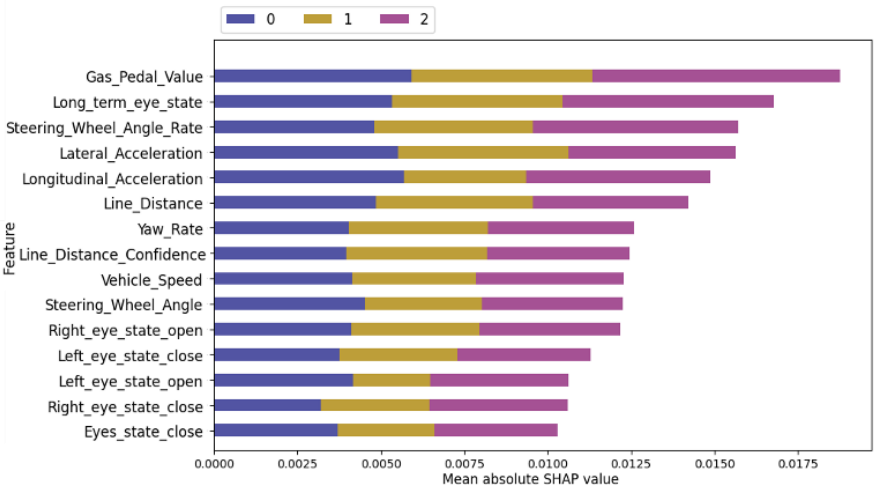


Figure 6.33: Mean absolute SHAP values (3 classes).

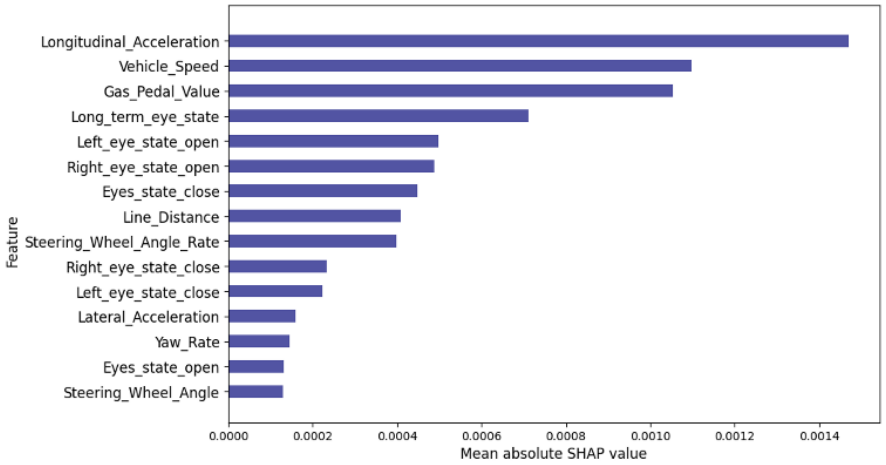


Figure 6.34: Mean absolute SHAP values (2 classes).

more consideration of lateral dynamics, in addition to the longitudinal factors (speed and acceleration).

These results confirm and strengthen, using real-world data, the awareness, raised only recently and only in simulation [212], of the importance of vehicular signals, which looks to be even more relevant in the 3-class problem.

6.2.4.7 Deployability

To evaluate deployability of the proposed 1D CNN models for driver distraction and inattention detection, we explored their deployment on three resource-constrained platforms such as the Raspberry Pi 5, NVIDIA Jetson Nano, and a mainstream STM microcontroller, that we consider as representative (also in terms of costs) of the range of embedded devices employable for the task in the target settings. The technical characteristics of the devices are reported in Table 6.16.

Table 6.16: Main features of the tested embedded boards.

Board	**CPU freq.**	**RAM**	**Flash**	**GPU**
STM32F401	84 MHz	96 KB	512 KB	✗
Raspberry Pi 5	2.40 GHz	8 GB	32 GB	✗
Jetson Nano	1.43 GHz	4 GB	16 GB	Maxwell

A first analysis concerns the neural network export format, which is an important factor for deployability. In fact, the Keras (.h5) format, which is the default option in desktop and cloud computing environments, has serious shortcomings for limited-resource settings. Table 6.17 compares its performance (in terms of model size and accuracy) with four other formats, namely Open Neural Network Exchange (ONNX), which is dedicated to cross-platform portability; TFLite 32, which is optimized for embedded devices;

Table 6.17: Performance of the different model formats.

Format	**Size (KB)**		**Accuracy (%)**	
	2 classes	**3 classes**	**2 classes**	**3 classes**
Keras (.h5)	395	440	82.11	77.78
ONNX	124	139	82.11	77.78
TFLite 32 bit float	123	138	82.11	77.78
TFLite 8 bit int	39	43	81.94	77.71
TensorRT	392	407	82.11	77.78

TFLite 8, which adds 8-bit quantization, to cut the memory footprint; and TensorRT, a deep learning inference optimizer and runtime tailored to NVIDIA GPUs [227]. Table 6.17 shows the greater complexity of the 3-class task (larger models, with lower accuracy). The accuracy columns have the same values for all the formats apart from the quantized one, because they simply encode in different ways the same network information. The TFLite 8 accuracy is lower, but only slightly, as quantization modifies the model. This is reflected by the significant drop in the size columns compared to the other formats. The other results highlight the fact that Keras encodes a model in a .h5 file, which is a generic hierarchical database format, unsuited for embedded deployment. TFLite, while being dedicated to light-weight devices, has a similar footprint than the ONNX portability format. The TensorRT format has a larger memory footprint, comparable to the Keras .h5 one, as it targets GPU-equipped devices (typically in the cloud or on desktop), not field-deployable boards.

Tables 6.18 and 6.19 extend the comparison from static characteristics to dynamic performance (in terms of average latency, and power and energy consumption per predicted sample), considering the three above mentioned embedded platforms.

Table 6.18: Embedded deployability metrics for the 2-class model.

Format	STM32F401			Raspberry Pi 5			Jetson Nano		
	Latency (ms)	**Power (mW)**	**Energy (mJ)**	**Latency (ms)**	**Power (mW)**	**Energy (mJ)**	**Latency (ms)**	**Power (mW)**	**Energy (mJ)**
Keras (.h5)	234.51	11.72	2.74	58.62	1,650	96.72	*115.02	*57.36	*6.60
ONNX	244.91	12.54	3.10	0.15	750	0.11	*0.85	*266.24	*0.23
TFLite 32 bit	212.13	16.24	3.44	0.23	2,200	0.51	0.64	66.48	0.04
TFLite 8 bit	51.12	11.31	0.60	0.19	1,500	0.29	0.57	51.60	0.03
TensorRT	N/A	N/A	N/A	N/A	N/A	N/A	*0.11	*180.03	*0.02

*Model running on GPU

Table 6.19: Embedded deployability metrics for the 3-class model.

Format	STM32F401			Raspberry Pi 5			Jetson Nano		
	Latency (ms)	**Power (mW)**	**Energy (mJ)**	**Latency (ms)**	**Power (mW)**	**Energy (mJ)**	**Latency (ms)**	**Power (mW)**	**Energy (mJ)**
Keras (.h5)	234.71	11.81	2.77	59.10	1,750	103.43	*119	*59.52	*7.10
ONNX	245.62	12.62	3.10	0.16	775	0.13	*0.87	*260.24	*0.23
TFLite 32 bit	212.53	16.33	3.47	0.23	2,250	0.52	0.62	81.62	0.05
TFLite 8 bit	51.21	11.74	0.60	0.19	1,600	0.31	0.52	54.48	0.03
TensorRT	N/A	N/A	N/A	N/A	N/A	N/A	*0.14	*185.07	*0.03

*Model running on GPU

The power consumption was measured using a multimeter for the Raspberry Pi, the jetson-stats library [228] for the Jetson Nano, for board monitoring and control, and the STM32 Power Shield (Nucleo expansion board) [229] for the STM32 board. In the reported Raspberry Pi tests, inference was executed by exploiting both CPU and GPU resources. To determine energy consumption, inference time was recorded and averaged over 100 runs for each device.

The results clearly highlight substantial variability across different combinations, underlining some clear patterns and critical trade-offs between inference speed and energy efficiency. Raspberry and Jetson Nano have different performance

characteristics than the STM32F401. Figure 6.35 highlights that, in general, they reduce latency by two orders of magnitude with respect to the STM32 microcontroller.

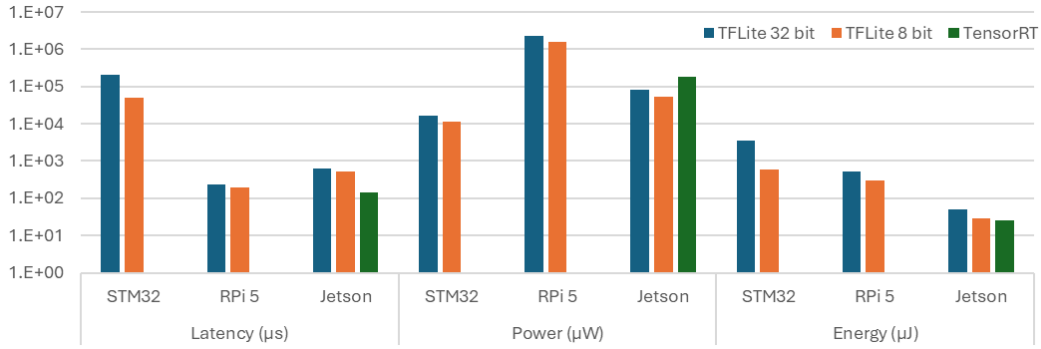


Figure 6.35: Performance chart (3-class model).

On the other hand, considering energy consumption, Jetson outperforms Raspberry by one order of magnitude and STM32 by two. Raspberry has a quite higher power consumption, especially compared to the microcontroller, but its higher speed allows for lower overall energy consumption. The measurements seem to indicate that the STM32 device is not energy efficient. However, considering the whole functioning cycle, it must be stressed that idle consumption is approximately 42 mW for the STM32 microcontroller, 2.4 W for the Raspberry, 0.6 W for the Jetson Nano. These results also suggest that, for the compact models considered in this study, GPU acceleration is not always strictly necessary to satisfy real-time constraints. In particular, inference times are already well below the 100 ms threshold commonly considered compatible with real-time operation, even without relying on the highest-performance configuration. However, this does not mean that GPU support is irrelevant. GPUs still provide a clear advantage in terms of parallelization and can further reduce inference time, thereby leaving more computational margin for additional control tasks, communication with sensors, and data acquisition processes running on the same system.

Moreover, some devices have ultra-low-power sleeping modes (200mW for the Nano) and duty-cycling techniques could be applied to reduce consumption.

Analyzing the formats, we observe that for Raspberry and Jetson Nano, the Keras

.h5 format shows higher latency and energy consumption, making it unsuited for embedded deployment. ONNX significantly reduces latency and energy consumption, confirming the literature [230]. The value of the TFLite optimization is demonstrated by its low latency, which makes it faster than the previous two formats and able to reduce energy consumptions by one order of magnitude. Allowing the exploitation of the GPU, TensorRT considerably reduces latency and energy consumption of the Jetson Nano deployment. Using the GPU, this board achieves the lowest latency and energy consumption. However, it is fair to point out that the Raspberry Pi 5 (86 mm × 56 mm × 16 mm and 56 g) is a considerably smaller and lighter device than the Jetson Nano (100 mm × 80 mm × 29 mm and 141 g). On the other hand, STM32F401 is less sensitive to the different formats, denoting higher latency (and thus also energy consumption), likely because of its much slower processor.

The 8-bit quantized format leads to reduced latency and power across all the platforms (but with the greatest advantage for the STM32 microcontroller), thus reducing also energy consumption. Notably, this comes at the cost of only a limited drop in accuracy (Table IX), which highlights a trade-off that should be carefully considered by production engineers. Comparing Tables 6.17 and 6.19, we also see that the 3-class problem only slightly increases latency and energy use across all the configurations.

Concerning the real-time performance constraint, both Raspberry Pi and Jetson Nano are well able to satisfy the maximum possible 24 Hz output rate, equal to the sampling rate. On the other hand, the F401 STM32 microcontroller achieves a processing rate of about 20 Hz, thus requiring halving the output rate, which may anyways be acceptable (and notable, considering the extremely embedded hardware characterization of this platform).

6.2.5 Limitations

While the proposed model uses a limited set of signals, the system nonetheless raises ethical and privacy questions that go beyond mere data minimization. A

fully responsible approach would involve explicit threat modeling (to anticipate misuse or re-identification risks), clearly defined data governance (to specify who can access what, under which conditions) and, overall, user-acceptance studies, to ensure that the system is useful for the driver and drivers feel comfortable and in control. All feature extraction happens on the fly inside the vehicle and no raw imagery, eye-tracking nor depth data is stored, thereby reducing long-term privacy exposure even as this underscores the need for robust transparency and ongoing oversight.

Eye-tracker devices can be quite expensive for deployment. However, cameras looking at the driver are already available in some medium/high level cars and more is expected in the next years. A scalable approach processing different types of inputs depending on which data is available in a particular vehicle model, can be a topic of future research.

The study did not evaluate model performance across demographic subgroups. This means that any subgroup differences would arise indirectly, through diverging movements or other patterns rather than explicit age or sex inputs. Stratifying the data by demographic groups could yield improved performance. However, we prefer not to collect demographic metadata also for privacy reasons.

All experiments are conducted on data from a single controlled track. The model's generalization to other real-world driving scenarios remains untested. The dataset is proprietary. However, public release of this kind of data would be very beneficial to support research in the field.

We conduct a ML-oriented performance analysis, with limited focus on its implications for the actual driving experience. Nevertheless, this analysis serves as a critical foundation for informing real-world vehicle testing, which will be essential in designing effective Human-Computer Interaction (HCI) systems. Such testing will enable the assessment of end-user acceptance, the optimization of warning thresholds, the measurement of user annoyance, and the trade-off between missed detections and false alarms.

Overall, the results presented in this chapter demonstrate the feasibility of DDD based on time series-dedicated deep learning models under realistic deployment

constraints. Using a strict split-by-user evaluation protocol, the proposed models achieve around 76% accuracy in the binary setting and approximately 70% accuracy in the three-class setting with short 0.5 s time windows, confirming the difficulty of the task when generalization across unseen users is required.

The analysis highlights the clear advantage of deep learning approaches over shallow models when dealing with heterogeneous and non-stationary signals, and shows that model performance is strongly influenced by both time-window length and inter-individual variability. In particular, extending the window length to 5 s consistently improves accuracy and AUC metrics, providing the best trade-off between temporal resolution and classification reliability.

Finally, explainability analyses identify vehicular and eye-tracking signals as dominant contributors to model decisions, while deployability experiments demonstrate sub-millisecond inference latency, sub-megabyte model sizes, and low energy consumption on embedded platforms. These results confirm that accurate and real-time driver monitoring is achievable on resource-constrained, on-board hardware, reinforcing the applicability of the proposed approach to real-world automotive systems.

CHAPTER 7

7. Towards Robust and Deployable Edge AI on FPGA

In the preceding chapters, this thesis has addressed the collection and management of time-series data, as well as the training and deployment of lightweight ML models on resource-constrained embedded platforms.

These approaches have been applied to different application domains, including human activity recognition, structural health monitoring, and driver state recognition, showing how time-series-based models can be brought close to real-world edge scenarios.

The experimental results discussed so far demonstrate that software-based deployment on microcontrollers, single-board computers, and GPU-enabled edge devices can already support real-time inference in several use cases. At the same time, these solutions expose practical limitations when tighter requirements on power efficiency, latency, and low-level control over computation are imposed, as often happens in safety-critical or always-on systems.

A natural next step is therefore to investigate deployment strategies that move beyond general-purpose processors and rely on hardware-level acceleration. FPGAs offer a different execution paradigm compared to CPUs and GPUs, enabling customized parallelism, fine-grained control over arithmetic precision, and deterministic behavior. These characteristics make FPGAs particularly appealing for IoT applications, where memory footprint and energy efficiency are as important as raw performance.

The work presented in this chapter is intentionally more exploratory than the fully developed studies discussed earlier. A substantial part of the thesis has been dedicated to building a solid experimental environment (covering data pipelines,

model training, evaluation metrics, and embedded deployment workflows) on top of which hardware-specialized implementations can now be explored.

Accordingly, this chapter explores FPGA-based implementations of lightweight models, targeting the Xilinx ZCU104 platform. It discusses the motivation, design considerations, and initial experimental results of this transition, focusing on hardware-aware deployment beyond software-based embedded solutions.

7.1 Battery State-of-Charge Estimation on FPGA

Lithium-ion batteries are a core component of modern electric and hybrid vehicles, power tools, and energy storage systems, where reliability, safety, and efficiency are critical requirements [231], [232], [233]. To operate batteries safely and effectively, Battery Management Systems (BMS) must continuously monitor internal states that cannot be directly measured, among which the State of Charge (SOC) plays a central role [234]. SOC represents the amount of usable energy remaining in the battery and directly impacts range estimation, power availability, and protection against over-charge and over-discharge conditions [235].

Accurate SOC estimation is a challenging task due to the complex, nonlinear, and time-varying behavior of lithium-ion batteries, which is influenced by temperature, load profiles, aging, and usage history. Traditional model-based approaches, often combined with adaptive filtering techniques such as Kalman filters, H-infinity filters, or particle filters [236], rely on precise battery models and extensive calibration. While effective under controlled conditions, these methods may struggle to generalize across different battery chemistries, operating regimes, and degradation states.

In recent years, data-driven and deep learning approaches have shown strong potential for SOC estimation by directly learning temporal dependencies from battery measurements [237], [238], [239], [240]. By treating voltage, current, and

Parts of this chapter has be published in Faryar, D., Berta, R., Fresta, M., Saad, A., Lazzaroni, L,. Ballout, H., Srour, O., Bellotti, F. (2026), “Lightweight Deep Learning for SOC Estimation of Various Lithium-Ion Batteries on Xilinx ZCU104 FPGA” In: Ruo Roch, M., et al. Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2025. Lecture Notes in Electrical Engineering, vol 1553. Springer, Cham. https://doi.org/10.1007/978-3-032-17174-0_4
The material has been adapted and extended to fit the structure and scope of this thesis.

temperature signals as time series, neural network models can capture complex battery dynamics without explicit physical modeling. However, deploying such models in real-world BMS scenarios introduces additional constraints: estimators must operate in real time, consume limited power, and fit within the memory and computational budgets of embedded hardware.

These constraints make battery SOC estimation a particularly relevant case study for FPGA-based Edge AI. As discussed earlier in this chapter, FPGAs provide deterministic execution, fine-grained control over numerical precision, and energy-efficient parallel computation [58], which are attractive properties for always-on battery monitoring systems. In this context, SOC estimation represents an ideal benchmark to investigate how time-series-based deep learning models can be adapted, simplified, and mapped onto hardware accelerators while preserving acceptable estimation accuracy.

In the following, we evaluate several lightweight deep learning architectures for SOC estimation and analyze their trade-offs in terms of accuracy, memory footprint, and suitability for FPGA deployment. The best-performing model is then synthesized on a Xilinx ZCU104 platform to assess its feasibility for low-power, real-time operation in embedded battery management systems.

7.1.1 Dataset

This study utilizes the UNIBO Powertools Dataset [241], collected by the University of Bologna to analyze the behavior of Lithium-ion batteries during repeated charge and discharge cycles. The dataset was specifically designed to evaluate a variety of battery cells intended for use in cleaning equipment such as vacuum and automated floor cleaners. Our analysis focuses on the standard test configuration, and the dataset comprises 16 batteries from different manufacturers, with nominal capacities ranging from 2.0Ah to 4.0Ah. The experiments were designed to capture battery behavior across various stages of usage, from beginning of life to end of life.

In the standard test protocol, batteries were discharged at a constant current of 5A.

The cycling procedure followed a structured approach: the charge cycle was conducted using a Constant Current-Constant Voltage method at 1.8A and 4.2V with a 100mA cut-off, while the discharge cycle applied a constant current until the cut-off voltage of 2.5V was reached. This cycle was repeated 100 times. During each discharge cycle, data was recorded every 10 seconds, providing high-resolution temporal information on battery dynamics. The input features included voltage (V), current (I), and temperature (T) measurements, with the corresponding SOC percentage as the target output for model training and evaluation.

Unlike the original dataset configuration [241], which uses the full sequence of 287 time steps as model input, and thus requires each cycle to start from a fixed full-charge reference at 100 %, our approach adopts a flexible windowing strategy to segment the data into shorter sequences of fixed length. This means that it can be applied at any SOC point, and we are not constrained to begin from 100 %, allowing the model to operate during partial cycles, mid-discharge, or real-world varied usage conditions, making our method far more advantageous in practical scenarios.

Also, the shorter windowing design choice aims to reduce the amount of input data required for each prediction, searching for smaller models suitable for edge deployment. However, this also increases the difficulty of the task, as the model SOC prediction has a more limited temporal context. We evaluated four different window lengths, in particular 6, 12, 18, and 30 time steps, corresponding to 1, 2, 3, and 5 minutes, to assess the impact of sequence length on model performance. To augment the dataset and increase the number of training samples, we applied a sliding window with a stride of 1. The original dataset already includes a predefined training-test split, and we further partitioned 20% of the training set to serve as a validation set for model selection during hyperparameter optimization. For each model architecture, we conducted 50 Bayesian optimization trials, with a maximum of 100 training epochs per trial. An early stopping criterion with a patience of 15 epochs was applied based on validation loss. The results reported in this section correspond to the best-performing configuration identified on the

validation set.

7.1.2 Deep Learning Models

We implemented and evaluated five deep learning architectures for the task of SOC estimation, each selected for its ability to model temporal dependencies and extract relevant patterns from sequential data. To optimize model performance, we employed Bayesian optimization to explore the best configuration of hyperparameters within predefined ranges for each architecture:

LSTM and Bidirectional LSTM models: Both architectures were configured with 1 to 3 layers, each containing 32 to 256 units (step size of 32), using the tanh activation function. The return_sequences parameter was enabled for all layers except the final one to maintain compatibility with the dense. The bidirectional variant enhances context modeling by processing sequences in both directions.

GRU and Bidirectional GRU models: Like the LSTM-based architectures, they featured 1 to 3 layers with 32 to 256 units per layer. GRUs provide a more lightweight alternative to LSTMs, with faster training and fewer parameters. The bidirectional GRU extends this by adding context from both directions of the input sequence.

1D-CNN model: The 1D-CNN model included between 1 and 3 convolutional layers, each with 32 to 256 filters with step 32 and kernel sizes ranging from 2 to 5. The SELU activation function was used, and padding 'same' to preserve sequence length.

Bayesian optimization also tuned the number of dense layers (1-3), units per layer (64 -512), dropout rate (0.0-0.5), and selected the learning rate from a discrete set of values: 1×10-5, 5×10-5, 1×10-4, 5×10-4, 1×10-3. All models were trained using the Huber loss function, which ensures robustness to outliers and stable convergence.

7.1.3 Experimental Results

To evaluate the effectiveness of the proposed approach, we conducted different experiments, comparing different architectures and window size (Table 7.1). Among all tested models, the 30 window size GRU demonstrated the best trade-

off between accuracy and size, achieving near the lowest error values (RMSE = 3.379%, MAE = 2.693%) while maintaining a compact .tflite size of just 741 KB. This configuration was selected for further deployment and analysis. The selected model configuration consists of a single GRU layer with 192 units, followed by three fully connected dense layers. A dropout layer with a rate of 0.1 was applied after the GRU. The dense layers were configured with 160, 64, and 512 units respectively, interleaved with dropout of 0.2 and 0.1 after the first and second dense layers. The learning rate was set to 1×10-3.

Table 7.1: Performance comparison of models for different window sizes.

Model	**Window Size**	**RMSE (%)**	**MSE (%)**	**MAE (%)**	**Size (KB)**
LSTM	6	4.663	0.217	3.730	1,199
	12	4.430	0.196	3.544	1,628
	18	4.103	0.168	3.282	790
	30	4.096	0.168	3.277	1,118
Bi-LSTM	6	4.519	0.204	3.615	4,760
	12	4.282	0.183	3.426	1,290
	18	4.157	0.173	3.325	2,820
	30	3.827	0.147	3.061	1,370
GRU	6	3.864	0.149	3.091	1,760
	12	3.699	0.137	2.959	980
	18	3.566	0.127	2.853	1,020
	30	3.379	0.114	2.693	741
Bi-GRU	6	4.718	0.223	3.774	5,520
	12	3.619	0.131	2.895	3,440
	18	3.483	0.121	2.807	1,670
	30	3.338	0.111	2.610	4,450
1D-CNN	6	4.411	0.195	3.529	588
	12	3.965	0.157	3.112	2,101
	18	3.844	0.148	3.075	2,831
	30	3.825	0.146	2.968	1,340

To validate our choice of smaller windows techniques, we compared in Table 7.2 our selected model (GRU, window size 30) against a state-of-the-art LSTM-based solution from the literature, which requires a fixed starting reference. While our model achieves slightly worse accuracy (RMSE = 3.38% vs 1.81%), it significantly outperforms in terms of size and number of parameters. Our GRU model requires less than one-fifth of the memory (741 KB vs 3,923 KB) and over five times fewer parameters (188K vs nearly 1M), making it better suitable for edge deployment than the state-of-the-art.

Table 7.2: Comparison with state-of-the-art.

Fixed reference	**Model**	**Input**	**RMSE (%)**	**MAE (%)**	**Size (KB)**	**Parameters**
Yes ([241])	LSTM	278, 3	1.81	0.96	3,923	996,993
No (our)	GRU	30, 3	3.38	2.69	741	188,449

To validate the feasibility of deployment on edge AI hardware, we synthesized the GRU (window 30) model on a Xilinx ZCU104 FPGA board. The model was reimplemented from scratch in C++, without relying on any high-level synthesis from Python-based frameworks. We then used Xilinx Vivado after the post-implementation phase (i.e., after placement and routing) to extract latency and power values from the timing and power reports. Therefore, these quantities were not measured on the physical board, but obtained as implementation-level estimates from the final mapped design, which are generally considered close to the actual hardware behavior. To ensure a fair comparison, both GRU and LSTM models were implemented without applying any architecture-specific optimizations. This allows for an unbiased evaluation of their inherent efficiency under the same development conditions. Table 7.3 reports the results in terms of latency, power consumption, energy usage, and resource utilization. Our GRU-based system achieved a latency of 38.76 ms and consumed only 59.70 mJ per inference, confirming its suitability for low-power real-time applications. Notably, the model utilized just 18.73% of Flip-Flops and 37.51% of Look-Up Tables, leaving room for further integration or multi-model deployment. By contrast, the state-of-the-art LSTM, using full window size as input, exhibits 15× higher

latency, 16× greater energy consumption, and increased resource demand, making it less suitable for edge deployment.

Table 7.3: FPGA deployment performance on ZCU104.

Model	**Latency (ms)**	**Power (W)**	**Energy (mJ)**	**FF (%)**	**LUT (%)**	**BRAM (%)**	**DSP (%)**
LSTM ([241])	585.45	1.61	942.58	37.91	46.13	69.74	29.41
GRU (our)	38.76	1.54	59.70	18.73	37.51	73.72	20.72

Overall, these results show that battery state-of-charge estimation represents a suitable and meaningful use case for FPGA-based Edge AI. The selected GRU model achieves a strong trade-off between accuracy and efficiency, reaching RMSE = 3.38% and MAE = 2.69% with a compact footprint of 741 KB, more than 5× smaller than a state-of-the-art LSTM solution while using over 5× fewer parameters (188k vs ~1M).

When deployed on a Xilinx ZCU104 FPGA, the proposed implementation achieves real-time inference with a latency of 38.76 ms and an energy consumption of only 59.70 mJ per inference, compared to 585 ms and 942 mJ for the reference LSTM model.

Resource utilization remains moderate (<19% FFs and <38% LUTs), leaving margin for integration or multi-model deployment.

It is important to clarify that the developed solution corresponds to the hardware IP of the inference model, rather than to a complete end-to-end hw/sw system. In the adopted setup, the processing system (PS) provides the input data to the programmable logic (PL), the PL executes the model, and the output predictions are then returned to the PS. Therefore, the presented results should be interpreted as referring to the hardware acceleration of the model core. This choice was made to ensure a fair comparison with similar works in the literature, where the evaluation is commonly focused on the model accelerator itself rather than on the full application pipeline. Accordingly, the reported resource utilization and timing results include the hardware logic and communication structures required for the

IP integration, but they do not include the software execution on the PS side, nor the corresponding PS-side energy consumption. In a complete real-time deployment, the full system would also include input preparation, data reordering, buffering, and transfer management between PS and PL. In that case, an optimized hardware (hw)/software (sw) pipeline should be designed according to the input data rate, AXI transfer behavior, and application-level timing requirements. For this reason, the current results should be regarded as model-IP results, while the design of the full end-to-end pipeline is left as a subsequent system-level integration step.

Beyond the specific SOC estimation task, these results demonstrate that carefully selected time-series models, combined with low-level FPGA-oriented implementations, can meet stringent constraints on latency, power, and resource usage, supporting the adoption of FPGA-based solutions for safety-critical and energy-constrained Edge AI applications.

7.2 FPGA-Based One-Class Anomaly Detection

Safety-critical systems such as automated driving require not only accurate decision-making models, but also mechanisms to detect and handle abnormal or unexpected operating conditions [242]. In real-world deployments, machine learning components may be exposed to inputs that deviate from the data distribution observed during training, commonly referred to as out-of-distribution (OOD) inputs [243]. These situations may arise due to sensor noise, unexpected environmental conditions, hardware faults, or the vehicle operating outside its intended Operational Design Domain (ODD).

Failing to detect OOD conditions can lead to unreliable or unsafe behavior, particularly in autonomous and semi-autonomous driving systems where machine learning models are directly involved in control or decision-making loops. For this reason, recent research has increasingly focused on augmenting learning-based systems with explicit monitoring components capable of identifying anomalous inputs and triggering appropriate safety responses [244],[245].

In this context, anomaly and OOD detection represent a natural complement to the

time-series–based predictive models discussed in the previous sections. While the battery SOC estimation task addressed in Section 7.1 focuses on accurate regression under normal operating conditions, OOD detection targets the identification of situations in which model predictions should not be trusted. From a deployment perspective, this task is particularly challenging, as monitoring mechanisms must operate continuously, with low latency and limited power consumption, alongside existing decision-making modules.

To investigate these aspects, this section focuses on the FPGA-based implementation of a One-Class Support Vector Machine (OCSVM) for OOD detection in an automated driving scenario. The proposed approach was developed within the Hi-Drive project and integrated with a neural-network–based decision-making module for low-speed maneuvering. The goal is to assess whether a fully hardware-accelerated monitoring system can meet the strict requirements of automotive embedded platforms in terms of latency, resource usage, and energy efficiency.

7.2.1 Methodology

The overall deployable system consists of two machine learning (ML) modules, namely the MLP agent for low-speed maneuvering decision-making and the OCSVM for input monitoring. An early implementation of the system is described in [246]. The MLP model has an input layer of 340 float32 numbers, two hidden layers with 512 neural units, and a final output layer of size 4 (corresponding to the 4 driving actions: throttle, brake, steering, and reverse). In addition to the weights, each layer includes a bias of 512 for the first and second hidden layers, and 4 for the output layer. This requires more than 400k multiply-accumulate (MACC) operations per inference. Therefore, a significant amount of storage is needed to allow the model's computation, besides the amount needed for static memorization of the NN parameters.

Parts of this chapter will be published in the conference proceedings of Società Italiana di Elettronica (SIE) 2026.
The material has been adapted and extended to fit the structure and scope of this thesis.

The OCSVM model was trained using a dataset containing in-distribution (ID) samples. The model learns a decision boundary around normal data, that is later used in inference to identify OOD data. This trained model contains 545 support vectors, each one with 305 features, 545 dual coefficients, and it uses Radial Basis Function (RBF) kernel, with a gamma value of 0.003194. These values represent the main memory requirement for the FPGA implementation.

While optimizations were implemented for both models, in the following we focus on the OCSVM (i.e., system robustness enhancement), as this is novel in literature.

The experiment exploited Vitis HLS to convert the C++ code into RTL design and Vivado Design Suite for later synthesis. To ensure fully hardware-accelerator design, the entire design is mapped to the FPGA fabric (i.e., PL), without relying on the PS or external memories. This requires that all the resources required by the models are available within the PL of the chosen device. For this, the XCZU7EV-2FFVC1156 (ZCU104) board is suitable for this implementation due to its optimal resources (i.e., 504K logic cells, 1728 DSP), with on-chip memory architecture of up to 38MB, including 312 Block-RAMs (BRAM) with 36Kb each, and distributed memory through Look-Up Tables (LUTs). Floating point numbers are stored as 32 bit fixed-point.

In this work, we focused on a critical design parameter that can be adjusted through HLS [247], namely the clock target period with related complexity of combinational logic path per clock cycle. The clock period affects the operating frequency and consequently the logic partitioning by the synthesis and implementation tools [248]. A fast clock may require that a long combinational logic path is segmented into smaller parts, as computation units should be completed in each clock cycle. The output of each logic path between two flip-flops (FFs) should be produced before the next rising clock edge, otherwise, the FF may latch an incorrect value. Storing intermediate values increases both the resources needed and the number of pipeline stages, and consequently the cycle count. The experiments were done using three clock periods targets: 5ns, 10ns, and 15ns, to explore the impact on the overall performance.

The hardware accelerator was developed using the Xilinx Vitis HLS tool. In order to improve throughput, several HLS optimizations were applied during the design-space exploration. In particular, internal buffers were completely partitioned by means of the directive `#pragma HLS ARRAY_PARTITION complete`, so as to remove BRAM port limitations and enable parallel access to all elements when required by the pipelined computation. The main processing loops were optimized through `#pragma HLS PIPELINE II=1`, with the goal of producing one output per clock cycle whenever possible, while selected loops were further parallelized through `#pragma HLS UNROLL`, with the unrolling factor chosen according to the specific loop in order to balance performance and resource utilization. Moreover, `#pragma HLS INLINE off` was used in selected functions to avoid excessively aggressive flattening and preserve a more controllable hardware structure during synthesis.

As for system integration, AXI interfaces were adopted according to the standard PS-PL communication scheme. In particular, `m_axi` interfaces were used for input and output data transfers, while `s_axilite` interfaces were used for control signals and parameter configuration. This allowed the generated IP to be integrated into the PS-PL system, enabling data transfers between the DDR memory in the PS and the accelerator implemented in the PL.

7.2.2 Experimental Results

7.2.2.1 Clock Target and Combinational Logic Path

The clock target affects the maximum allowable delay for combinational logic and influences how Vivado performs pipelining during synthesis. Vivado performs physical implementation of the hardware design generated by the HLS tool, which is the process of determining and connecting the resources on the FPGA (Placing and Routing), by partitioning its logic into pipeline stages to meet the timing requirements defined by the clock period. For instance, setting the clock target to 10 ns requires that each combinational path (i.e., uninterrupted logic that processes data between two registers) does not exceed the specified time length.

Conversely, to fit a 5 ns clock design, the synthesis tool must split any longer path, which explains the increased number of cycles in this case. Under a certain threshold, the clock period may lead to time violations, exceeding the hardware capabilities. For OCSVM, the threshold was 3 ns, determined by the Worst Negative Slack (WNS) timing report metrics [249]. Table 7.4 synthetizes the terms of the performance/resource trade-offs at the three clock targets considered. The latency and power values reported in Table 7.4 were obtained from the Vivado post-implementation timing and power reports, after placement and routing, and should therefore be interpreted as implementation-level estimates rather than as direct on-board measurements. Since these reports are generated from the final RTL/netlist mapped onto the target FPGA, they are generally considered accurate and close to the real hardware behavior, although small deviations may still occur under actual board-level operating conditions.

Table 7.4: Performance, resource utilization and power consumption at different clocks.

Clock Target [ns]	**Clock cycles**	**Latency [ms]**	**FFs**	**LUTs**	**BRAM**	**FFs + LUTs Power [mW]**	**BRAM Power [mW]**
5	674,166	3.37	733	898	183	24	91.5
10	337,356	3.373	473	841	183	8	43.92
15	336,811	5	539	784	183	5	29.28

As shown in Table 7.4, running the OCSVM model at a clock target of 5 ns requires approximately 674,166 cycles, with a latency of 3.37 ms. When synthesizing the same design with a slower clock target of 10 ns, each sample inference needed 337,356 cycles, with an almost identical latency of 3.373 ms. A more relaxed clock period of 15 ns was evaluated as well, which resulted in 336,811 clock cycles close to that at 10 ns but with an inference time of about 5 ms. Considering different clock targets, it was observed that the number of BRAMs remains the same, but the power consumption for each BRAM (BRAM power/BRAM) decreases from 0.5 mW at 5 ns to: 0.24 mW at 10 ns, and 0.16 mW at 15 ns (Table 7.4). At 15 ns clock, Vivado was able to meet the timing constraints with less overall logic resources. Particularly, less LUTs were used

comparing to the other two clock targets, while the FFs are more utilized than the 10ns clock to balance the placement and routing. The dynamic power consumption for the utilized LUTs and FFs is reported in Table 7.4 and shows an increase with the tighter clock target. This increase is not only attributed to greater resource utilization, but we also measured that the dynamic power per individual FF increased proportionally with the clock frequency (2.52 mW @5 ns, 1.26 mW @10 ns, and 0.84 mW @15 ns). The hardware synthesis for the 5 ns and 10 ns clock targets led to almost identical throughput (296 inferences/sec) against the 200 inferences/sec for the 15 ns clock target. This comes with a trade-off with dynamic power (230 mW @5 ns, 111 mW @10 ns, 72 mW @15 ns), as shown in Figure 7.1.

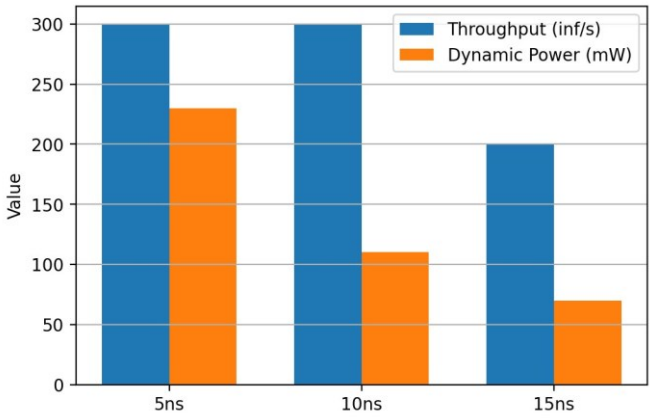


Figure 7.1:Throughput vs. Dynamic Power at Different Clock Periods.

Considering a 10 ns clock target, the overall design, that includes both the OCSVM and the MLP models, achieved an inference up to 82.2 inf/sec, which is suited for real-time performance, with energy consumption 1.3mJ and average power 106mW.

7.2.2.2 Assessing compiler optimization

Table 7.5 shows the performance improvement achieved through the Vitis compiler's pragma optimizations (essentially: pipelining and unrolling) on a 10 ns clock design. Also in this case, the reported latency and power values were derived from Vivado post-implementation reports and not from direct physical measurements on the board. Table 7.6 highlights how latency reduction is

obtained through a significant change in resource allocation.

Table 7.5: ZCU104 Performance.

Model	Without Pragmas			With Pragmas		
	Time [ms]	Power [mW]	Energy [mJ]	Time [ms]	Power [mW]	Energy [mJ]
OCSVM	3.37	111	0.37	0.007	1,479	0.01
MLP	8.7	107	0.93	1.5	735	1.1

Table 7.6: Resource Utilization on ZCU104.

Resource	OCSVM		MLP	
	Without Pragmas	With Pragmas	Without Pragmas	With Pragmas
LUT	0.37%	13.27%	36.84%	56.82%
FF	0.10%	4.94%	0.08%	12.76%
BRAM	58.65%	98.72%	37.50%	37.66%
DSP	2.03%	72.40%	0.35%	20.20%

For the OCSVM particularly, the reduction is significant and the change radical. Notably, by applying aggressive pipeline design to meet tighter timing constraints for the OCSVM, the BRAM usage is almost saturating the available units.

These optimizations led also to a surge in the utilization percentage of FFs, LUTs, and DSPs. We thus see that optimizations implied using more, and more specialized resources from the board. This results in one order of magnitude increase in power consumption and three orders of magnitude decrease in latency, leading to a drop in energy consumption of one order of magnitude per inference.

Pragma optimization looks less beneficial for MLP. Latency decreases 5.8x, but power increases 6.9x and energy 1.2x. The BRAM utilization remains unaffected, and the increase in LUT and, overall, DSP is significant, but not dramatic as in the OCSVM, while the increase in FFs is more significant. Considering the extensive use of MAC operations by MLP, we argue that utilization of a specialized Deep Learning Processing Unit (DPU) could be more beneficial for this model.

For the overall system (OCSVM + MLP), the unoptimized version achieves a throughput of 82.2 inf/sec with energy consumption 1.3 mJ and average pow-er 108 mW, while pragma optimization speeds up to 667 inf/sec, with an energy cost

of 1.1 mJ and an average power of 733 mW. This demonstrates real-time capability, at relatively low power. Moreover, only marginally does the addition of OOD input monitoring impact the performance of the vehicular decision-making system.

Overall, the presented results show that FPGA-based one-class anomaly detection can be integrated into safety-critical automotive systems with limited impact on latency and energy consumption. The OCSVM hardware implementation achieves inference latencies in the range of 3.3-5 ms depending on the clock target, while sustaining throughputs up to 296 inferences/s at 5-10 ns clock periods.
When synthesized at a 10 ns clock target, the complete system (OCSVM + MLP) operates at 82.2 inferences/s with an average power consumption of 106 mW and an energy cost of 1.3 mJ per inference, meeting real-time requirements for low-speed automated driving. Aggressive compiler optimizations further reduce OCSVM latency by almost three orders of magnitude (from 3.37 ms to 7 μs), at the cost of increased resource utilization and power, highlighting clear trade-offs between performance and efficiency.
Compared to software-based execution on general-purpose embedded platforms, the FPGA implementation provides orders-of-magnitude reductions in latency and energy consumption, while enabling fully hardware-accelerated OOD monitoring alongside a neural-network-based decision-making module. These results demonstrate that continuous anomaly detection can be realized as a self-contained hardware component, reinforcing the feasibility of FPGA-based architectures for robust and dependable Edge AI in automated driving applications.

CHAPTER 8

8. Conclusion and Future Directions

This thesis addressed the design of an end-to-end, deployable workflow for time-series data in IoT and Edge AI applications, covering the entire process from data collection and management to model training and real-time deployment on embedded hardware under realistic constraints on memory, latency, power consumption, and robustness.

Rather than treating ML models as isolated solutions, this work adopted a perspective in which algorithms are evaluated as components of embedded systems operating under strict physical and architectural constraints. Across all chapters, model accuracy was therefore considered alongside execution latency, memory footprint, energy consumption, and hardware compatibility, reflecting the requirements of real-world deployment scenarios.

A first contribution concerns data management. By extending the Measurify framework with flexible CSV-based workflows and semantic modeling, the thesis demonstrated how heterogeneous time-series datasets can be ingested, organized, and reused across domains while preserving contextual information and without requiring user-side modifications. This layer enabled reproducible experimentation and consistent dataset handling, forming the basis for all subsequent machine learning developments.

Building on this infrastructure, the thesis investigated lightweight deep learning models for time-series analysis across multiple domains.

In human and sports activity recognition, the proposed edge-to-cloud workflow enabled easy and efficient dataset creation and real-time inference on low-cost embedded devices under extremely limited resource constraints.

In structural health monitoring, state-of-the-art models, MiniRocket and WaveNet,

achieved high accuracy on long raw vibration sequences while remaining deployable on edge platforms.

In driver distraction detection, deep learning models specifically designed for time-series processing proved effective under realistic evaluation protocols, highlighting the importance of split-by-user validation and short time windows for real-time operation.

Across all application domains, the thesis consistently emphasized deployability as a primary design objective, while at the same time advancing the state-of-the-art through improved accuracy, reduced model size, and more realistic evaluation settings. Rather than treating deployment as a secondary step, each study explicitly quantified inference latency, memory footprint, and energy consumption on target embedded platforms, demonstrating how architectural choices directly affect practical feasibility.

This experimental evidence highlights concrete trade-offs between model complexity and deployability: compact time-series models can achieve competitive or state-of-the-art performance while meeting real-time constraints on microcontrollers, single-board computers, and embedded systems.

The FPGA-based case studies further extend these findings beyond software-based deployment, showing that hardware-level implementations can reduce inference latency by more than an order of magnitude and significantly improve energy efficiency for selected tasks such as battery state-of-charge estimation and anomaly detection.

Beyond the specific application domains and experimental results, the main impact of this thesis lies in demonstrating, through quantitative evidence, that advanced time-series intelligence can be realistically deployed on resource-constrained hardware without sacrificing reliability or real-time performance. By systematically measuring latency, memory footprint, and energy consumption alongside accuracy, this work contributes concrete benchmarks and design guidelines for engineers developing Edge AI systems in safety-critical and industrial contexts.

The results presented across multiple domains show that careful model selection, architecture design, and hardware-aware implementation choices can bridge the gap between state-of-the-art machine learning research and practical embedded deployment. In this sense, the thesis provides not only new application-specific results, but also experimental evidence that supports a more deployable and sustainable approach to Edge AI, where performance, efficiency, and robustness are jointly considered from the early stages of system design.

8.1 Future works

The work presented in this thesis opens several directions for future research in deployable Edge AI systems. First, the demonstrated feasibility of lightweight time-series models under strict resource constraints motivates further investigation of learning paradigms that reduce reliance on labeled data, such as self-supervised and unsupervised approaches, particularly in safety-critical domains where annotation is costly or ambiguous.

From a hardware and deployment perspective, the results obtained on microcontrollers and FPGA platforms highlight the potential of advanced quantization techniques, mixed-precision arithmetic, and hardware-aware neural architecture design to further improve energy efficiency and real-time performance. Building on the FPGA prototypes presented in this thesis, future work can explore more automated and accessible toolchains for translating trained models into efficient hardware implementations.

At the system level, the integration between data management infrastructures and deployment toolchains demonstrated in this work provides a foundation for automated dataset-to-hardware pipelines, enabling reproducible experimentation at scale. Finally, extending the presented methodologies to long-term, real-world deployments will allow the evaluation of system reliability, maintenance costs, and user acceptance over time, supporting the transition of Edge AI technologies from controlled experiments to operational environments.

Bibliography

[1] D. Wei *et al.*, "Dataflow Management in the Internet of Things: Sensing, Control, and Security," *Tsinghua Science and Technology*, vol. 26, no. 6, pp. 918–930, Dec. 2021, doi: 10.26599/TST.2021.9010029.

[2] W. B. Qaim *et al.*, "Towards Energy Efficiency in the Internet of Wearable Things: A Systematic Review," *IEEE Access*, vol. 8, pp. 175412–175435, 2020, doi: 10.1109/ACCESS.2020.3025270.

[3] K. M. Alam, M. Saini, and A. E. Saddik, "Toward Social Internet of Vehicles: Concept, Architecture, and Applications," *IEEE Access*, vol. 3, pp. 343–357, 2015, doi: 10.1109/ACCESS.2015.2416657.

[4] M. R. Bashir, A. Q. Gill, G. Beydoun, and B. Mccusker, "Big Data Management and Analytics Metamodel for IoT-Enabled Smart Buildings," *IEEE Access*, vol. 8, pp. 169740–169758, 2020, doi: 10.1109/ACCESS.2020.3024066.

[5] A. A. Zakaria, T. Amr, and A. A. Ragheb, "IoT in Smart Urban Planning: A Comprehensive Review of Applications, Developments, and Engineering Perspectives," *IEEE Access*, vol. 13, pp. 135316–135335, 2025, doi: 10.1109/ACCESS.2025.3594019.

[6] Z. Han, J. Zhao, H. Leung, K. F. Ma, and W. Wang, "A Review of Deep Learning Models for Time Series Prediction," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 7833–7848, Mar. 2021, doi: 10.1109/JSEN.2019.2923982.

[7] R. Lohiya and A. Thakkar, "Application Domains, Evaluation Data Sets, and Research Challenges of IoT: A Systematic Review," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 8774–8798, Jun. 2021, doi: 10.1109/JIOT.2020.3048439.

[8] "Measurify," Measurify. Accessed: Jan. 14, 2026. [Online]. Available: http://measurify.info/

[9] "GitHub - measurify/server: Measurify RESTful API." Accessed: Jan. 14, 2026. [Online]. Available: https://github.com/measurify/server

[10] M. Fresta, A. Capello, F. Bellotti, L. Lazzaroni, M. Cossu, and R. Berta, "Supporting a .csv-based Workflow in MongoDB for Data Analysts," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, Jun. 2023, pp. 1–4. doi: 10.1109/ISIE51358.2023.10228044.

[11] "Hi-Drive Deployment of Higher Automation." Accessed: Jan. 14, 2026. [Online]. Available: https://www.hi-drive.eu/

[12] R. Kanjilal, M. Furkan Kucuk, and I. Uysal, "Human Activity Recognition: A Review of RFID and Wearable Sensor Technologies Powered by AI," *IEEE Journal of Radio Frequency Identification*, vol. 9, pp. 180–199, 2025, doi: 10.1109/JRFID.2025.3561345.

[13] R. Zinno, S. S. Haghshenas, G. Guido, and A. VItale, "Artificial Intelligence and Structural Health Monitoring of Bridges: A Review of the State-of-the-Art," *IEEE Access*, vol. 10, pp. 88058–88078, 2022, doi: 10.1109/ACCESS.2022.3199443.

[14] A. Kashevnik, R. Shchedrin, C. Kaiser, and A. Stocker, "Driver Distraction Detection Methods: A Literature Review and Framework," *IEEE Access*, vol. 9, pp. 60063–60076, 2021, doi: 10.1109/ACCESS.2021.3073599.

[15] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019, doi: 10.1109/ACCESS.2018.2890150.

[16] T. Mladenova and I. Valova, "Performance Study of MySQL and MongoDB for

IoT Data Processing and Storage," in *2022 International Conference Automatics and Informatics (ICAI)*, Oct. 2022, pp. 60–63. doi: 10.1109/ICAI55857.2022.9960134.

[17] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments," *IEEE Access*, vol. 8, pp. 110656–110668, 2020, doi: 10.1109/ACCESS.2020.3002164.

[18] L. G. Wiseso, M. Imrona, and A. Alamsyah, "Performance Analysis of Neo4j, MongoDB, and PostgreSQL on 2019 National Election Big Data Management Database," in *2020 6th International Conference on Science in Information Technology (ICSITech)*, Oct. 2020, pp. 91–96. doi: 10.1109/ICSITech49800.2020.9392041.

[19] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld, "Updating XML," *SIGMOD Rec.*, vol. 30, no. 2, pp. 413–424, May 2001, doi: 10.1145/376284.375720.

[20] "Extensible Markup Language (XML) 1.0." Accessed: Apr. 06, 2023. [Online]. Available: https://www.w3.org/TR/1998/REC-xml-19980210.html

[21] "JSON." Accessed: Apr. 06, 2023. [Online]. Available: https://www.json.org/json-en.html

[22] "The HDF5® Library & File Format," The HDF Group. Accessed: Apr. 06, 2023. [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/

[23] B. B. P. Maurya, A. Ray, A. Upadhyay, B. Gour, and A. U. Khan, "Recursive Stock Price Prediction With Machine Learning And Web Scrapping For Specified Time Period," in *2019 Sixteenth International Conference on Wireless and Optical Communication Networks (WOCN)*, Dec. 2019, pp. 1–3. doi: 10.1109/WOCN45266.2019.8995080.

[24] F. Bellotti *et al.*, "Managing Big Data for Addressing Research Questions in a Collaborative Project on Automated Driving Impact Assessment," *Sensors*, vol. 20, no. 23, Art. no. 23, Jan. 2020, doi: 10.3390/s20236773.

[25] M. S. Raja, M. Anurag, Ch. P. Reddy, and N. R. Sirisala, "Machine Learning Based Heart Disease Prediction System," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2021, pp. 1–5. doi: 10.1109/ICCCI50826.2021.9402653.

[26] J. Mitlöhner, S. Neumaier, J. Umbrich, and A. Polleres, "Characteristics of Open Data CSV Files," in *2016 2nd International Conference on Open and Big Data (OBD)*, Aug. 2016, pp. 72–79. doi: 10.1109/OBD.2016.18.

[27] G. J. J. van den Burg, A. Nazábal, and C. Sutton, "Wrangling messy CSV files by detecting row and type patterns," *Data Min Knowl Disc*, vol. 33, no. 6, pp. 1799–1820, Nov. 2019, doi: 10.1007/s10618-019-00646-y.

[28] R. Berta, A. Kobeissi, F. Bellotti, and A. De Gloria, "Atmosphere, an Open Source Measurement-Oriented Data Framework for IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1927–1936, Mar. 2021, doi: 10.1109/TII.2020.2994414.

[29] "Home - OpenIoT." Accessed: Jan. 14, 2026. [Online]. Available: https://openiot.org/

[30] "Introduction - WebThings Documentation." Accessed: Jan. 14, 2026. [Online]. Available: https://webthings.io/docs/wot/introduction/

[31] "Introduction," BIG IoT. Accessed: Jan. 14, 2026. [Online]. Available: https://big-iot.github.io/

[32] "Gateway IoT sicuro, dispositivo gateway IoT - AWS IoT Core - AWS," Amazon

Web Services, Inc. Accessed: Jan. 14, 2026. [Online]. Available: https://aws.amazon.com/it/iot-core/
[33] "Servizi di cloud computing | Microsoft Azure." Accessed: Jan. 14, 2026. [Online]. Available: https://azure.microsoft.com/it-it
[34] S. Kumar, P. Singh, and A. Singh, "A Review of Optimized Computational Strategies for IoT: Cloud, Fog, and Edge Computing Approaches," in *2025 5th International Conference on Pervasive Computing and Social Networking (ICPCSN)*, May 2025, pp. 925–930. doi: 10.1109/ICPCSN65854.2025.11035529.
[35] J. Pan and J. McElhannon, "Future Edge Cloud and Edge Computing for Internet of Things Applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, Feb. 2018, doi: 10.1109/JIOT.2017.2767608.
[36] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative Cloud and Edge Computing for Latency Minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, May 2019, doi: 10.1109/TVT.2019.2904244.
[37] K. Cao, S. Hu, Y. Shi, A. W. Colombo, S. Karnouskos, and X. Li, "A Survey on Edge and Edge-Cloud Computing Assisted Cyber-Physical Systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7806–7819, Nov. 2021, doi: 10.1109/TII.2021.3073066.
[38] M. De Donno, K. Tange, and N. Dragoni, "Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog," *IEEE Access*, vol. 7, pp. 150936–150948, 2019, doi: 10.1109/ACCESS.2019.2947652.
[39] J. Yao *et al.*, "Edge-Cloud Polarization and Collaboration: A Comprehensive Survey for AI," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6866–6886, Jul. 2023, doi: 10.1109/TKDE.2022.3178211.
[40] P. Subramani, P. M. Parekh, V. Balasubramaniam, G. Tavva, G. Kumar Gupta, and A. Jain, "Energy-Efficient Semiconductor Architectures for Edge AI and IoT Systems," in *2025 International Conference on Intelligent and Secure Engineering Solutions (CISES)*, Aug. 2025, pp. 1159–1163. doi: 10.1109/CISES66934.2025.11265118.
[41] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An Overview on Edge Computing Research," *IEEE Access*, vol. 8, pp. 85714–85728, 2020, doi: 10.1109/ACCESS.2020.2991734.
[42] C. Yahyati *et al.*, "A Systematic Review of State-of-the-Art TinyML Applications in Healthcare, Education, and Transportation," *IEEE Access*, vol. 13, pp. 204513–204562, 2025, doi: 10.1109/ACCESS.2025.3633575.
[43] E. A. Adeniyi, S. A. Ajagbe, O. A. Oki, A. O. Adebayo, and O. Olasupo, "Application of Machine Learning Algorithm in Cloud-to-edge Computing: Analysis and Limitations," in *2023 IEEE AFRICON*, Sep. 2023, pp. 1–6. doi: 10.1109/AFRICON55910.2023.10293346.
[44] M. I. Patel, S. Suthar, and J. Thakar, "Survey on Image Compression using Machine Learning and Deep Learning," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, May 2019, pp. 1103–1105. doi: 10.1109/ICCS45141.2019.9065473.
[45] D. G. Pinto and S. B. Mane, "Energy Efficient Implementation of CNNs Using Pruning: Techniques and Insights," in *2025 International Conference on Information, Implementation, and Innovation in Technology (I2ITCON)*, Jul. 2025, pp. 1–5. doi: 10.1109/I2ITCON65200.2025.11210640.
[46] C. J. Schaefer, P. Taheri, M. Horeni, and S. Joshi, "The Hardware Impact of Quantization and Pruning for Weights in Spiking Neural Networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 5, pp. 1789–

1793, May 2023, doi: 10.1109/TCSII.2023.3260701.
[47] W. Duan, Z. Liu, C. Jia, S. Wang, S. Ma, and W. Gao, “Differential Weight Quantization for Multi-Model Compression,” *IEEE Transactions on Multimedia*, vol. 25, pp. 6397–6410, 2023, doi: 10.1109/TMM.2022.3208530.
[48] M. Singh, L. Mohanty, N. Gupta, Y. Bansal, and S. Garg, “Q-Net Compressor: Adaptive Quantization for Deep Learning on Resource-Constrained Devices,” in *2024 International Conference on Computing, Sciences and Communications (ICCSC)*, Oct. 2024, pp. 1–6. doi: 10.1109/ICCSC62048.2024.10830393.
[49] K. Zhang and G. Liu, “Layer Pruning for Obtaining Shallower ResNets,” *IEEE Signal Processing Letters*, vol. 29, pp. 1172–1176, 2022, doi: 10.1109/LSP.2022.3171128.
[50] J. Hu *et al.*, “A Dynamic Pruning Method on Multiple Sparse Structures in Deep Neural Networks,” *IEEE Access*, vol. 11, pp. 38448–38457, 2023, doi: 10.1109/ACCESS.2023.3267469.
[51] R. Sayed, H. Azmi, H. Shawkey, A. H. Khalil, and M. Refky, “A Systematic Literature Review on Binary Neural Networks,” *IEEE Access*, vol. 11, pp. 27546–27578, 2023, doi: 10.1109/ACCESS.2023.3258360.
[52] Y.-M. Qian and X. Xiang, “Binary Neural Networks for Speech Recognition,” *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 5, pp. 701–715, May 2019, doi: 10.1631/FITEE.1800469.
[53] tensorflow, “tensorflow/tensorflow/lite at master · tensorflow/tensorflow,” GitHub. Accessed: Jan. 14, 2026. [Online]. Available: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite
[54] R. David *et al.*, “TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems,” Mar. 13, 2021, *arXiv*: arXiv:2010.08678. doi: 10.48550/arXiv.2010.08678.
[55] *tensorflow/tflite-micro*. (Jan. 14, 2026). C++. tensorflow. Accessed: Jan. 14, 2026. [Online]. Available: https://github.com/tensorflow/tflite-micro
[56] “Edge Impulse - The Leading Edge AI Platform.” Accessed: Jan. 14, 2026. [Online]. Available: https://www.edgeimpulse.com
[57] A. Biscontini, E. Popovici, and A. Temko, “Machine Learning for FPGA Electronic Design Automation,” *IEEE Access*, vol. 12, pp. 182640–182662, 2024, doi: 10.1109/ACCESS.2024.3511345.
[58] F. Spagnolo, P. Corsonello, F. Frustaci, and S. Perri, “Efficient implementation of signed multipliers on FPGAs,” *Computers and Electrical Engineering*, vol. 116, p. 109217, May 2024, doi: 10.1016/j.compeleceng.2024.109217.
[59] J. Borrego-Carazo, D. Castells-Rufas, E. Biempica, and J. Carrabina, “Resource-Constrained Machine Learning for ADAS: A Systematic Review,” *IEEE Access*, vol. 8, pp. 40573–40598, 2020, doi: 10.1109/ACCESS.2020.2976513.
[60] J. Lee and J. Park, “Edge AI for Structural Health Monitoring: An FPGA-Based Approach on IoT Sensor Nodes,” in *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, Jun. 2025, pp. 515–522. doi: 10.1109/DCOSS-IoT65416.2025.00085.
[61] M. Milton, A. Benigni, and A. Monti, “Real-Time Multi-FPGA Simulation of Energy Conversion Systems,” *IEEE Transactions on Energy Conversion*, vol. 34, no. 4, pp. 2198–2208, Dec. 2019, doi: 10.1109/TEC.2019.2938811.
[62] Á. Hernandez *et al.*, “FPGA-Based Track Circuit for Railways Using Transmission Encoding,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 437–448, Jun. 2012, doi: 10.1109/TITS.2011.2170976.
[63] A. Hachem, I. Belalchheb, Y. Moline, F. Carrel, and G. Corre, “FPGA

Implementation of MLP, 1D-CNN and TTTratio algorithms for Neutron/Gamma-ray Discrimination using Plastic Scintillator," in *2023 IEEE Nordic Circuits and Systems Conference (NorCAS)*, Oct. 2023, pp. 1–7. doi: 10.1109/NorCAS58970.2023.10305446.

[64] J. Bartels, A. Hagihara, L. Minati, K. K. Tokgoz, and H. Ito, "An Integer-Only Resource-Minimized RNN on FPGA for Low-Frequency Sensors in Edge-AI," *IEEE Sensors Journal*, vol. 23, no. 15, pp. 17784–17793, Aug. 2023, doi: 10.1109/JSEN.2023.3286580.

[65] Z. S. Zaghloul and N. Elsayed, "The FPGA Hardware Implementation of the Gated Recurrent Unit Architecture," in *SoutheastCon 2021*, Mar. 2021, pp. 1–5. doi: 10.1109/SoutheastCon45413.2021.9401819.

[66] "AMD Vitis™ HLS," AMD. Accessed: Jan. 15, 2026. [Online]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

[67] "AMD Vivado™ Design Suite," AMD. Accessed: Jan. 15, 2026. [Online]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html

[68] C. Xu *et al.*, "The Case for FPGA-Based Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2610–2619, Jul. 2022, doi: 10.1109/TMC.2020.3041781.

[69] W. Qi, X. Xu, K. Qian, B. W. Schuller, G. Fortino, and A. Aliverti, "A Review of AIoT-Based Human Activity Recognition: From Application to Technique," *IEEE Journal of Biomedical and Health Informatics*, vol. 29, no. 4, pp. 2425–2438, Apr. 2025, doi: 10.1109/JBHI.2024.3406737.

[70] Z. H. Warsi, S. M. Irshad, F. Khan, M. A. Shahbaz, M. Junaid, and S. U. Amin, "Sensors for Structural Health Monitoring: A Review," in *2019 Second International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)*, Nov. 2019, pp. 1–6. doi: 10.1109/INTELLECT47034.2019.8955453.

[71] M. Gyllenhammar, G. Rodrigues de Campos, and M. Törngren, "The Road to Safe Automated Driving Systems: A Review of Methods Providing Safety Evidence," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 4, pp. 4315–4345, Apr. 2025, doi: 10.1109/TITS.2025.3532684.

[72] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines," *IEEE Access*, vol. 9, pp. 120043–120065, 2021, doi: 10.1109/ACCESS.2021.3107975.

[73] R. Berta, A. Kobeissi, F. Bellotti, and A. De Gloria, "Atmosphere, an Open Source Measurement-Oriented Data Framework for IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1927–1936, Mar. 2021, doi: 10.1109/TII.2020.2994414.

[74] "Centraline qualità dell'aria (misurazioni giornaliere)." Accessed: Oct. 01, 2025. [Online]. Available: https://opendata.comune.bologna.it/explore/dataset/centraline-qualita-aria/

[75] "The Data Catalog Platform," data.world. Accessed: Oct. 01, 2025. [Online]. Available: https://data.world/

[76] L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang, "A review on the key issues for lithium-ion battery management in electric vehicles," *Journal of Power Sources*, vol. 226, pp. 272–288, Mar. 2013, doi: 10.1016/j.jpowsour.2012.10.060.

[77] R. Xiong, J. Cao, Q. Yu, H. He, and F. Sun, "Critical Review on the Battery State of Charge Estimation Methods for Electric Vehicles," *IEEE Access*, vol. 6, pp.

1832–1843, 2018, doi: 10.1109/ACCESS.2017.2780258.
[78] "Optimizing Precision Cell Measurement Accuracy in Automotive Battery Management Systems." Accessed: Jan. 15, 2026. [Online]. Available: https://www.powersystemsdesign.com/articles/optimizing-precision-cell-measurement-accuracy-in-automotive-battery-management-systems/145/15772
[79] S. Gallinaro, "Higher Reliability, Safety, and 30% Longer Lifetime with Advanced Battery Management in Healthcare Energy Storage Systems".
[80] K. P. Schneider *et al.*, "Analytic Considerations and Design Basis for the IEEE Distribution Test Feeders," *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3181–3188, May 2018, doi: 10.1109/TPWRS.2017.2760011.
[81] R. Wagle, P. Sharma, C. Sharma, and M. Amin, "Optimal power flow based coordinated reactive and active power control to mitigate voltage violations in smart inverter enriched distribution network," *International Journal of Green Energy*, vol. 21, no. 2, pp. 359–375, Jan. 2024, doi: 10.1080/15435075.2023.2196324.
[82] F. Ni, P. H. Nguyen, J. F. G. Cobben, H. E. Van den Brom, and D. Zhao, "Three-phase state estimation in the medium-voltage network with aggregated smart meter data," *International Journal of Electrical Power & Energy Systems*, vol. 98, pp. 463–473, Jun. 2018, doi: 10.1016/j.ijepes.2017.12.033.
[83] Y. Himeur, A. Alsalemi, F. Bensaali, and A. Amira, "Smart non-intrusive appliance identification using a novel local power histogramming descriptor with an improved k-nearest neighbors classifier," *Sustainable Cities and Society*, vol. 67, p. 102764, Apr. 2021, doi: 10.1016/j.scs.2021.102764.
[84] A. Ruano *et al.*, "NILM Techniques for Intelligent Home Energy Management and Ambient Assisted Living: A Review," *Energies*, vol. 12, no. 11, Jun. 2019, doi: 10.3390/en12112203.
[85] G. W. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, Dec. 1992, doi: 10.1109/5.192069.
[86] "Project Jupyter." Accessed: Jan. 15, 2026. [Online]. Available: https://jupyter.org
[87] "Docker Desktop: The #1 Containerization Tool for Developers | Docker." Accessed: Jan. 15, 2026. [Online]. Available: https://www.docker.com/products/docker-desktop/
[88] L. G. Wiseso, M. Imrona, and A. Alamsyah, "Performance Analysis of Neo4j, MongoDB, and PostgreSQL on 2019 National Election Big Data Management Database," in *2020 6th International Conference on Science in Information Technology (ICSITech)*, Oct. 2020, pp. 91–96. doi: 10.1109/ICSITech49800.2020.9392041.
[89] A. Krechowicz, S. Deniziak, and G. Łukawski, "Highly Scalable Distributed Architecture for NoSQL Datastore Supporting Strong Consistency," *IEEE Access*, vol. 9, pp. 69027–69043, 2021, doi: 10.1109/ACCESS.2021.3077680.
[90] "React – A JavaScript library for building user interfaces." Accessed: Jan. 15, 2026. [Online]. Available: https://legacy.reactjs.org/
[91] C. Macrae, *Vue.js: Up and Running*. Accessed: Jan. 15, 2026. [Online]. Available: https://www.oreilly.com/library/view/vue-js-up-and/9781491997239/
[92] "Chart.js." Accessed: Jan. 15, 2026. [Online]. Available: https://www.chartjs.org/
[93] F. Bellotti *et al.*, "Managing Big Data for Addressing Research Questions in a Collaborative Project on Automated Driving Impact Assessment," *Sensors*, vol. 20, no. 23, Nov. 2020, doi: 10.3390/s20236773.
[94] "Testing Frameworks for Javascript | Write, Run, Debug | Cypress." Accessed: Jan. 15, 2026. [Online]. Available: https://www.cypress.io/

[95] “Servizi di cloud computing: Amazon Web Services (AWS),” Amazon Web Services, Inc. Accessed: Jan. 15, 2026. [Online]. Available: https://aws.amazon.com/it/
[96] “Node.js — Run JavaScript Everywhere.” Accessed: Jan. 15, 2026. [Online]. Available: https://nodejs.org/en
[97] “MongoDB: The World’s Leading Modern Database,” MongoDB. Accessed: Jan. 15, 2026. [Online]. Available: https://www.mongodb.com/
[98] A. Ghibellini, L. Bononi, and M. Di Felice, “Intelligence at the IoT Edge: Activity Recognition with Low-Power Microcontrollers and Convolutional Neural Networks,” in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA: IEEE, Jan. 2022, pp. 707–710. doi: 10.1109/CCNC49033.2022.9700665.
[99] Y. Liu, “Research and Development of GNSS Wearable Device for Sports Performance Monitoring by Example of Soccer Player Analysis∗,” in *Proceedings of the 2022 6th International Conference on Electronic Information Technology and Computer Engineering*, Xiamen China: ACM, Oct. 2022, pp. 901–906. doi: 10.1145/3573428.3573590.
[100] H. Zhang, Z. Zhang, N. Gao, Y. Xiao, Z. Meng, and Z. Li, “Cost-Effective Wearable Indoor Localization and Motion Analysis via the Integration of UWB and IMU,” *Sensors*, vol. 20, no. 2, p. 344, Jan. 2020, doi: 10.3390/s20020344.
[101] M. Farjana, A. B. Fahad, S. E. Alam, and M. M. Islam, “An IoT- and Cloud-Based E-Waste Management System for Resource Reclamation with a Data-Driven Decision-Making Process,” *IoT*, vol. 4, no. 3, Art. no. 3, Sep. 2023, doi: 10.3390/iot4030011.
[102] V. A. Arcobelli *et al.*, “mCrutch: A Novel m-Health Approach Supporting Continuity of Care,” *Sensors*, vol. 23, no. 8, Art. no. 8, Jan. 2023, doi: 10.3390/s23084151.
[103] L. Sudha Kumari and A. Z. Kouzani, “A Miniaturized Closed-Loop Optogenetic Brain Stimulation Device,” *Electronics*, vol. 11, no. 10, Art. no. 10, Jan. 2022, doi: 10.3390/electronics11101591.
[104] D. Sunehra, V. S. Sreshta, V. Shashank, and B. U. Kumar Goud, “Raspberry Pi Based Smart Wearable Device for Women Safety using GPS and GSM Technology,” in *2020 IEEE International Conference for Innovation in Technology (INOCON)*, Nov. 2020, pp. 1–5. doi: 10.1109/INOCON50539.2020.9298449.
[105] S. Rihana and J. Mondalak, “Wearable fall detection system,” in *2016 3rd Middle East Conference on Biomedical Engineering (MECBME)*, Oct. 2016, pp. 84–87. doi: 10.1109/MECBME.2016.7745414.
[106] M. J. Rodrigues, O. Postolache, and F. Cercas, “Wearable Smart Sensing and UWB System for Fall Detection in AAL Environments,” in *2023 IEEE Sensors Applications Symposium (SAS)*, Jul. 2023, pp. 1–6. doi: 10.1109/SAS58821.2023.10254065.
[107] A. Dabbous, M. Fresta, F. Bellotti, and R. Berta, “Arduino Nano-Based System for Tennis Shot Classification,” *Lecture Notes in Electrical Engineering*, vol. 1113 LNEE, pp. 357–362, 2024, doi: 10.1007/978-3-031-48711-8_43.
[108] A. Soro, G. Brunner, S. Tanner, and R. Wattenhofer, “Recognition and Repetition Counting for Complex Physical Exercises with Deep Learning,” *Sensors*, vol. 19, no. 3, p. 714, Feb. 2019, doi: 10.3390/s19030714.
[109] K. Xia, H. Wang, M. Xu, Z. Li, S. He, and Y. Tang, “Racquet Sports Recognition Using a Hybrid Clustering Model Learned from Integrated Wearable Sensor,”

Sensors, vol. 20, no. 6, p. 1638, Mar. 2020, doi: 10.3390/s20061638.
[110] N. F. Ghazali, N. Shahar, N. A. Rahmad, N. A. J. Sufri, M. A. As'ari, and H. F. M. Latif, "Common sport activity recognition using inertial sensor," in *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, Batu Feringghi: IEEE, Mar. 2018, pp. 67–71. doi: 10.1109/CSPA.2018.8368687.
[111] "Nano Family," Arduino Official Store. Accessed: Feb. 13, 2024. [Online]. Available: https://store.arduino.cc/pages/nano-family
[112] C.-C. Yang, R. Pandey, T.-Y. Tu, Y.-P. Cheng, and P. C.-P. Chao, "An efficient energy harvesting circuit for batteryless IoT devices," *Microsyst Technol*, vol. 26, no. 1, pp. 195–207, Jan. 2020, doi: 10.1007/s00542-019-04544-7.
[113] "Flutter - Build apps for any screen." Accessed: Feb. 14, 2024. [Online]. Available: //flutter.dev/
[114] V. Thambawita *et al.*, "PMData: a sports logging dataset," in *Proceedings of the 11th ACM Multimedia Systems Conference*, Istanbul Turkey: ACM, May 2020, pp. 231–236. doi: 10.1145/3339825.3394926.
[115] M. Fresta *et al.*, "Efficient Uploading of.Csv Datasets into a Non-Relational Database Management System," in *Applications in Electronics Pervading Industry, Environment and Society*, vol. 1036, R. Berta and A. De Gloria, Eds., in Lecture Notes in Electrical Engineering, vol. 1036. , Cham: Springer Nature Switzerland, 2023, pp. 9–15. doi: 10.1007/978-3-031-30333-3_2.
[116] R. Berta, A. Kobeissi, F. Bellotti, and A. De Gloria, "Atmosphere, an Open Source Measurement-Oriented Data Framework for IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1927–1936, Mar. 2021, doi: 10.1109/TII.2020.2994414.
[117] A. Dabbous, M. Fresta, F. Bellotti, and R. Berta, "Neural Architecture for Tennis Shot Classification on Embedded System," in *Applications in Electronics Pervading Industry, Environment and Society*, F. Bellotti, M. D. Grammatikakis, A. Mansour, M. Ruo Roch, R. Seepold, A. Solanas, and R. Berta, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer Nature Switzerland, 2024, pp. 97–102. doi: 10.1007/978-3-031-48121-5_14.
[118] R. Khalife, R. Mrad, A. Dabbous, and A. Ibrahim, "Real-Time Implementation of Tiny Machine Learning Models for Hand Motion Classification," in *Applications in Electronics Pervading Industry, Environment and Society*, F. Bellotti, M. D. Grammatikakis, A. Mansour, M. Ruo Roch, R. Seepold, A. Solanas, and R. Berta, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer Nature Switzerland, 2024, pp. 487–492. doi: 10.1007/978-3-031-48121-5_70.
[119] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Evaluating effectiveness of shallow and deep networks to intrusion detection system," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2017, pp. 1282–1289. doi: 10.1109/ICACCI.2017.8126018.
[120] "Products | Joy-IT." Accessed: Mar. 13, 2024. [Online]. Available: https://joy-it.net/en/products/JT-UM120
[121] A. Testoni and M. Di Felice, "A software architecture for generic human activity recognition from smartphone sensor data," in *2017 IEEE International Workshop on Measurement and Networking (M&N)*, Naples, Italy: IEEE, Sep. 2017, pp. 1–6. doi: 10.1109/IWMN.2017.8078368.
[122] P. Balkhi and M. Moallem, "A Multipurpose Wearable Sensor-Based System for Weight Training," *Automation*, vol. 3, no. 1, pp. 132–152, Feb. 2022, doi: 10.3390/automation3010007.
[123] Y.-L. Hsu, S.-C. Yang, H.-C. Chang, and H.-C. Lai, "Human Daily and Sport

Activity Recognition Using a Wearable Inertial Sensor Network," *IEEE Access*, vol. 6, pp. 31715–31728, 2018, doi: 10.1109/ACCESS.2018.2839766.
[124] T. Perumal, E. Ramanujam, S. Suman, A. Sharma, and H. Singhal, "Internet of Things Centric-Based Multiactivity Recognition in Smart Home Environment," *IEEE Internet of Things Journal*, vol. 10, no. 2, pp. 1724–1732, Jan. 2023, doi: 10.1109/JIOT.2022.3209970.
[125] R. Berta, A. Mazzara, F. Bellotti, A. De Gloria, and L. Lazzaroni, "Edgine, A Runtime System for IoT Edge Applications," in *Applications in Electronics Pervading Industry, Environment and Society*, vol. 738, S. Saponara and A. De Gloria, Eds., in Lecture Notes in Electrical Engineering, vol. 738. , Cham: Springer International Publishing, 2021, pp. 261–266. doi: 10.1007/978-3-030-66729-0_31.
[126] "Nano 33 IoT | Arduino Documentation." Accessed: Jun. 19, 2023. [Online]. Available: https://docs.arduino.cc/hardware/nano-33-iot
[127] R. Ma, A. Ahmadzadeh, S. F. Boubrahimi, and R. A. Angryk, "A Scalable Segmented Dynamic Time Warping for Time Series Classification," in *Artificial Intelligence and Soft Computing*, L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. M. Zurada, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 407–419. doi: 10.1007/978-3-030-20915-5_37.
[128] L. Büthe, U. Blanke, H. Capkevics, and G. Tröster, "A wearable sensing system for timing analysis in tennis," in *2016 IEEE 13th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, Jun. 2016, pp. 43–48. doi: 10.1109/BSN.2016.7516230.
[129] O. Hazem and A. F. Al-Sadek, "Detection of Tennis Strokes using Wearable Sensor," in *2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sep. 2022, pp. 1–6. doi: 10.23919/SoftCOM55329.2022.9911405.
[130] M. Fresta *et al.*, "Low-Cost, Edge-Cloud, End-to-End System Architecture for Human Activity Data Collection," in *Applications in Electronics Pervading Industry, Environment and Society*, F. Bellotti, M. D. Grammatikakis, A. Mansour, M. Ruo Roch, R. Seepold, A. Solanas, and R. Berta, Eds., Cham: Springer Nature Switzerland, 2024, pp. 444–449. doi: 10.1007/978-3-031-48121-5_64.
[131] M. Fresta *et al.*, "End-to-End Dataset Collection System for Sport Activities," *Electronics (Switzerland)*, vol. 13, no. 7, 2024, doi: 10.3390/electronics13071286.
[132] M. Fresta *et al.*, "Efficient Uploading of.Csv Datasets into a Non-Relational Database Management System," in *Applications in Electronics Pervading Industry, Environment and Society*, R. Berta and A. De Gloria, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer Nature Switzerland, 2023, pp. 9–15. doi: 10.1007/978-3-031-30333-3_2.
[133] M. Fresta, A. Capello, F. Bellotti, L. Lazzaroni, M. Cossu, and R. Berta, "Supporting a .csv-based Workflow in MongoDB for Data Analysts," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, Jun. 2023, pp. 1–4. doi: 10.1109/ISIE51358.2023.10228044.
[134] "Z24 Bridge benchmark." Accessed: Jan. 17, 2024. [Online]. Available: https://bwk.kuleuven.be/bwm/z24/z24
[135] J. Maeck and G. De roeck, "DESCRIPTION OF Z24 BENCHMARK," *Mechanical Systems and Signal Processing*, vol. 17, no. 1, pp. 127–131, Jan. 2003, doi: 10.1006/mssp.2002.1548.
[136] P. Santaniello and P. Russo, "Bridge Damage Identification Using Deep Neural

Networks on Time–Frequency Signals Representation," *Sensors*, vol. 23, no. 13, Art. no. 13, Jan. 2023, doi: 10.3390/s23136152.
[137] A. van den Oord *et al.*, "WaveNet: A Generative Model for Raw Audio," Sep. 19, 2016, *arXiv*: arXiv:1609.03499. doi: 10.48550/arXiv.1609.03499.
[138] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs," Jun. 07, 2016, *arXiv*: arXiv:1412.7062. doi: 10.48550/arXiv.1412.7062.
[139] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 10, 2015, *arXiv*: arXiv:1512.03385. doi: 10.48550/arXiv.1512.03385.
[140] A. van den Oord, N. Kalchbrenner, L. Espeholt, koray kavukcuoglu, O. Vinyals, and A. Graves, "Conditional Image Generation with PixelCNN Decoders," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/b1301141feffabac455e1f90a7de2054-Paper.pdf
[141] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," presented at the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Jul. 2017, pp. 1800–1807. doi: 10.1109/CVPR.2017.195.
[142] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Mar. 16, 2016, *arXiv*: arXiv:1603.04467. doi: 10.48550/arXiv.1603.04467.
[143] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels," *Data Min Knowl Disc*, vol. 34, no. 5, pp. 1454–1495, Sep. 2020, doi: 10.1007/s10618-020-00701-z.
[144] A. Dempster, D. F. Schmidt, and G. I. Webb, "MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Aug. 2021, pp. 248–257. doi: 10.1145/3447548.3467231.
[145] "MiniRocket — sktime documentation." Accessed: Jan. 20, 2024. [Online]. Available: https://www.sktime.net/en/stable/examples/transformation/minirocket.html
[146] S. Sony, S. Gamage, A. Sadhu, and J. Samarabandu, "Multiclass Damage Identification in a Full-Scale Bridge Using Optimally Tuned One-Dimensional Convolutional Neural Network," *J. Comput. Civ. Eng.*, vol. 36, no. 2, p. 04021035, Mar. 2022, doi: 10.1061/(ASCE)CP.1943-5487.0001003.
[147] T. Le-Xuan, T. Bui-Tien, and H. Tran-Ngoc, "A novel approach model design for signal data using 1DCNN combing with LSTM and ResNet for damaged detection problem," *Structures*, vol. 59, p. 105784, Jan. 2024, doi: 10.1016/j.istruc.2023.105784.
[148] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, Apr. 2023, doi: 10.1016/j.suscom.2023.100857.
[149] K. J. Singh and D. S. Kapoor, "A survey of IoT platforms: Create Your Own Internet of Things," *IEEE Consumer Electron. Mag.*, vol. 6, no. 2, pp. 57–68, Apr. 2017, doi: 10.1109/MCE.2016.2640718.

[150] S. M. Lundberg and S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Jul. 11, 2022. [Online]. Available: https://papers.nips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html
[151] C. Molnar, *Interpretable Machine Learning*. 2022. Accessed: May 07, 2024. [Online]. Available: https://christophm.github.io/interpretable-ml-book/
[152] F. Bellotti, L. Lazzaroni, A. Capello, M. Cossu, A. De Gloria, and R. Berta, “Explaining a Deep Reinforcement Learning (DRL)-Based Automated Driving Agent in Highway Simulations,” *IEEE Access*, vol. 11, pp. 28522–28550, 2023, doi: 10.1109/ACCESS.2023.3259544.
[153] F. Sakr, R. Berta, J. Doyle, H. Younes, A. De Gloria, and F. Bellotti, “Memory Efficient Binary Convolutional Neural Networks on Microcontrollers,” in *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*, Jul. 2022, pp. 169–177. doi: 10.1109/EDGE55608.2022.00032.
[154] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: training deep neural networks with binary weights during propagations,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, in NIPS’15. Cambridge, MA, USA: MIT Press, Dec. 2015, pp. 3123–3131.
[155] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, “Binary neural networks: A survey,” *Pattern Recognition*, vol. 105, p. 107281, Sep. 2020, doi: 10.1016/j.patcog.2020.107281.
[156] “Larq | Binarized Neural Network development.” Accessed: Jan. 15, 2026. [Online]. Available: https://larq.dev/
[157] T. Le-Xuan, T. Bui-Tien, and H. Tran-Ngoc, “A novel approach model design for signal data using 1DCNN combing with LSTM and ResNet for damaged detection problem,” *Structures*, vol. 59, p. 105784, Jan. 2024, doi: 10.1016/j.istruc.2023.105784.
[158] A. Kashevnik, R. Shchedrin, C. Kaiser, and A. Stocker, “Driver Distraction Detection Methods: A Literature Review and Framework,” *IEEE Access*, vol. 9, pp. 60063–60076, 2021, doi: 10.1109/ACCESS.2021.3073599.
[159] T. Stewart, “Overview of Motor Vehicle Crashes in 2020,” Art. no. DOT HS 813 266, Mar. 2022, Accessed: Aug. 05, 2024. [Online]. Available: https://trid.trb.org/View/1922738
[160] “J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International.” Accessed: Mar. 11, 2025. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/
[161] L. Pipkorn, E. Tivesten, C. Flannagan, and M. Dozza, “Driver Response to Take-Over Requests in Real Traffic,” *IEEE Transactions on Human-Machine Systems*, vol. 53, no. 5, pp. 823–833, Oct. 2023, doi: 10.1109/THMS.2023.3304003.
[162] T. Ersal, H. J. A. Fuller, O. Tsimhoni, J. L. Stein, and H. K. Fathy, “Model-Based Analysis and Classification of Driver Distraction Under Secondary Tasks,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 692–701, Sep. 2010, doi: 10.1109/TITS.2010.2049741.
[163] M. A. Regan, C. Hallett, and C. P. Gordon, “Driver distraction and driver inattention: Definition, relationship and taxonomy,” *Accident Analysis & Prevention*, vol. 43, no. 5, pp. 1771–1781, Sep. 2011, doi: 10.1016/j.aap.2011.04.008.
[164] A. Kashevnik, R. Shchedrin, C. Kaiser, and A. Stocker, “Driver Distraction

Detection Methods: A Literature Review and Framework," *IEEE Access*, vol. 9, pp. 60063–60076, 2021, doi: 10.1109/ACCESS.2021.3073599.
[165] R. Dewar and P. Olson, *Human Factors in Traffic Safety, Second Edition*. 2007. Accessed: Aug. 05, 2024. [Online]. Available: https://trid.trb.org/View/815003
[166] A. Capello *et al.*, "Exploiting Big Data for Experiment Reporting: The Hi-Drive Collaborative Research Project Case," *Sensors*, vol. 23, no. 18, Art. no. 18, Jan. 2023, doi: 10.3390/s23187866.
[167] F. Faul, E. Erdfelder, A. Buchner, and A.-G. Lang, "Statistical power analyses using G*Power 3.1: Tests for correlation and regression analyses," *Behavior Research Methods*, vol. 41, no. 4, pp. 1149–1160, Nov. 2009, doi: 10.3758/BRM.41.4.1149.
[168] A. Ezzouhri, Z. Charouh, M. Ghogho, and Z. Guennoun, "Robust Deep Learning-Based Driver Distraction Detection and Classification," *IEEE Access*, vol. 9, pp. 168080–168092, 2021, doi: 10.1109/ACCESS.2021.3133797.
[169] G. Li, W. Yan, S. Li, X. Qu, W. Chu, and D. Cao, "A Temporal–Spatial Deep Learning Approach for Driver Distraction Detection Based on EEG Signals," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 2665–2677, Oct. 2022, doi: 10.1109/TASE.2021.3088897.
[170] F. Sajid, A. R. Javed, A. Basharat, N. Kryvinska, A. Afzal, and M. Rizwan, "An Efficient Deep Learning Framework for Distracted Driver Detection," *IEEE Access*, vol. 9, pp. 169270–169280, 2021, doi: 10.1109/ACCESS.2021.3138137.
[171] A. Misra, S. Samuel, S. Cao, and K. Shariatmadari, "Detection of Driver Cognitive Distraction Using Machine Learning Methods," *IEEE Access*, vol. 11, pp. 18000–18012, 2023, doi: 10.1109/ACCESS.2023.3245122.
[172] T. Liu, Y. Yang, G.-B. Huang, Y. K. Yeo, and Z. Lin, "Driver Distraction Detection Using Semi-Supervised Machine Learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1108–1120, Apr. 2016, doi: 10.1109/TITS.2015.2496157.
[173] M. Gjoreski, M. Ž. Gams, M. Luštrek, P. Genc, J.-U. Garbas, and T. Hassan, "Machine Learning and End-to-End Deep Learning for Monitoring Driver Distractions From Physiological and Visual Signals," *IEEE Access*, vol. 8, pp. 70590–70603, 2020, doi: 10.1109/ACCESS.2020.2986810.
[174] N. Merat, A. H. Jamson, F. C. H. Lai, and O. Carsten, "Highly Automated Driving, Secondary Task Performance, and Driver State," *Hum Factors*, vol. 54, no. 5, pp. 762–771, Oct. 2012, doi: 10.1177/0018720812442087.
[175] N. von Janczewski, J. Wittmann, A. Engeln, M. Baumann, and L. Krauß, "A *meta-analysis* of the n-back task while driving and its effects on cognitive workload," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 76, pp. 269–285, Jan. 2021, doi: 10.1016/j.trf.2020.11.014.
[176] V. Jeyhani, M. Mantysalo, K. Noponen, T. Seppanen, and A. Vehkaoja, "Effect of Different ECG Leads on Estimated R-R Intervals and Heart Rate Variability Parameters," *Annu Int Conf IEEE Eng Med Biol Soc*, vol. 2019, pp. 3786–3790, Jul. 2019, doi: 10.1109/EMBC.2019.8857954.
[177] J. D. Van Der Laan, A. Heino, and D. De Waard, "A simple procedure for the assessment of acceptance of advanced transport telematics," *Transportation Research Part C: Emerging Technologies*, vol. 5, no. 1, pp. 1–10, Feb. 1997, doi: 10.1016/S0968-090X(96)00025-3.
[178] R. Agarwal and J. Prasad, "A Conceptual and Operational Definition of Personal Innovativeness in the Domain of Information Technology," *Information Systems Research*, vol. 9, no. 2, pp. 204–215, Jun. 1998, doi: 10.1287/isre.9.2.204.

[179] S. V. Wass, T. J. Smith, and M. H. Johnson, “Parsing eye-tracking data of variable quality to provide accurate fixation duration estimates in infants and adults,” *Behav Res*, vol. 45, no. 1, pp. 229–250, Mar. 2013, doi: 10.3758/s13428-012-0245-6.

[180] D. A. Huffman, “The synthesis of sequential switching circuits,” *Journal of the Franklin Institute*, vol. 257, no. 3, pp. 161–190, Mar. 1954, doi: 10.1016/0016-0032(54)90574-8.

[181] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, in ICML ’09. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 609–616. doi: 10.1145/1553374.1553453.

[182] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, “1D convolutional neural networks and applications: A survey,” *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, Apr. 2021, doi: 10.1016/j.ymssp.2020.107398.

[183] A. Dempster, D. F. Schmidt, and G. I. Webb, “MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, in KDD ’21. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 248–257. doi: 10.1145/3447548.3467231.

[184] F. Yu and V. Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” Apr. 30, 2016, *arXiv*: arXiv:1511.07122. doi: 10.48550/arXiv.1511.07122.

[185] A. van den Oord *et al.*, “WaveNet: A Generative Model for Raw Audio,” Sep. 19, 2016, *arXiv*: arXiv:1609.03499. Accessed: Aug. 05, 2024. [Online]. Available: http://arxiv.org/abs/1609.03499

[186] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Aug. 05, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[187] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[188] “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition[Book].” Accessed: Jan. 02, 2025. [Online]. Available: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

[189] K. Benidis *et al.*, “Deep Learning for Time Series Forecasting: Tutorial and Literature Survey,” *ACM Comput. Surv.*, vol. 55, no. 6, p. 121:1-121:36, Dec. 2022, doi: 10.1145/3533382.

[190] A. Dabbous, R. Berta, M. Fresta, H. Ballout, L. Lazzaroni, and F. Bellotti, “Bringing Intelligence to the Edge for Structural Health Monitoring. The Case Study of the Z24 Bridge,” *IEEE Open Journal of the Industrial Electronics Society*, pp. 1–15, 2024, doi: 10.1109/OJIES.2024.3434341.

[191] P. Domingos, “A few useful things to know about machine learning,” *Commun. ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012, doi: 10.1145/2347736.2347755.

[192] D. J. J. Braithwaite, “A Guide for Analysing Electrodermal Activity (EDA) & Skin Conductance Responses (SCRs) for Psychological Experiments”.

[193] W. Morales-Alvarez, O. Sipele, R. Léberon, H. H. Tadjine, and C. Olaverri-Monreal, "Automated Driving: A Literature Review of the Take over Request in Conditional Automation," *Electronics*, vol. 9, no. 12, Art. no. 12, Dec. 2020, doi: 10.3390/electronics9122087.
[194] J. Son, M. Park, and B. B. Park, "The effect of age, gender and roadway environment on the acceptance and effectiveness of Advanced Driver Assistance Systems," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 31, pp. 12–24, May 2015, doi: 10.1016/j.trf.2015.03.009.
[195] S. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," Nov. 24, 2017, *arXiv*: arXiv:1705.07874. doi: 10.48550/arXiv.1705.07874.
[196] O. Loyola-González, "Black-Box vs. White-Box: Understanding Their Advantages and Weaknesses From a Practical Point of View," *IEEE Access*, vol. 7, pp. 154096–154113, 2019, doi: 10.1109/ACCESS.2019.2949286.
[197] M. A. Regan, C. Hallett, and C. P. Gordon, "Driver distraction and driver inattention: Definition, relationship and taxonomy," *Accident Analysis & Prevention*, vol. 43, no. 5, pp. 1771–1781, Sep. 2011, doi: 10.1016/j.aap.2011.04.008.
[198] H. B. Sundfør, F. Sagberg, and A. Høye, "Inattention and distraction in fatal road crashes – Results from in-depth crash investigations in Norway," *Accident Analysis & Prevention*, vol. 125, pp. 152–157, Apr. 2019, doi: 10.1016/j.aap.2019.02.004.
[199] A. Amditis *et al.*, "Towards the Automotive HMI of the Future: Overview of the AIDE-Integrated Project Results," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 567–578, Sep. 2010, doi: 10.1109/TITS.2010.2048751.
[200] F. Bellotti, A. D. Gloria, R. Montanari, N. Dosio, and D. Morreale, "COMUNICAR: designing a multimedia, context-aware human-machine interface for cars," *Cogn Tech Work*, vol. 7, no. 1, pp. 36–45, Mar. 2005, doi: 10.1007/s10111-004-0168-9.
[201] M. L. Cunningham and M. A. Regan, "Driver distraction and inattention in the realm of automated driving," *IET Intelligent Transport Systems*, vol. 12, no. 6, pp. 407–413, 2018, doi: 10.1049/iet-its.2017.0232.
[202] X. Shi, "Driver Distraction Behavior Detection Framework Based on the DWPose Model, Kalman Filtering, and Multi-Transformer," *IEEE Access*, vol. 12, pp. 80579–80589, 2024, doi: 10.1109/ACCESS.2024.3406605.
[203] S. Obaid, N. Z. Bawany, S. Tahzeeb, T. Qamar, and M. H. Mughal, "Deep Learning-Based Driver Activity Recognition Using Diverse Driver Profiles," *IEEE Access*, vol. 12, pp. 187192–187209, 2024, doi: 10.1109/ACCESS.2024.3489220.
[204] G. Azizoglu and A. N. Toprak, "MELD3: Integrating Multi-Task Ensemble Learning for Driver Distraction Detection," *IEEE Access*, vol. 12, pp. 186022–186034, 2024, doi: 10.1109/ACCESS.2024.3509033.
[205] S. Hwang, A. G. Banerjee, and L. N. Boyle, "Predicting Driver's Transition Time to a Secondary Task Given an in-Vehicle Alert," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 5, pp. 4739–4745, May 2022, doi: 10.1109/TITS.2020.3018395.
[206] X. Zhao, Z. Li, C. Zhao, and C. Wang, "Real-Time Multistep Time-Series Prediction of Driver's Head Pose During IVIS Secondary Tasks for Human–Machine Codriving and Distraction Warning Systems," *IEEE Sensors Journal*, vol. 22, no. 24, pp. 24364–24379, Dec. 2022, doi: 10.1109/JSEN.2022.3216057.

[207] G. Strle, A. Košir, J. Sodnik, and K. S. Pečečnik, “Physiological Signals as Predictors of Cognitive Load Induced by the Type of Automotive Head-Up Display,” *IEEE Access*, vol. 11, pp. 87835–87848, 2023, doi: 10.1109/ACCESS.2023.3305383.
[208] M. O. Rashid *et al.*, “Self-DSNet: A Novel Self-ONNs based Deep Learning Framework for Multimodal Driving Distraction Detection,” *IEEE Access*, pp. 1–1, 2025, doi: 10.1109/ACCESS.2025.3545359.
[209] L. Mou *et al.*, “Multimodal driver distraction detection using dual-channel network of CNN and Transformer,” *Expert Systems with Applications*, vol. 234, p. 121066, Dec. 2023, doi: 10.1016/j.eswa.2023.121066.
[210] M. Gjoreski, M. Ž. Gams, M. Luštrek, P. Genc, J.-U. Garbas, and T. Hassan, “Machine Learning and End-to-End Deep Learning for Monitoring Driver Distractions From Physiological and Visual Signals,” *IEEE Access*, vol. 8, pp. 70590–70603, 2020, doi: 10.1109/ACCESS.2020.2986810.
[211] A. Misra, S. Samuel, S. Cao, and K. Shariatmadari, “Detection of Driver Cognitive Distraction Using Machine Learning Methods,” *IEEE Access*, vol. 11, pp. 18000–18012, 2023, doi: 10.1109/ACCESS.2023.3245122.
[212] M. Fresta *et al.*, “Deep Learning-Based Real-Time Driver Cognitive Distraction Detection,” *IEEE Access*, vol. 13, pp. 26589–26607, 2025, doi: 10.1109/ACCESS.2025.3539392.
[213] A. Shylendra, P. Shukla, S. Mukhopadhyay, S. Bhunia, and A. R. Trivedi, “Low Power Unsupervised Anomaly Detection by Nonparametric Modeling of Sensor Statistics,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 8, pp. 1833–1843, Aug. 2020, doi: 10.1109/TVLSI.2020.2984472.
[214] P. Schäfer and U. Leser, “Fast and Accurate Time Series Classification with WEASEL,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, in CIKM ’17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 637–646. doi: 10.1145/3132847.3132980.
[215] Q. Dai, J. Liu, and J.-P. Yang, “SWSEL: Sliding Window-based Selective Ensemble Learning for class-imbalance problems,” *Engineering Applications of Artificial Intelligence*, vol. 121, p. 105959, May 2023, doi: 10.1016/j.engappai.2023.105959.
[216] M. Zhao, A. Sadhu, and M. Capretz, “Multiclass anomaly detection in imbalanced structural health monitoring data using convolutional neural network,” *Journal of Infrastructure Preservation and Resilience*, vol. 3, Aug. 2022, doi: 10.1186/s43065-022-00055-4.
[217] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.
[218] J. H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine,” *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.
[219] Y. Yu, X. Si, C. Hu, and J. Zhang, “A review of recurrent neural networks: Lstm cells and network architectures,” *Neural Comput.*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019, doi: 10.1162/neco_a_01199.
[220] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds., Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. doi: 10.3115/v1/D14-1179.
[221] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, “Review of Image Classification Algorithms Based on Convolutional Neural Networks,” *Remote*

Sensing, vol. 13, no. 22, Art. no. 22, Jan. 2021, doi: 10.3390/rs13224712.
[222] M. Banko and E. Brill, "Scaling to Very Very Large Corpora for Natural Language Disambiguation," in *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, France: Association for Computational Linguistics, Jul. 2001, pp. 26–33. doi: 10.3115/1073012.1073017.
[223] A. Halevy, P. Norvig, and F. Pereira, "The Unreasonable Effectiveness of Data," *IEEE Intell. Syst.*, vol. 24, no. 2, pp. 8–12, Mar. 2009, doi: 10.1109/MIS.2009.36.
[224] "More data usually beats better algorithms," Datawocky. Accessed: Mar. 15, 2025. [Online]. Available: https://anand.typepad.com/datawocky/2008/03/more-data-usual.html
[225] H. Namdari, A. Haghighi, and S. M. Ashrafi, "Short-term urban water demand forecasting; application of 1D convolutional neural network (1D CNN) in comparison with different deep learning schemes," *Stoch Environ Res Risk Assess*, Sep. 2023, doi: 10.1007/s00477-023-02565-3.
[226] "Delegated regulation - EU - 2023/2590 - EN - EUR-Lex." Accessed: Jun. 26, 2025. [Online]. Available: https://eur-lex.europa.eu/eli/reg_del/2023/2590/oj/eng
[227] "NVIDIA TensorRT," NVIDIA Docs. Accessed: Mar. 17, 2025. [Online]. Available: https://docs.nvidia.com/tensorrt/index.html
[228] "Jetson Stats," NVIDIA Developer. Accessed: Mar. 17, 2025. [Online]. Available: https://developer.nvidia.com/embedded/community/jetson-projects/jetson_stats
[229] "X-NUCLEO-LPM01A - STM32 Power shield, Nucleo expansion board for power consumption measurement (UM2243) - STMicroelectronics." Accessed: Mar. 17, 2025. [Online]. Available: https://www.st.com/en/evaluation-tools/x-nucleo-lpm01a.html
[230] M. Openja, A. Nikanjam, A. H. Yahmed, F. Khomh, and Z. M. J. Jiang, "An Empirical Study of Challenges in Converting Deep Learning Models," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct. 2022, pp. 13–23. doi: 10.1109/ICSME55016.2022.00010.
[231] D. Zhang, C. Zhong, P. Xu, and Y. Tian, "Deep Learning in the State of Charge Estimation for Li-Ion Batteries of Electric Vehicles: A Review," *Machines*, vol. 10, no. 10, Art. no. 10, Oct. 2022, doi: 10.3390/machines10100912.
[232] U. M. Damodarin, G. C. Cardarilli, L. Di Nunzio, M. Re, and S. Spanò, "Smart Electric Vehicle Charging Management Using Reinforcement Learning on FPGA Platforms," *Sensors*, vol. 25, no. 8, Art. no. 8, Jan. 2025, doi: 10.3390/s25082585.
[233] H. Dhungana, F. Bellotti, M. Fresta, P. Dhungana, and R. Berta, "Assessing a Measurement-Oriented Data Management Framework in Energy IoT Applications," *Energies*, vol. 18, no. 13, Art. no. 13, Jan. 2025, doi: 10.3390/en18133347.
[234] A. B. de Lima, M. B. C. Salles, and J. R. Cardoso, "State-of-Charge Estimation of a Li-Ion Battery using Deep Forward Neural Networks," Sep. 20, 2020, *arXiv*: arXiv:2009.09543. doi: 10.48550/arXiv.2009.09543.
[235] D. Ouyang, M. Chen, J. Liu, R. Wei, J. Weng, and J. Wang, "Investigation of a commercial lithium-ion battery under overcharge/over-discharge failure conditions," *RSC Adv.*, vol. 8, no. 58, pp. 33414–33424, Sep. 2018, doi: 10.1039/C8RA05564E.
[236] K. Omiloli, A. Awelewa, I. Samuel, O. Obiazi, and J. Katende, "State of charge estimation based on a modified extended Kalman filter," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 5, Art. no. 5, Oct. 2023, doi: 10.11591/ijece.v13i5.pp5054-5065.
[237] A. Dubey, A. Zaidi, and A. Kulshreshtha, "State-of-Charge Estimation Algorithm

for Li-ion Batteries using Long Short-Term Memory Network with Bayesian Optimization," in *2022 Second International Conference on Interdisciplinary Cyber Physical Systems (ICPS)*, May 2022, pp. 68–73. doi: 10.1109/ICPS55917.2022.00021.

[238] E. Chemali, P. J. Kollmeyer, M. Preindl, R. Ahmed, and A. Emadi, "Long Short-Term Memory Networks for Accurate State-of-Charge Estimation of Li-ion Batteries," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6730–6739, Aug. 2018, doi: 10.1109/TIE.2017.2787586.

[239] Y.-C. Wang, N.-C. Shao, G.-W. Chen, W.-S. Hsu, and S.-C. Wu, "State-of-Charge Estimation for Lithium-Ion Batteries Using Residual Convolutional Neural Networks," *Sensors*, vol. 22, no. 16, Art. no. 16, Jan. 2022, doi: 10.3390/s22166303.

[240] M. A. Hannan *et al.*, "Deep learning approach towards accurate state of charge estimation for lithium-ion batteries using self-supervised transformer model," *Sci Rep*, vol. 11, no. 1, p. 19541, Oct. 2021, doi: 10.1038/s41598-021-98915-8.

[241] K. L. Wong, M. Bosello, R. Tse, C. Falcomer, C. Rossi, and G. Pau, "Li-Ion Batteries State-of-Charge Estimation Using Deep LSTM at Various Battery Specifications and Discharge Cycles," in *Proceedings of the Conference on Information Technology for Social Good*, in GoodIT '21. New York, NY, USA: Association for Computing Machinery, Sep. 2021, pp. 85–90. doi: 10.1145/3462203.3475878.

[242] A. Pighetti, F. Bellotti, R. Berta, A. Cavallaro, L. Lazzaroni, and C. Oh, "RAFT: Regularized Adversarial Fine-Tuning to Enhance Deep Reinforcement Learning for Self-Parking," *IEEE Sensors Letters*, pp. 1–4, 2025, doi: 10.1109/LSENS.2025.3600982.

[243] S. Lu, Y. Wang, L. Sheng, A. Zheng, L. He, and J. Liang, "Recent Advances in OOD Detection: Problems and Approaches," Sep. 21, 2024, *arXiv*: arXiv:2409.11884. doi: 10.48550/arXiv.2409.11884.

[244] A. Capello *et al.*, "Exploiting Big Data for Experiment Reporting: The Hi-Drive Collaborative Research Project Case," *Sensors*, vol. 23, no. 18, Art. no. 18, Jan. 2023, doi: 10.3390/s23187866.

[245] L. Forneris *et al.*, "Implementing Deep Reinforcement Learning (DRL)-based Driving Styles for Non-Player Vehicles," *International Journal of Serious Games*, vol. 10, no. 4, Art. no. 4, Nov. 2023, doi: 10.17083/ijsg.v10i4.638.

[246] H. Ballout *et al.*, "Runtime Input Monitoring of a Reinforcement Learning-Based Agent for Automated Car Parking," in *Applications in Electronics Pervading Industry, Environment and Society*, M. Ruo Roch, F. Bellotti, R. Berta, M. Martina, and P. Motto Ros, Eds., Cham: Springer Nature Switzerland, 2025, pp. 387–394. doi: 10.1007/978-3-031-84100-2_46.

[247] Z. Lin, Y. Chen, Y. Xie, C. Chen, J. Yu, and J. Chen, "An analytical timing-driven placer for modern heterogeneous FPGAs," *J Supercomput*, vol. 81, no. 1, p. 268, Dec. 2024, doi: 10.1007/s11227-024-06755-w.

[248] S. A. Alam, D. Gregg, G. Gambardella, T. Preusser, and M. Blott, "On the RTL Implementation of FINN Matrix Vector Compute Unit," Apr. 11, 2022, *arXiv*: arXiv:2201.11409. doi: 10.48550/arXiv.2201.11409.

[249] Y. Shi *et al.*, "Timing-Driven Global Placement by Efficient Critical Path Extraction," Feb. 28, 2025, *arXiv*: arXiv:2503.11674. doi: 10.48550/arXiv.2503.11674.