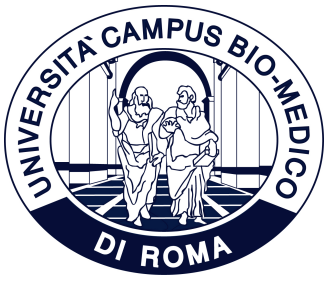


**Università Campus Bio-Medico di Roma**

Corso di Dottorato di Ricerca in
Scienze e Ingegneria per l'Uomo e l'Ambiente
XXXVII ciclo

# Pushing the boundaries of healthcare: AI-IoT solutions for a secure precise and explainable healthcare system

**Lorenzo Petrosino**

Tutor:
Prof. Luca Vollero
Dott. Ing. Mario Merone

3/January/2025

# Declaration of Authorship

I, Lorenzo PETROSINO, declare that this thesis titled, "Pushing the boundaries of healthcare: AI-IoT solutions for a secure precise and explainable healthcare system" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

# *Acknowledgements*

My first and most heartfelt thanks go to my family, who, with love and unwavering support, gave me the privilege of dedicating additional years of my life to study and education. Without you, without your constant presence and encouragement, none of this would have been possible. Thank you for giving me the freedom to pursue my dreams and for believing in me even in the moments when I struggled to do so myself.

A special thanks goes to my supervisors, Professor Vollero and Professor Merone. To Vollero, I owe his calm wisdom and extraordinary technical expertise: he has been a beacon in the storms of research, with an almost prophetic ability to find solutions to any problem I brought to him. Merone, on the other hand, enriched my experience also with an invaluable human dimension, proving to be not only a supervisor but also a friend, and an older brother, always ready to listen and understand me not only as a researcher but, above all, as a person. To both of them, I express my deepest gratitude.

Thank you to my travel companions, Luca, Federico, Onorato, Paolo, Daniele, and Alessandro. More than colleagues, you have been true friends, with whom I have shared every phase of this experience from enthusiasm to challenges supporting each other every day. We have built more than just a professional relationship: we have formed a team.

A heartfelt thanks also goes to the new additions to the group, Claudia, Agnese, Anna, Ruben, and Alberto, who, despite the short time we have worked together, have already given me so much. I am sure that the future holds great satisfaction for you.

And then there are them, the usual crazy ones. Those eccentrics who have always been there, that bunch of misfits whom I know, and have always known, I can count on. Banfro, Fabrizio, Riccardo, and Alfredo, thank you from the bottom of my heart for your friendship, for being an absolute certainty in a world that often changes too quickly.

Finally, a special thanks to Elena, a wonderful person who manages to put up with me every day with a smile on her face. Thank you for your patience, your support, for being my refuge and my strength throughout this journey.

To all of you, thank you. This thesis bears my name, but it is the result of a journey I have taken with you.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **NN** | Neural Network |
| **MLP** | Multi-Layer Perceptron |
| **CNN** | Convolutional Neural Network |
| **RNN** | Recurrent Neural Network |
| **LSTM** | Long Short-Term Memory |
| **VLSTM** | Vanilla Long Short-Term Memory |
| **BiLSTM** | Bidirectional Long Short-Term Memory |
| **XAI** | Explainable Artificial Intelligence |
| **LIME** | Local Interpretable Model-Agnostic Explanation |
| **IoU** | Intersection Over Union |
| **MAC** | Multiply-Accumulate |
| **ROM** | Read-Only Memory |
| **RAM** | Random Access Memory |
| **MCU** | Microcontroller Unit |
| **IDE** | Integrated Development Environment |
| **RGB** | Red, Green, Blue |
| **ADPKD** | Autosomal Dominant Polycystic Kidney Disease |
| **LiTS** | Liver Tumor Segmentation |
| **BCE** | Binary Cross-Entropy |
| **CT** | Computed Tomography |
| **IRCCS** | Istituto di Ricovero e Cura a Carattere Scientifico |
| **ZKP** | Zero-Knowledge Proof |
| **iZKP** | Interactive Zero-Knowledge Proof |
| **NIZK** | Non-interactive Zero-Knowledge |
| **SNARK** | Succinct Non-interactive Zero-Knowledge Proof |
| **DLT** | Distributed Ledger Technologies |
| **FL** | Federated Learning |
| **DiD** | Decentralized Identifiers |
| **CGM** | Continuous Glucose Monitoring |
| **CSII** | Continuous Subcutaneous Insulin Infusion |
| **CHO** | Operative Carbohydrates |
| **IOB** | Insulin On Board |
| **BGLP** | Blood Glucose Level Prediction |
| **EVM** | Ethereum Virtual Machine |
| **OTfR** | One Time for Round |
| **OTO** | One Time Only |
| **OfEC** | Only for External Check |
| **OiN** | Only if Needed |
| **EC** | Every Check |
| **Lopo** | Leave One Patient Out |
| **GDPR** | General Data Protection Regulation of the European Union |

| | |
|---|---|
| **CCPA** | California Consumer Privacy Act |
| **OL** | Online Learning |
| **ELM** | Extreme Learning Machine |
| **OS-ELM** | Online Sequential Extreme Learning Machine |
| **ROS-ELM** | Robust Online Sequential Extreme Learning Machine |
| **FedROS-ELM** | Federated Robust Online Sequential Extreme Learning Machine |
| **T1DM** | Type 1 Diabetes Mellitus |
| **BGL** | Blood Glucose Level |
| **PH** | Prediction Horizon |
| **SWstep** | Sliding Window Step |
| **CEG** | Clarke Error Grid |
| **RMSE** | Root Mean Square Error |
| **FLOPs** | Floating Point Operations |
| **ReLU** | Rectified Linear Unit |
| **VPU** | Vision Processing Unit |

# Chapter 1

# Introduction

## 1.1 Motivation

The integration of Artificial Intelligence (AI) with the Internet of Things (IoT) has revolutionized numerous domains, and healthcare stands as a critical beneficiary of these advancements. The promise of improved diagnostic precision, personalized treatment, and enhanced operational efficiency is shaping the future of Healthcare 5.0 [3,4]. However, this transformation is not without challenges. Complex computational demands, the need for explainable and interpretable models, and stringent security and privacy requirements hinder the seamless adoption of AI-IoT solutions in clinical and real-world scenarios. These issues provided the impetus for this research, which aims to develop robust methodologies addressing these pressing challenges.

Edge devices, a cornerstone of IoT ecosystems, are constrained by limited computational resources, energy efficiency, and memory capacity. Yet, they must perform complex inference tasks in real time to enable actionable insights in critical scenarios like continuous glucose monitoring or arrhythmia detection. Current AI solutions often rely on cloud computing, introducing latency and risking data breaches. The need to optimize AI models for resource-constrained environments drives this research's focus on model compression techniques such as pruning and quantization, ensuring that cutting-edge Neural Networks (NNs) can operate effectively within the constraints of edge devices.

While AI systems have achieved remarkable performance in various applications, their black-box nature poses significant challenges for adoption in healthcare. Clinicians and stakeholders require transparent and interpretable solutions to trust AI-driven recommendations, especially in life-critical situations. This research emphasizes the development of explainable AI techniques, such as feature-based knowledge distillation to embed transparency into predictive models. These efforts aim to enhance user trust and support clinical decision-making by offering clear insights into the reasoning behind AI predictions.

Healthcare data are inherently sensitive, and ensuring their security is paramount. Conventional centralized learning approaches require data to be collected to train predictive models, often compromising patient confidentiality and trust. This research prioritizes privacy-preserving methodologies by integrating Federated Learning frameworks with cryptographic enhancements such as Zero-Knowledge Proofs and Distributed Ledger Technology. These innovations enable secure, scalable AI deployments that uphold data privacy without compromising performance, particularly in decentralized IoT environments.

The motivation for this thesis is rooted in the potential of AI-IoT solutions to transform healthcare delivery. From enabling real-time monitoring and diagnosis to facilitating predictive and preventive healthcare, the applications of this technology

are vast. This work addresses the interdisciplinary challenges required to bridge the gap between theoretical advancements and practical deployments, ensuring that AI-IoT systems are not only effective but also trustworthy, interpretable, and secure.

In summary, this research is driven by the pressing need to overcome the technical, ethical, and operational barriers to AI-IoT integration in healthcare. By focusing on computational efficiency, interpretability, and security, this work contributes to the realization of intelligent, impactful, and sustainable solutions for Healthcare 5.0.

## 1.2 Dissertation Structure

This dissertation is organized to provide a clear and logical flow from foundational concepts to advanced methodologies and their practical applications. Chapters 3, 4, and 5 share a similar structure, functioning as self-contained units. Each Chapter addresses a distinct theme, presents its methodologies, and findings. This design allows for a modular understanding of the research while maintaining a cohesive narrative throughout the dissertation.

The research begins with an introduction in 1, where the context, motivation, and primary objectives of the study are presented. This Chapter lays the groundwork by identifying key challenges in deploying secure, efficient, and interpretable AI-IoT systems in healthcare.

The following Chapter, 2, delves into the state-of-the-art technologies and methodologies relevant to AI and IoT integration. It discusses the fundamental principles of edge computing, model compression, and federated learning while highlighting the critical gaps this research aims to address.

In 3, the focus shifts to optimizing AI models for deployment on edge devices. Techniques such as pruning and quantization are explored to enable real-time applications in resource-constrained environments 3.3. These methodologies are validated through case studies, including glucose level prediction 3.4 and microplastic detection 3.6, showcasing their practicality and versatility.

The discussion on explainability and interpretability is presented in 4. This Chapter emphasizes the importance of transparent AI systems in healthcare, introducing techniques such as feature-based knowledge distillation 4.5.These methods are applied to real-world scenarios, such as predicting adverse events in Type 1 Diabetes (T1D) 4.4 and enhancing medical imaging solutions 4.3, demonstrating the value of interpretable models in critical applications.

Security and privacy considerations are addressed in 5, where novel federated learning frameworks are introduced. These frameworks incorporate Zero-Knowledge Proofs and Distributed Ledger Technology to ensure data integrity and scalability while preserving privacy. The Chapter also details advanced architectures like dRAIN 5.3 and graph-aided federated learning 5.6, which provide robust solutions for secure IoT networks.

Finally, 6 synthesizes the findings of this research. This Chapter reflects on the implications of the results and proposes future directions, aiming to further advance Healthcare 5.0 by balancing computational efficiency, interpretability, and security.

## 1.3 Contribution

This doctoral research encompasses a variety of methodologies and applications in the development of resource-efficient, interpretable, and secure AI systems. The contributions can be grouped into three main areas:

1. **Integration of AI and IoT**
2. **Explainability and Interpretability of Models**
3. **Secure and Privacy-Preserving Systems**

### 1.3.1 Integration of AI and IoT

One of the core objectives of this thesis was to design and deploy AI solutions that can function effectively on edge devices in IoT environments:

- **Model Optimization on Edge Devices:** Techniques such as *pruning* and *quantization* were explored to reduce Multiply–Accumulate operations (MACs), memory requirements, and speed up inference times. These methods enabled NNs to run concurrently on cost-effective, low-power hardware.
- **Hardware Acceleration:** By leveraging specialized Vision Processing Unit (VPU) devices (e.g. Movidius Neural compute stick 1 & 2), classification times were shown to decrease by up to 84.5% and 90%, respectively. These findings confirm that high-performance AI is indeed feasible in constrained IoT scenarios without compromising portability.
- **Cross-Disciplinary Impact:** The optimized solutions were successfully applied in various domains—ranging from crowd detection to real-time microplastic detection and pediatric T1D blood glucose prediction. This extensive applicability underscores the relevance of lightweight AI for solving real-world challenges across diverse sectors.

### 1.3.2 Explainability and Interpretability of Models

A second pillar of this work involved enhancing the transparency of AI systems:

- **Feature-Based Knowledge Distillation:** A novel approach was presented for pulmonary disease classification, transferring knowledge from complex networks to more compact models, thus maintaining accuracy and interpretation with a lighter computational approach compared to the state-of-the-art.
- **Explainability in healthcare settings:** Statistical methods for the explanation of the predictive model are applied in a health-related context to optimize treatment strategies and inform preventive measures.

### 1.3.3 Secure and Privacy-Preserving Systems

Finally, the research addressed the need for security and data protection in distributed and resource-constrained environments:

- **Federated Learning with Cryptographic Enhancements:** Through the integration of Zero-Knowledge Proofs (ZKPs) and Distributed Ledger Technology (DLT), the thesis introduced a robust framework for data verifiability and reliability in Federated Learning (FL).
- **Hardware Root of Trust and Distributed Architectures:** The combination of hardware-based security modules with DLT solutions presented with the dRAIN system, ensures auditable and tamper-resistant data management. This approach supports strong security guarantees suitable for real-world IoT scenarios.

- **Privacy-Conscious Biomedical Applications:** Solutions like the triple-regressor FedROS-ELM for Blood Glucose Level forecasting and the Graph-aided Federated Learning (GaFL) framework highlight how decentralized, privacy-preserving methods can deliver clinically reliable predictions. By modeling patient similarity via graph-based clustering, GaFL efficiently balances personalization and data protection, further validating the viability of secure, federated AI in healthcare.

In summary, this thesis advances the state of the art in designing AI systems tailored to conditions where computational resources, interpretability, and privacy must be carefully balanced. By integrating optimization techniques, interpretability frameworks, and strong security measures, the research provides a comprehensive roadmap for deploying intelligent, trustworthy, and resource-aware solutions across a wide range of real-world healthcare applications.

# Chapter 2

# Background

## 2.1 AI in Healthcare

Artificial Intelligence (AI) has emerged as a transformative force in healthcare, providing advanced tools that enhance diagnostic precision, treatment personalization, and operational efficiency. By leveraging large-scale data and sophisticated machine learning algorithms, AI systems can analyze complex datasets to uncover patterns and deliver predictive insights. Applications such as medical imaging, disease forecasting, drug discovery, and patient monitoring are now at the forefront of Healthcare 5.0. However, despite its potential, the integration of AI in healthcare is fraught with challenges that must be addressed to unlock its full capabilities.

### 2.1.1 The Computational Bottleneck

The Internet of Medical Things (IoMT) is transforming healthcare by enabling interconnected devices to perform real-time monitoring, diagnosis, and data analysis. However, a significant challenge lies in deploying advanced neural network architectures on edge devices with limited memory, computational power, and energy resources. These constraints hinder the execution of complex models locally, increasing dependence on cloud-based solutions, which in turn introduces latency and reliability issues. Another critical factor is inference latency. In applications such as continuous patient monitoring or emergency alert systems, delays in inference can have serious consequences. Ensuring real-time responsiveness on resource-constrained devices remains a pressing problem that demands innovative solutions.

To address these challenges, researchers are focusing on two key areas: model compression techniques that optimize neural networks for edge devices, and edge computing frameworks that enable efficient on-device inference. The following subsections explore these approaches in detail.

#### Edge Computing

Edge Computing plays a pivotal role in addressing the integration challenge by reducing reliance on centralized data processing. In IoMT systems, edge devices can locally process and analyze data collected from medical sensors, transmitting only critical insights to central servers. This approach minimizes latency, reduces bandwidth usage, and enhances data security by limiting the exposure of raw patient data [5]. Despite its advantages, implementing Edge Computing in IoMT introduces challenges related to computational efficiency, resource allocation, and energy consumption. To overcome these hurdles, researchers are optimizing edge algorithms for resource-constrained environments, such as wearable devices and remote monitoring systems. For instance, in continuous cardiac monitoring, edge devices can

detect arrhythmias in real-time without transmitting sensitive data, ensuring both timely intervention and data privacy.

### Pruning for Model Compression

Pruning, first introduced by LeCun et al. in [6], is a key technique for optimizing neural networks by removing redundant parameters, such as unnecessary connections or neurons, without significantly affecting model performance. By reducing the size and complexity of the network, pruning minimizes the computational load during inference, making it ideal for resource-constrained IoMT devices.

This technique has demonstrated effectiveness in real-time applications, such as blood pressure monitors and arrhythmia detection systems [7], where rapid and accurate predictions are essential. Structured pruning methods, which target specific elements like entire layers or channels, provide predictable improvements in efficiency while maintaining the functional integrity of the model [8]. These methods also simplify deployment by ensuring compatibility with hardware accelerators optimized for structured computations.

Unstructured pruning, on the other hand, offers higher compression ratios by removing individual weights based on their contribution to the network. Although it can yield more compact models, this approach often requires additional post-processing steps, such as sparse matrix representations, to fully exploit the computational benefits. Regardless of the method, pruned models empower IoMT devices to deliver precise, energy-efficient predictions while reducing latency, which is critical in applications requiring immediate interventions.

### Quantization for Model Compression

Quantization reduces the numerical precision of neural network parameters and activations, transitioning from 32-bit floating-point representations to lower-bit formats such as 16-bit, 8-bit, or even binary. This reduction dramatically decreases memory usage, computational demand, and power consumption, making it particularly well-suited for IoMT devices, such as portable ECG monitors or glucose sensors.

Quantization-aware training (QAT) has emerged as a robust approach to mitigate the accuracy loss typically associated with quantization. By incorporating quantization effects during training, QAT ensures that the resulting models perform nearly as well as their full-precision counterparts. This technique is especially advantageous in medical applications, where even slight accuracy deviations can have significant implications.

Post-training quantization is another widely used method, where a pre-trained model is converted into a lower-precision format. While simpler and faster to implement, this approach may introduce more noticeable accuracy degradation compared to QAT. Advances in mixed-precision quantization, which selectively quantize specific layers or operations based on their sensitivity, strike a balance between model size and performance.

Quantization not only enhances model deployability on constrained hardware but also supports compatibility with hardware accelerators designed for integer arithmetic, further boosting inference speed [9]. Together, these capabilities position quantized models as indispensable components of IoMT systems, enabling real-time, reliable analytics in environments with stringent resource constraints.

### 2.1.2 The Importance of Explanation

The interpretation of results and the explainability of AI models in healthcare represent fundamental pillars for bridging the gap between AI and clinical practice. In a field where the stakes involve human lives, the ability to provide transparent and interpretable insights is not merely an added value but a necessity. Clinicians require not only accurate predictions but also a clear understanding of the reasoning behind these predictions to make informed decisions, trust the models, and adapt treatments effectively.

Explainable Artificial Intelligence (XAI) aims to enhance transparency by shedding light on how AI models arrive at their predictions. Methods like Grad-CAM (Gradient-Weighted Class Activation Mapping) and LIME (Local Interpretable Model-Agnostic Explanations) provide insights into feature importance and model behavior, making these black-box systems more interpretable. These methods offer clinicians a quantitative understanding of the factors influencing a model's output, thereby fostering trust and facilitating integration into medical workflows.

Beyond statistical tools, the design of inherently interpretable systems also plays a crucial role.

#### Grad-CAM: Gradient-Weighted Class Activation Mapping

Grad-CAM [10] (Gradient-Weighted Class Activation Mapping) is a popular technique for visualizing and interpreting the decisions made by deep neural networks, especially in computer vision tasks. It builds upon the concept of Class Activation Mapping (CAM) [11] by utilizing the gradients of a specific target class flowing into the final convolutional layer of a convolutional neural network (CNN). Grad-CAM produces a coarse localization map that highlights the regions in the input image most relevant to the model's decision for the given class.

To generate these class activation maps, Grad-CAM computes the gradient of the class score (logits) with respect to the feature maps of the last convolutional layer. These gradients are then globally averaged to obtain weights, which are subsequently multiplied by the corresponding feature maps. Finally, a ReLU operation is applied to emphasize only the positive contributions of the feature maps. The resulting localization map highlights the pixels in the input image that have a significant influence on the final prediction.

A key advantage of Grad-CAM over other visualization methods is its ability to preserve the spatial information of the feature maps while incorporating class-specific gradients. This allows Grad-CAM to produce class-discriminative and spatially coherent heatmaps, making it particularly suitable for tasks like object detection and fine-grained image classification. Additionally, Grad-CAM can be seamlessly integrated with other interpretability techniques (e.g., Guided Backpropagation) to yield more precise visual explanations. By offering a straightforward and intuitive way to reveal the inner workings of CNNs, Grad-CAM has become a valuable tool for increasing transparency and trust in deep learning models, aiding both developers and domain experts in diagnosing model behavior and identifying potential biases or errors.

#### LIME: Local Interpretable Model-agnostic Explanations

LIME [12] is a versatile technique designed to explain individual predictions of machine learning models. Unlike methods tied to specific algorithms, LIME is **model-agnostic**, meaning it can be applied to any black-box model, regardless of its underlying architecture. LIME approximates the complex model locally around the instance being explained by a simpler, interpretable surrogate model, such as a linear regression or decision tree. This local approach ensures that the explanations are highly relevant to the specific prediction under scrutiny.

The primary advantage of LIME lies in its ability to highlight **local interpretability**, focusing on the contributions of features to individual predictions rather than global model behavior. By perturbing the input data and observing how these changes affect the model's output, LIME identifies the most influential features for a given instance. These perturbations are weighted to ensure that the surrogate model is a faithful representation of the black-box model's behavior in the vicinity of the data point.

The need for interpretability goes beyond technical considerations; it also addresses ethical, regulatory, and psychological dimensions. Transparent models are essential for ethical AI deployment, complying with medical regulations, and fostering user confidence. By making AI outputs understandable and actionable, explainability bridges the gap between advanced technologies and practical clinical applications, as also mentioned by Maccaro et al. [13]. In summary, the integration of explainability into AI systems for healthcare is not merely a technical challenge but a multidimensional endeavor. It ensures that AI can support clinicians in providing patient-centered care while maintaining trust, accountability, and ethical responsibility. As AI continues to advance, explainability remains a cornerstone of its effective and safe adoption in the medical field.

### 2.1.3 The Privacy Dilemma

In the era of Healthcare 5.0, vast amounts of patient-centric data offer unprecedented opportunities for personalized medicine, early disease detection, and predictive analytics [14]. However, leveraging this sensitive information also raises serious concerns about unauthorized access and data misuse, which can lead to identity theft, breach of patient confidentiality, and the erosion of trust in healthcare services [15]. This tension between advancing data-driven healthcare and preserving stringent privacy standards constitutes the *"privacy dilemma."* To address it, researchers and organizations are exploring a range of privacy-preserving technologies, including Federated Learning (FL) which keeps data local while sharing only model parameters—Zero-Knowledge Proofs (ZKPs), and Distributed Ledger Technologies (DLTs). Together, these approaches aim to strike a balance between harnessing the full potential of medical data and maintaining the highest levels of security and compliance within the healthcare domain.

#### Federated Learning

Presented in [16] FL is an emerging machine learning paradigm that offers a novel solution to the challenges of data privacy, security, and regulatory compliance, especially in domains like healthcare, where protecting patient information is paramount. Unlike traditional centralized approaches—which gather vast amounts of raw data into a single repository—FL keeps data localized, with each participating device or

institution training a model on its own dataset and then transmitting only model updates or gradients to a central server. This decentralized framework significantly reduces the risk of exposing confidential information to unauthorized parties and aligns with stringent regulations such as the General Data Protection Regulation (GDPR) in the European Union or the Health Insurance Portability and Accountability Act (HIPAA) in the United States. One of the most widely adopted methods for aggregating locally trained models in this setting is Federated Averaging (FedAvg), a process in which the central server computes a weighted average of the model parameters received from each participant; the resulting global model is then broadcast back to all nodes for the next round of training. This iterative cycle continues until convergence, capitalizing on each participant's locally available data without ever requiring that data to leave the institution where it resides. Nevertheless, implementing FL in a real-world environment presents unique challenges, including heterogeneity in data distributions, which may vary widely among different hospitals, clinics, or wearable sensor devices, differences in network bandwidth, and susceptibility to adversarial behavior. Nevertheless, FL can provide immense benefits to healthcare: enabling the aggregation of diverse clinical information from wearable device measurements, imaging data, and laboratory tests to electronic health records into more robust and generalizable predictive models that can assist diagnosis, personalized treatment plans, or patient monitoring [17].

#### Zero-Knowledge Proof

Zero-Knowledge Proof (ZKP) technology fundamentally relies on transforming the statement to be proven into a well-defined computational representation, often an *arithmetic circuit*, so that the correctness of each operation (e.g., additions, multiplications) can be rigorously verified without exposing any underlying inputs [18]. In practice, this means expressing the function or condition in a gate-by-gate manner, which enables cryptographic techniques (such as polynomial commitments or pairing-based cryptography) to validate the final result without revealing intermediate data. Beyond these classical "arithmetic-centric" ZKP constructions, *garbled circuits*, a secure computation method originally introduced in the context of multiparty protocols, can also be adapted to serve as a foundation for zero-knowledge. In a garbled-circuit-based ZKP, the circuit is "garbled" (or encrypted) so that neither the *prover* nor the *verifier* can glean any private details from its internal structure, yet they can still confirm the correctness of the outcome. Essentially, the prover commits to an encrypted version of the circuit and their inputs, and then interacts with the verifier in a way that convinces the latter that the final output is accurate—without disclosing how each gate was computed or what the inputs were. This dual approach—arithmetic circuits for direct polynomial-based proofs, and garbled circuits for secure evaluation—illustrates the versatility of ZKP frameworks. By employing these techniques, healthcare applications can verify operations on sensitive patient data, such as diagnostic calculations or personalized treatment recommendations, fully confident that no unauthorized party (or even the verifier itself) will gain access to the underlying patient information [19].

#### Distributed Ledger Technology

Distributed Ledger Technology (DLT) provides a decentralized and tamper-proof framework for recording and verifying transactions or data changes. In traditional

blockchain-based systems like Bitcoin and Ethereum, information is grouped into sequential blocks, each cryptographically linked to the previous one, creating a transparent and verifiable chain of records distributed across multiple nodes [20,21]. This inherent distribution limits the influence of any single authority and facilitates consensus among participants, helping to ensure data integrity and resist unauthorized modifications. Beyond standard blockchains, Directed Acyclic Graph (DAG)-based ledgers such as IOTA or Hedera Hashgraph store transactions in a network of interlinked "vertices" rather than in linear blocks, potentially enabling higher scalability and faster transaction confirmation under specific network conditions. Whether blockchain- or DAG-based, DLT solutions also support features like decentralized identity management.

When integrated into healthcare environments, DLT can facilitate secure patient record management, enable transparent sharing of medical test results among authorized parties, and streamline regulatory compliance by providing robust traceability of data access and modifications. Furthermore, tokenization mechanisms can be employed to incentivize data sharing in research contexts, while cryptographic techniques ensure that sensitive information remains confidential. This convergence of distributed ledgers, privacy-preserving AI, and healthcare applications holds the potential to not only strengthen data governance and patient trust, but also to promote innovation in clinical decision-making, telemedicine, and personalized treatment strategies.

### 2.1.4 The case study of Diabetes

Since many sections in this thesis address issues related to the management of glycemic levels in people with diabetes, an introduction to the topic is necessary. Type 1 diabetes (T1D) is a metabolic disorder with an autoimmune aetiology, also referred to as Type 1 Diabetes Mellitus (T1DM). In 2021, there were 6.7 million deaths attributable to diabetes worldwide among individuals between 20 and 79 years of age. This figure represents 32.6% of the total number of deaths among people younger than 60 [22]. If not treated properly, T1D can lead to both short- and long-term complications, including micro- and macro-vascular diseases that can damage the kidneys, eyes, liver, and circulatory system [23]. Although T1D has no cure, it can be managed through daily insulin administrations aimed at keeping the glycemic level in the euglycemic range, i.e., between 70 and 180 mg/dL.

The management of T1D remains a challenge due to its nature, which precludes the implementation of effective and consolidated prevention methods. Consequently, many studies have focused on nutritional, genetic, and immunomodulatory aspects to prevent or delay disease onset [24–26]. While these approaches are promising, their large-scale application remains difficult because of inherent complexity. The conventional treatment of T1D relies on exogenous insulin administration to maintain the correct blood glucose level (BGL), typically between 70 and 180 mg/dL [27]. At present, insulin administration is performed based on the patient's current BGL, thereby introducing potential timing-related complications. Indeed, clinical practice usually involves correcting non-euglycemic situations only after they have reached suboptimal or dangerous levels. A feasible way to address this issue is to implement preventive strategies rather than merely corrective measures.

Two main technologies are employed for BGL monitoring: capillary blood glucose measurement and continuous glucose monitoring (CGM). Capillary measurement provides an accurate snapshot of the current BGL, whereas CGM enables continuous glycemic monitoring over time, with sampling periods of up to one sample per minute. CGM is therefore a powerful tool for preventing undesirable or hazardous glycemic conditions. Moreover, the introduction of predictive methods for glycemic levels in close connection with CGM devices has been shown to improve the quality of life of T1D patients, partly by reducing the fear of hypoglycemic episodes [28]. Despite CGM devices having greatly enhanced disease management [29], frequent hyperglycemic (CGM $> 180\,\mathrm{mg/dL}$) and hypoglycemic (CGM $< 70\,\mathrm{mg/dL}$) events are still reported in clinical practice.

Currently, a major area of T1D research lies in the development of data-driven predictive methods for BGL, with the aim of predicting future levels accurately. Two main predictive approaches are employed:

- *Regression of glucose levels*, which uses capillary or interstitial measurements to forecast the actual glucose value;
- *Classification of glycemic states*, focusing on euglycemia, hypoglycemia, or hyperglycemia (often giving special attention to the onset of hypoglycemia) [30].

An additional subdivision is whether the model uses a univariate or a multivariate dataset. Univariate approaches rely on a single variable (often just glucose itself), whereas multivariate approaches leverage multiple variables. Although multivariate methods have greater intrinsic complexity and require aligning heterogeneous data in time, they may introduce noise and uncertainty that can compromise performance [31]. Several studies comparing univariate and multivariate models have shown that, for BGL regression tasks, their performance is broadly comparable, with only minor differences. In [32], this concept is discussed in detail by comparing three distinct feature sets.

**State of the art in glycemic level prediction** Although numerous physiological-based mathematical models exist for predicting future glycemic levels [33, 34], recent research has mainly shifted toward data-driven models. Machine learning and neural network/deep learning approaches are frequently adopted, with neural networks typically yielding superior results as shown in [35]. Several methods have also been proposed to update model training online, capturing more recent changes in the glycemic trend [36]. A widely accepted metric for evaluating the performance of BGL forecasting is the Root Mean Square Error (RMSE), formally defined in Section 3.4.1. Briefly, the smaller its value, the better the predictive performance.

Within the broader category of machine learning techniques, Bunescu et al. [37] leverage a three-compartment physiological model of blood glucose dynamics to derive features for a Support Vector Regression (SVR), trained on patient-specific data. Their model is validated on data from five T1D patients in a private dataset. Similarly, Georga et al. [38] present a Random Forest regression technique for personalized glucose prediction in T1D. This multivariate approach incorporates CGM data, physiological features, and lifestyle information. In [39], Sparacino et al. propose a first-order AR model with time-varying parameters, estimated at each time step through recursive least squares. They test multiple forgetting factor values with a 30- and 45-minute PH on CGM data of 28 T1D patients from a private dataset, attaining results sufficiently accurate to mitigate critical adverse events.

In the realm of neural networks and deep learning, various well-established architectures have been successfully applied to glucose prediction, sometimes achieving the top performance reported in the literature [40,41]. Some entirely new models have also been proposed for this specific task. Mosquera-Lopez et al. [42] introduce an LSTM recurrent neural network with a corrective module, targeting a 30-minute PH. Training is performed on data from more than 4,000 patients in a private dataset, and tested on an additional 10 patients. Li et al. [43] adopt a recurrent Convolutional Neural Network (CNN) to predict glucose levels for both simulated patients in the UVA/Padova simulator [44] and 10 patients from a private dataset, with 30- and 60-minute horizons. In [32], Butt et al. applied a BiLSTM network to three feature sets (univariate and multivariate), while Martinsson et al. [45] employed LSTM-based RNNs using a univariate approach. Khadem et al. [46] integrated MLP and LSTM predictions within an ensemble framework, and Nemat et al. [47] presented three distinct ensemble learning strategies incorporating Linear Models, VLSTMs, and BiLSTMs, each combined differently and fed into a Meta-Learner. Furthermore, Acuna et al. [48] proposed three approaches of increasing complexity—XGBoost, 1-D CNN, and a Transformer-based model—achieving comparable performance, with the Transformer ultimately outperforming the alternatives.

Overall, while these methods vary in approach and complexity, they share the common goal of improving glycemic control and, ultimately, the quality of life for individuals with T1D. In the quest to balance accuracy, computational efficiency, and data privacy, researchers continue to explore novel architectures, online learning methods, and secure computing environments. The outlook is promising, as technological advancements and large-scale data availability pave the way for increasingly refined models with tangible benefits to T1D management.

# Chapter 3

# AI for Edge devices

Nowadays, the Internet of Things (IoT) is playing an increasingly crucial role in healthcare, thanks to the widespread adoption of smart devices capable of receiving, recording, and transmitting patient data. However, the large amount of data generated by modern healthcare sensors requires appropriate technologies to store and process such sensitive information securely. To address these needs, more and more solutions are turning to AI within IoT infrastructures to facilitate data processing and enhance patient outcomes.

Moreover, the advancement of embedded systems, i.e., combinations of hardware and software designed for a specific function, has led them to represent one of the key solutions in managing healthcare data streams generated by a multitude of monitoring devices [49]. Such systems allow for a decentralized approach, shifting computational tasks closer to the data sources—a concept known as edge computing [50]. Adopting edge computing solutions in healthcare settings enables better scalability of architectures by avoiding excessive reliance on the Cloud and reducing dependence on an always-on Internet connection [51]. Consequently, this leads to lower overhead and power requirements than those typically needed by large data centers. In addition, edge computing strengthens robustness against cyber threats, safeguarding patient data by limiting the amount of information that must travel over networks. This privacy-by-design feature greatly reduces one of the most vulnerable phases for external attacks in healthcare environments.

In addition, edge computing also addresses latency issues by minimizing the distance between data acquisition points and processing nodes. Think of all the control systems in the automotive [52–55] and industrial robotics sectors [56], which require real-time processing; similarly, in healthcare, critical applications such as surgical robotics and acute patient monitoring demand immediate decision-making. A decentralized approach reduces latency in two ways: first, it decreases network traffic, and second, it brings the decision layer physically closer to the data source [57].

At the same time, the growth of deep learning has propelled AI to the forefront, especially in key areas of medical imaging and diagnostics, fueling the need for high-performance edge computing systems that can support cutting-edge Deep Neural Networks (DNN) algorithms. However, the constraints of embedded systems, combined with the computational intensity of certain DNNs, demand careful algorithm selection and optimization to provide real-time patient feedback and maintain high standards of care. Often, even starting from an excellent baseline solution, the need arises to streamline the algorithm, reducing the computational load to enable competitive performance and cost-effectiveness for healthcare applications.

Several tasks exemplify the synergy between AI and embedded systems in healthcare: detecting abnormal events such as sudden patient falls [58], analyzing human

movement patterns [59], managing congestion in clinical facilities, securing wireless medical networks [60], identifying patients through biometrics, leveraging geotextual hierarchical clustering for epidemiological surveillance [61], and monitoring pedestrian flow around hospitals. These examples highlight how combining AI with embedded systems can transform and enhance healthcare services, ultimately improving patient care and operational efficiency.

## 3.1 State of the art

The growing interest in embedded solutions equipped with AI deep learning algorithms has led to the definition of specific compression and acceleration techniques for the prediction phase. The aims are to strictly adhere to the real-time constraints and to decrease the boards' overloads. Below, we present a review of the main approaches and techniques.

As reported in [62], pruning is the first type of model reduction, based on the evaluation of the parameter's importance. This method removes parameters identified as unimportant subject to classification performance constraints and optimizes the reduced Neural Network to maximize performance. Two types of pruning exist: (i) non-structured, in which Neural Network regularity and sparsity are not taken into account resulting in cache and memory access inefficiency; and (ii) structured pruning in which Neural Network regularity is an additional constraint.

The non-structured pruning approaches provide interesting performance, especially in terms of compression. In [63], Han et al. propose an iterative pruning technique for all the weights under a certain threshold after which retrain and optimize the remaining Neural Network. This method reduces memory requirements with a compression factor equal to 9. In [64], Guo et al. propose an approach that they call dynamic Neural Network surgery. This method allows recovering a critical but pruned connection based on a post-cut evaluation. The compression factor of this method reaches 17.7. Eventually, Li et al., in [65], manage to achieve a compression factor equal to 82 by restating the threshold tuning problem into a constrained optimization problem.

The structured approaches have slightly reduced compression gains when compared to the non-structured ones, but they usually couple Neural Network compression with better prediction and acceleration. In [66], Liu et al. use L1 regularization on the scaling factor of the Batch Normalization layers to train compact Neural Networks, reaching a compression factor and a prediction acceleration equal to, respectively, 6.6 and 1.43, without any drop in the top-5 accuracy value. Conversely, ThiNet, presented in [67], prunes filters by using the next layer statistics information instead of the current layer, obtaining 16.63 compression, 3.3 acceleration, and 0.52% top-5 accuracy drop for the VGG-16 net.

A second technique to reduce NNs size is quantization: this can concern weights, activations, or both. Among the weights quantization techniques, the Binary Connect (BC) ( [68]) uses only two possible weights: $-1$ and $+1$. Tests on the ImageNet dataset show a 19.2% top-5 accuracy drop, but with a compression factor equal to 16 for the weights and a significant speed-up due to the complete removal of any multiplication. Moving from BC, the Binary Weight Network (BWN, [69]) better approximates the Neural Network to the starting CNN through a scaling factor, showing an improvement in the top-5 accuracy, which drops only by 6.2%, and slightly worsening in terms of memory and speed. The Binarized Neural Network (BNN, [70]) extends quantization to both weights and activations. It shows a 29.8% top-5 accuracy

drop and a drastic decrease in memory usage. Similarly, the XNOR-Network [69] is an extension of BWN: it leverages the binary operations to approximate convolutions, achieving a 58 speed up in classification times and top-5 accuracy drops equal to 11% and 16% for AlexNet and ResNet-18, respectively.

A third Neural Network reduction technique is model distillation, whose aim is to transfer knowledge from a larger Neural Network (teacher) to a smaller distilled one (student). This approach finds a first excellent application in the Noisy Teacher, where Sau et al. ( [71]) propose a noise-based regularizer to improve the performance from 99.03% (teacher) accuracy to 99. 14% (student) in the MNIST dataset. In another study, Romero et al.( [72])propose FitNets to train a thinner and deeper model on CIFAR-10 dataset. The student net reaches 91.63% accuracy from 90.18% of its teacher, decreasing from 9 to 2.5 million parameters.

Other techniques require the implementation of specific Neural Network strategies, such as the application of depthwise and pointwise convolutions, such as [73] on MobileNets, and filters size and number reduction, such as [74] on SqueezeNets. Lastly, the low-rank factorization technique is used to accelerate NNs by finding an approximate low-rank tensor (or matrix) close to the starting tensor and easier to decompose [75,76].

**The case of Diabetes management:** normally, machine learning techniques are validated on laboratory setup, and, when they are applied in practice, they are performed directly on servers or centralized processing units. The task of future glycemic levels prediction makes no exception, as most systems that perform real-time prediction exchange data between an edge device, only used to gather information, and the cloud, where the actual glucose level forecasting is performed [77,78]. This is mainly due to the memory limits of edge-computing devices. Nonetheless, the drawback of such systems is that they constantly require an internet connection to work; this is not arguable with regards to medical devices, because an interruption in the signal may result in missing decision support to the user. However, the increasing development of new, more powerful and dedicated hardware, combined with the widespread use of IoT (Internet of Things) tools, is enabling the emergence of a branch of AI known as inference at the edge [79,80]. This involves the machine learning models being run directly from a proximity device using data collected from associated sensors. Taking into account also the increasingly telemedicine-oriented approach [81,82], it is clear that the possibilities given by inference at the edge can be exploited to create predictive models that work in real-time with patient data to both improve the quality of life of patients and increase the ability of the physicians to extract useful information from the sensor data. Compared to systems that run on the cloud, edge computing can provide more reliable real-time service, with low latency, and they are not limited by internet connectivity.

## 3.2 Datasets

In this section, a brief overview of the datasets employed in the subsequent analyses is provided. Each dataset will be described in terms of its origin, scope, and key characteristics. Presenting these datasets at the outset will help establish a clear context for the analyses to follow and underscore their relevance to the research questions at hand.

**Face mask dataset** Consisting of 1406 images divided equally into 2 classes. One of the key resources utilized in this study is a private, request-access dataset specifically constructed for the analysis of masked versus unmasked facial images. Introduced here for the first time, this dataset consists of 1406 images evenly divided into two classes: "with face mask" and "without face mask" (see Figure 3.1). Notably, it is generative in nature: the masked images were produced by accurately overlaying a mask onto the original unmasked images. Furthermore, the dataset encompasses images of varying dimensions ranging from 222x124x3 to 1258x1024x3 pixels thereby reflecting a level of heterogeneity that more closely aligns with real-world conditions.

FIGURE 3.1: Images belonging to the two classes of the dataset.

**Microplastic dataset** A synthetic dataset has been generated according to [83]. The data simulate the signals generated by the particle traversal through the measurement system over time, approximating them as Gaussian functions. Each Gaussian function within the dataset is characterized by distinct parameters such as mean and standard deviation mapped to capture the variability in the sizes of microplastics that may be present in the flow. We added noise to the signal to simulate potential measurement interference, mimicking real-world conditions in which extraneous factors can affect the accuracy of the readings. The signal depends on several parameters: the signal amplitude variation ($\mathcal{A}$), the signal-noise ratio (SNR), and the a priori probability ($P_1$) of finding the particle in the signal. Each signal has a length of 41 samples and is labeled with 0 or 1, indicating the presence or absence of the particle. Each realization of the dataset is composed of 1000 signals resulting in a matrix of shape $1000 \times 41$ and a ground-truth vector with a dimension $1000 \times 1$. In this paper, all the generated signals have SNR equal to 0.05, $P_1 = 0.5$ and $\mathcal{A} \in [0.6, 1.4]$, by varying these parameters, it is possible to reconstruct a realistic range of particle sizes that reflects the natural heterogeneity of microplastics.

**Simulated patients dataset** Data were produced for 10 pediatric patients by running several simulations in the UVA/Padova simulator [44]. Such a tool allows to generate different scenarios for *in silico* patients by only providing a meal schedule. For each day of simulation, we considered a baseline 5-meal schedule including 45, 20, 70, 20, 80 $g$ of carbohydrates ingested at time 8:00, 10:30, 13:00, 17:00, 20:00, respectively. To make the simulation more realistic, each meal time was shifted by an amount of time randomly chosen from a uniform distribution of $\pm 60$ minutes, whereas each amount of ingested carbohydrates was randomly modified by an amount of grams taken from a uniform distribution of $\pm 20\, g$. The simulator is able to determine the optimal insulin boluses to be injected for each meal of a specific patient, and can thus provide the glycemic evolution for each subject for a preset number of days. However, the tool allows the user to modify the insulin bolus

value, and to include a sensor error in the CGM readings. Data are generated with a 1-minute sampling.

Two different datasets were generated on a scenario consisting of 30 days of simulation with 5 meals per day. The first scenario has no errors in sensor reading and insulin administration, as automatically computed by the simulator, and thus corresponds to an ideal T1D management. Differently, we created the second scenario using the same meal schedule as the first scenario, but by including CGM sensor errors and by forcing the presence of hyperglycemic and hypoglycemic events. We were able to achieve such a goal by first allowing the UVA/Padova simulator to run a simulation with its own optimal bolus control; then, we extracted the vector of injected boluses and added random noise taken from a uniform distribution. In particular, each bolus consisting of $I$ insulin units was modified according to the equation 3.1:

$$\hat{I} = I + z \tag{3.1}$$

where $z$ is a random value taken in the interval $[-3, 3]$. In practice, each bolus was increased or decreased by no more than 3 units of insulin from its optimal value. The modified bolus vector was given as effective bolus vector to the UVA/Padova to run the simulations for this scenario. This makes such a scenario more realistic, because in real life the increase or decrease in blood sugar levels occurs mainly due to an inaccurate estimate of the amount of carbohydrates ingested, or to deviations in correction dosing [84]: we added noise on insulin boluses to simulate human error.

The datasets consist of information on blood glucose levels and data on insulin (bolus, basal, and injection were added together and considered as a one) and finally carbohydrate intake. Specifically, the final datasets consider Insulin-On-Board (IOB) as insulin feature, which was manually generated by exploiting a mathematical model [85]. IOB is a quantity referred to the amount of rapid-acting insulin still active in the patient's body after a bolus injection, and thus provides deeper information on the recent history of insulin injections compared to the punctual insulin values themselves. The range of time for considering insulin still active is roughly between 2 and 8 hours [86]. IOB is estimated differently among the main insulin pump companies, but in all cases its calculation is based on insulin action plots which forecast the percentage of residual insulin as a function of time. For the Insulet pump, which is the one we selected when using the simulator, the active insulin time is equal to 3 hours and the shape of insulin action plot is linear [85]. Thus, the value of IOB for each timestamp $t$ was computed as:

$$IOB(t) = \sum_{i=0}^{179} \alpha(i) u(t-i) \tag{3.2}$$

where $u(t-i)$ represents the insulin injection at timestamp $t-i$, and $\alpha(i) = 1 - i/180$ is the coefficient corresponding to the insulin decay curve. It is worth noting that only past insulin values (i.e., corresponding to timestamps $\leq t$) are used to compute the $IOB$. Specifically, 100% of the latest insulin injection value contributes to $IOB(t)$, whereas the contribution linearly decreases to 0 for older values in the previous 3 hours. Straightforwardly, the first 3 hours of data of each patient were not used to train the predictive models, as they were used to initialize the IOB values. A graphical example of 5 days of data concerning the CGM sensor reading, the ingested carbohydrates, and the IOB of a sample patient generated with a 1-minute sampling using the simulator and the pre-processing are reported in Figure 3.2.

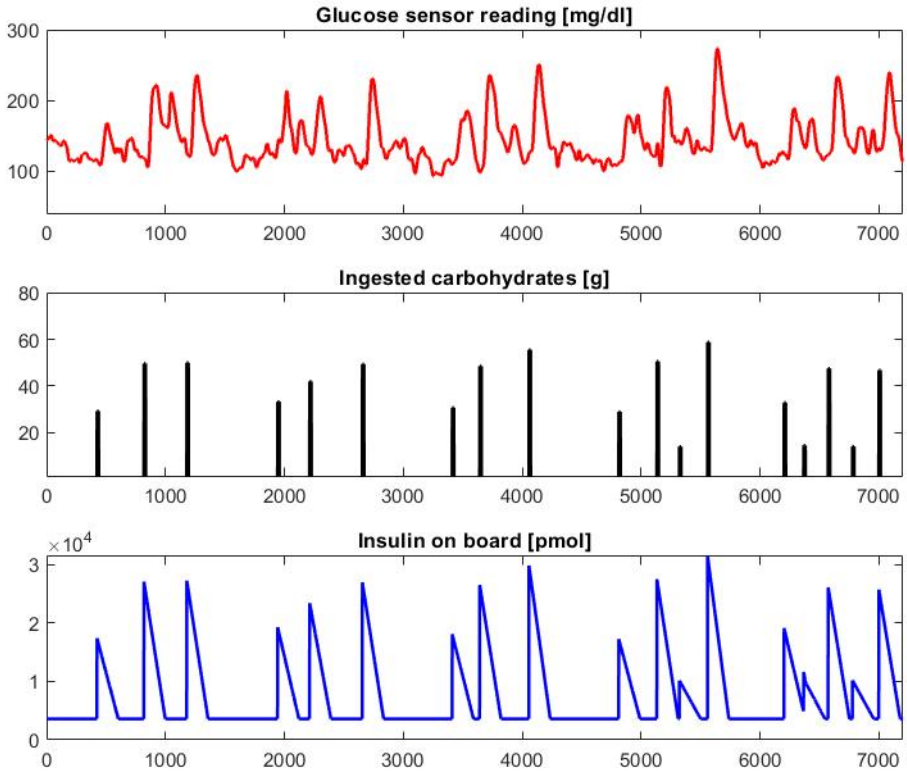


FIGURE 3.2: Graphical example of 5 days of data generated for patient child#007. Many hyperglycemic ($G > 180\,mg/dl$) values can be observed due to the modification of the optimal bolus values.

## 3.3 A Practical Approach to the Analysis and Optimization of Neural Networks on Embedded Systems

To overcome the limitation presented in the section 3.1 we have implemented a standardized optimization method for porting NN to embedded devices, which is valid regardless of the AI classification task, algorithm pipeline, and board type [87]. The universality of the proposed method stems from the assumption that there is no NN suitable to solve any classification problem and different NN adapt differently to different microcontrollers. In particular, our method is based on three steps:

1. Best trade-off step: it helps to search for the best NN solution between different pipelines for a specific board in use.
2. Optimization step: starting from the NNs identified in the previous step, it evaluates different optimization techniques searching for further improvements.
3. Explanation step: evaluates the classification criteria correctness for the task at hand with LIME interpretability algorithms.

The application task to pragmatically utilize the proposed method is the detection of crowded states. Historically, this problem was approached with regression [88], detection [89–91], or density estimation methods [92], but the best results in terms of performance have been recently obtained with the advent of convolutional Neural Networks [93,94]. To date, the state of the art related to the crowd counting problem has always focused unilaterally on performance aspects, while few edge implementations emerge.

### 3.3.1 Microcontrollers and AI integration

Unlike Personal Computers, typically, microcontroller boards rely on light architectures, tuned to perform fewer and application tailored-tasks. This is due to size and power constraints that translate into memory and computational limits. Indeed, microcontrollers usually have: (i) small Flash memory to host data (few MBs), (ii) small RAM memory (few hundreds kBs), (iii) reduced clock speeds (hundreds of MHz), and (iv) small data bus width (8/16 bits). An important aspect to consider is the power-to-performance ratio (typically in the tens of $\mu$W/MHz range). Indeed, power consumption and processing power are both critical factors for algorithm designers, especially when the goal is to avoid the overloading of embedded systems. From this point of view, it is important to have a trade-off between power consumption and algorithms complexity.

The offline deployment of NNs on embedded systems requires a two steps process: first, it requires defining and training the NN on a Workstation and, then, tuning the trained Neural Network on the chosen microcontroller. The design must take into account the capabilities of the physical device on which the Neural Network will be deployed. The constraints to deal with are the following:

$$\mathrm{ROM}_{\mu\mathrm{C}} > \mathrm{Parameters}_{\mathrm{NN}} \tag{3.3}$$

$$\mathrm{RAM}_{\mu\mathrm{C}} > \mathrm{Activations}_{\mathrm{NN}} \tag{3.4}$$

$$\frac{\mathrm{NNcomplexity}_{\mathrm{FLOPS}}}{\mathrm{MCU}_{\mathrm{FLOPS/s}}} < \frac{\text{samples in a window}}{\mathrm{f}_{\text{sensor sampling rate/s}}} \tag{3.5}$$

Equation 3.3 constrains the number of NN parameters to load in the available $ROM_{\mu C}$ storage capacity of the read-only memory of the microcontroller. Equation 3.4 limits the number of activations and depends on the amount of available RAM. When managing NNs, the layers that impact the most on RAM are the convolutional ones. Hence, a trade-off arises between classification performance, which depends mostly on the number of convolutional layers, and time performance, which may suffer the memory overload. Equation 3.5 represents the constraint for real-time applications: the ability to process the data on time and without losses.

Above inequalities can be rewritten as:

$$ROM_{\mu C} > Weights_{NN} \tag{3.6}$$

$$RAM_{\mu C} > Activations_{NN} \tag{3.7}$$

$$T_{inference} < T_{Acquisition} \tag{3.8}$$

It is worth noting that these constraints depend exclusively on the NN architecture. When porting Neural Networks on embedded systems, however, we also need to consider additional burdens, such as the memory needed in pre- and post-processing phases (input and output Neural Networks buffers) and the execution of other concurrent programs. These constraints, combined with the growing need for edge implementations, have increasingly pushed the search for high-performance optimization solutions. However, as often happens, optimization leads to an even greater degree of abstraction, leading to a trust decline that users pour on models. For this reason, it is necessary to combine the explainability of the model with the optimization process.

In the next sections, we present the edge system, the Neural Network, and the application chosen for the study.

**Edge System** In this work, we choose the STM32F767ZIT6U to develop our embedded system prototypes. We based our choice on the presence of a state-of-the-art microcontroller with excellent processing capabilities, and wide availability of peripherals for industrial applications with sufficient memories for NN porting and execution. The main characteristics of the device are:

- 2 MB of Flash Memory;
- 512 kB of static RAM;
- Max $f_{clock}$ = 216 MHz.

In the study, we set the maximum clock speed in all the tests.

We used STM32CubeIDE for software development and to interact with the device. STM32CubeIDE is an open-source IDE and toolchain system which allows, among the others, to inspect ST Microelectronics $\mu$Cs and assess in advance, before code assembly, portability issues related to memory requirements. For each supported Neural Network, we evaluated its main deployment characteristics: ROM, RAM, Multiply and Accumulate operations (MACs), and quantity of parameters. We dissected the analysis of the total RAM into three components: input, Neural Network, and output RAM. These components refer to the Neural Network input layer, the hidden layers, and the Neural Network output layer, respectively. We trained and optimized all the models considered in this work on a workstation. We verified NN instances on both the embedded system and the workstation to obtain a deep insight into the parameters and the configurations impact on performance.

**Neural Networks** In this work, we focused on MobileNets [73], whose full model is reported in Table 3.1. This Neural Network class implements an architecture with highly efficient convolutional layers called depthwise separable convolutions, and this makes them highly optimized for embedded and mobile applications.

TABLE 3.1: Complete MobileNet architecture for binary classification. Dw Conv and FC indicate depthwise convolutions and fully connected, $\alpha$ the width multiplier, and $\rho$ the resolution multiplier.

| | **Type/Stride** | **Filter Shape** | **Input Size** |
|---|---|---|---|
| | Conv/s2 | $3 \times 3 \times 3 \times 32 \cdot \alpha$ | $224 \cdot \rho \times 224 \cdot \rho \times 3$ |
| | Dw Conv/s1 | $3 \times 3 \times 32 \cdot \alpha$ | $112 \cdot \rho \times 112 \cdot \rho \times 32 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 32 \cdot \alpha \times 64 \cdot \alpha$ | $112 \cdot \rho \times 112 \cdot \rho \times 32 \cdot \alpha$ |
| | Dw Conv/s2 | $3 \times 3 \times 64 \cdot \alpha$ | $112 \cdot \rho \times 112 \cdot \rho \times 64 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 64 \cdot \alpha \times 128 \cdot \alpha$ | $56 \cdot \rho \times 56 \cdot \rho \times 64 \cdot \alpha$ |
| | Dw Conv/s1 | $3 \times 3 \times 64 \cdot \alpha$ | $56 \cdot \rho \times 56 \cdot \rho \times 128 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 128 \cdot \alpha \times 128 \cdot \alpha$ | $56 \cdot \rho \times 56 \cdot \rho \times 128 \cdot \alpha$ |
| | Dw Conv/s2 | $3 \times 3 \times 128 \cdot \alpha$ | $56 \cdot \rho \times 56 \cdot \rho \times 128 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 128 \cdot \alpha \times 256 \cdot \alpha$ | $28 \cdot \rho \times 28 \cdot \rho \times 128 \cdot \alpha$ |
| | Dw Conv/s1 | $3 \times 3 \times 256 \cdot \alpha$ | $28 \cdot \rho \times 28 \cdot \rho \times 256 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 256 \cdot \alpha \times 256 \cdot \alpha$ | $28 \cdot \rho \times 28 \cdot \rho \times 256 \cdot \alpha$ |
| | Dw Conv/s2 | $3 \times 3 \times 256 \cdot \alpha$ | $28 \cdot \rho \times 28 \cdot \rho \times 256 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 256 \cdot \alpha \times 256 \cdot \alpha$ | $28 \cdot \rho \times 28 \cdot \rho \times 256 \cdot \alpha$ |
| $5\times$ | Dw Conv/s1 | $3 \times 3 \times 512 \cdot \alpha$ | $14 \cdot \rho \times 14 \cdot \rho \times 512 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 512 \cdot \alpha \times 512 \cdot \alpha$ | $14 \cdot \rho \times 14 \cdot \rho \times 512 \cdot \alpha$ |
| | Dw Conv/s2 | $3 \times 3 \times 512 \cdot \alpha$ | $14 \cdot \rho \times 14 \cdot \rho \times 512 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 512 \cdot \alpha \times 1024 \cdot \alpha$ | $7 \cdot \rho \times 7 \cdot \rho \times 1024 \cdot \alpha$ |
| | Dw Conv/s2 | $3 \times 3 \times 1024 \cdot \alpha$ | $7 \cdot \rho \times 7 \cdot \rho \times 512 \cdot \alpha$ |
| | Conv/s1 | $1 \times 1 \times 1024 \cdot \alpha \times 1024 \cdot \alpha$ | $7 \cdot \rho \times 7 \cdot \rho \times 1024 \cdot \alpha$ |
| | Avg Pool/s1 | $7 \cdot \rho \times 7 \cdot \rho$ | $7 \cdot \rho \times 7 \cdot \rho \times 1024 \cdot \alpha$ |
| | FC/s1 | $1024 \cdot \alpha \times 2$ | $1024 \cdot \alpha \times 2$ |
| | Softmax | Classifier | $1 \times 1 \times 2$ |

DepthWise separable convolutions make use of two hyperparameters with a value in the range [0, 1], the width multiplier $\alpha$, and the resolution multiplier $\rho$. The former controls the number of channels or channel depth, whereas the latter controls the input image resolution. We trained the MobileNets in full and shallow versions, where the shallow version reduces the number of consecutive convolutional layers from 5 to 1, varying their characteristic parameters $\alpha$ and $\rho$, and considering only RGB images.

**Application Choice** The method presented in Section 3.3.2 will also evaluate the algorithm's performance aspects. Therefore, it is necessary to choose a practical application as an example, without which it would be possible to analyze only the computational parameters. However, we emphasize the method's independence from the application. The purpose of the section is to define an approach and not to provide a specific case study.

The application example refers to crowded and uncrowded state classification in video surveillance contexts and social monitoring. Two main reasons have led to this choice: the first is related to porting since the classification task represents a versatile approach for a computer vision problem in an embedded environment.

The second is an interpretability reason: treating crowd detection as a classification task can conduct different Neural Network learning logics. LIME, which is the method's third part, can highlight these training phenomena and hopefully explain some Neural Network performance behaviors. Relating to the example state of the art the best results in terms of performance have been recently obtained with the advent of convolutional Neural Networks [93, 94].

According to this, to address the problem we trained, validated and tested several MobileNets versions on Fudan Shanghai Tech dataset [93]. The dataset includes 15,000 samples of 100 different scenes between indoor and outdoor environments, randomly divided into two sets: a training set of 9000 samples and a test set of 6000 samples. The criteria behind this particular dataset choice were the following:

- High number of samples;
- Average people per image consistent with typical values for indoor and outdoor workplace video surveillance systems;
- Excellent images starting resolution;
- Images from different contexts being able to represent a factor in favor of net generalization ability in the prediction phase.

Concerning the training phase, the classifier's state division threshold was set to 20 people, maintaining a conservative approach between scene volumes and instances quantity for the image. Therefore, an unbalanced binary problem has been addressed, with a training set consisting of 2230 samples belonging to class 1 (crowd less than 20 people) and 6770 of class 2 (crowd greater than 20 people).

### 3.3.2 Compression Method

The propose method is divided into three steps. Firstly we present the best trade-off step. In this phase, information is extracted from the NN, e.g., RAM, ROM storage memory, and compared with the constraints due to the device itself. Secondly the optimization step is explained. In this step, we consider three different approaches: regularization, pruning, quantization, and their possible combined use, so that the computational burden can be reduced and the inference time can be improved without degrading the goodness of the models. Finally, we present a possible approach to explain the logic behind model decisions, an operation that becomes even more important as a result of the network modifications made in the optimization step.

**Best Trade-Off Step** In the first of the method three steps, ROM and RAM memories, MACs operations, and the number of parameters were extracted for each NN structure. By referring to the first two equations discussed in Section 3.3.1, Neural Networks that do not fit the board's limits in terms of total RAM, ROM, or both, were removed.

Once obtained, the only implementable ones, to assess both performance and computational burden factors, the Neural Network choice was based on the best trade-off between accuracy, $T_{inference\,board}$, and loss factor $\beta$.

To burst these parameters, we conventionally indicate as Py-Net the NN implemented in Python on the workstation, whereas we indicate as C-Net the NN loaded on board and so implemented in C language.

As accuracy values, Py-Net performances on the entire test set were used, since, in the preliminary assessment, the relative error-index L2 between the Py-Net (Full

Neural Network) output layer and the C-Net one never exceeded the critical value of 0.01 (cross-validations based on external test sets always returned 100% accuracy). Relating to the example, to achieve better accuracies and address the unbalanced problem, the class weights method was applied during the training phase, correlating weights to sample distribution, as in Equation 3.9.

$$Class_i\, weight = \frac{1}{N\, classes} \times \frac{Total\, samples}{Class_i\, samples} \tag{3.9}$$

Concerning the other two best trade-off parameters, to understand networks behaviors either on a single image or on streams of images (up to 100 frames), different smaller random test sets of 1, 10, 50, and 100 images were created. When a test set shows multiple images, every frame has its time inference. Over these test sets, the $T_{inference\, board}$ in terms of means ($\overline{T_{inference\, board}}$ [ms]) and standard deviations ($\sigma_{\overline{T_{inference\, board}}}$ [ms]) was analyzed. Mean and standard deviation values were carried out for each implementable NN on all test frames and on all test sets. According to this, we expressed $T_{inference\, board} = \overline{T_{inf\, board}} \pm \sigma_{\overline{T_{inf\, board}}}[ms]$, $T_{inference\, pc} = \overline{T_{inf\, pc}} \pm \sigma_{\overline{T_{inf\, pc}}}[ms]$, and the loss factor $\beta = \frac{T_{inference\, board}}{T_{inference\, pc}}$ as $\beta = \overline{\beta} \pm \sigma_{\overline{\beta}}$.

In this work, we focused on RGB images to better exploit database characteristics and the capabilities of the chosen LIME explainers. However, a similar investigation also on grayscale images is conducted in [95].

Two more steps refine the best trade-off. The former is a canonical optimization step based on regularization, pruning, and quantization techniques to improve Neural Network speed and system resource utilization. The latter aims to highlight the superpixels that mostly influence the classifier decision-making for each image, and quantify the global goodness through the Intersection Over Union parameter.

**Optimization Step** In this step, the aim was to investigate the impact of several mechanisms to improve Neural Network performances and understand the sensitivity of parameters to the different optimization techniques. To obtain a proper real-time application, both performance and computational parameters were analyzed: accuracy and time inference represent performance metrics, whereas RAM, ROM, and MACs represent computational metrics. Furthermore, we analyzed the impact of the number of the Neural Network on all metrics.

Among all the possible ways to optimize a Neural Network [62], three different approaches were considered:

- Regularization;
- Pruning;
- Quantization.

Regularization and Pruning were executed in parallel, then the resulting nets from the previous optimization steps were quantized as follows below. The first approach is related to specific regularization attempts applied on the best trade-off to enhance performances. Specifically, the three considered regularization attempts are:

1. Attempt R1: addition of a L2 regularization penalty = 0.0005 on layer 17 ("dense" with kernel regularizer).

2. Attempt R2: addition of L2 regularization penalty = 0.001 on layer 17 ("dense" with kernel regularizer).

3. Attempt R3: addition of L2 regularization penalty = 0.005 on layer 17 ("dense" with kernel regularizer) and L2 regularization penalty = $1 \times 10^{-5}$ on layer 15 (last depthwise convolution with activation regularizer on reLUs).

For R1, the L2 value equal to $5 \times 10^{-4}$ on weights has been chosen by considering Alex Krizhevsky, et al. [96] observation, according to which small-weight regularizations for convolutional Neural Networks act not only as such but even as learning improvers. Additionally, Simonyan et al. [97] used $1 \times 10^{-5}$ as weight decay to develop a CNN for the ImageNet dataset, founding this a promising quantity. Following Reed et al. [98], a 0.001 value has been chosen and selected between the good tests for R2. Finally, R3 includes both weight and activation L2 techniques and was dictated by the positive result.

The second approach is a pruning method that deletes sequentially, first, and then simultaneously the Neural Network's bottlenecks in terms of time inference. Indeed, we profiled layer-by-layer different parameters, among which also the time inference. Five pruning attempts have been performed:

1. Attempt P1: the value stride is equal to 2 on layer 15, used in all subsequent tests and intended to lighten the dimensional change on pooling.

2. Attempt P2: removal of layers 9–10.

3. Attempt P3: removal of layers 5–6.

4. Attempt P4: removal of layers 5–6 and 9–10.

5. Attempt P5: removal of layers 0–1, 5–6, and 9–10.

The last approach is an 8-bit quantization which aims to evaluate how pixels, weights, and layer outputs representation reduction from 32 to 8 bits influences the global and layer-by-layer effectiveness in terms of all parameters. In practice, the quantization has been applied on all the Neural Networks resulting from the pruning and regularization. This approach required different steps including:

- The creation of a Keras "quantization aware model" from the starting Neural Network, to partially rebuild Neural Network architecture in order to facilitate the following steps.

- The training and validation of a quantization-aware model. It is worth noting that this Neural Network topology is not quantized yet.

- The conversion from Keras retrained quantization aware model (.h5) to the TensorFlow Lite model (.tflite).

- The evaluation of TensorFlow Lite model times and results.

We have to observe that quantization and pruning have a unique strategy approach, passing from 32 to 8 bits and starting from layer-by-layer analysis, respectively. Differently, regularization depends strictly on the application, so even if it is included as a technique within the method it does not have a unique way of implementation. As mentioned above, the method's second step, composed of all the techniques discussed above, is essential to improve the real-time application in

edge trying to minimize any kind of performance loss. It represents a reference step particularly significant to all the borderline cases where a starting Neural Network cannot be ported to the edge because of its excessive memory usage, allowing one to estimate and assess the goodness of alternative solutions.

**Explanation Step** There are several approaches to explain the logic behind model decisions. In this study, Local Interpretable Model-Agnostic Explanation (LIME) [99] has been used. This algorithm processes the image to be explained and the model prediction on this image. Then, it divides the image into superpixels, and analyzing the local decision boundary, returns as output the most influencing superpixels in conditioning the decision in a specified class. In practice, the algorithm's output is a binary mask of important superpixels.

In order to better understand the optimization steps impact, the LIME interpretability algorithm was exploited, applying it to the same 500 test samples used in the accuracy evaluation. The LIME algorithm creates an explainer for each image, and it returns an explanation for the specific frame. This latter is a mask that identifies the most relevant superpixels of the frame, those contributing the most to the correct classification outcome. From these masks, we computed the IOU (Intersection Over Union) coefficient for each test image, processing the masks of the original NN and that of the simplified one. After that, we investigated if accuracy and IoU trend lines correlate to demonstrate whether the optimization process affects the decision criterion.

### 3.3.3 Results and Discussions

**Best Trade-Off** Table 3.2 reports the behavior of every RGB MobileNet in terms of the principal investigation parameters. A complete analysis is shown in [95]. Among all these nets only two (described in black on Table 3.2) are portable on the chosen edge device, respecting all memory constraints. Hence, the best trade-off choice between these is represented in Table 3.3: the two accuracy values are similar, with a small superiority of the deep model, but shallow structure shows, simultaneously, a time inference of $145.38 \pm 0.740$ [ms] and a $\beta$ factor of $2.213 \pm 0.070$ lower than those demonstrated by the same deep version. The mandatory hard real-time constraint guided the shallow model as the best trade-off for this study, despite the small performance leak.

TABLE 3.2: Computational and performance aspects of the tested RGB MobileNets.

| Neural Networks | Parameters | ROM [kB] | RAM [kB] = IN + NET + OUT | MACs | Acc [ADIM] |
|---|---|---|---|---|---|
| Deep 128 $\alpha$50 | 819,618 | 3.13 MB | 733.45 = 196.81 + 536.83 + 8B | 49,486,206 | 0.880 |
| Shallow 128 $\alpha$50 | 475,811 | 1.82 MB | 733.45 = 196.81 + 536.83 + 8B | 27,611,533 | 0.774 |
| Deep 160 $\alpha$25 | 213,586 | 823.63 | 724.62 = 307.20 + 417.41 + 8B | 21,456,078 | 0.822 |
| Deep 128 $\alpha$25 | 213,586 | 834.32 | 465.03 = 196.61 + 268.41 + 8B | 13,733,070 | 0.812 |
| Shallow 160 $\alpha$25 | 123,346 | 481.82 | 724.62 = 307.20 + 417.41 + 8B | 12,558,798 | 0.841 |
| Shallow 128 $\alpha$25 | 123,346 | 476.13 | 465.03 = 196.61 + 268.41 + 8B | 8,038,350 | 0.800 |

TABLE 3.3: Best trade-off method application over the two possible RGB choices.

| Neural Networks | Acc [ADIM] | $\overline{T_{inf}} \pm \sigma_{\overline{T_{inf}}}$ [ms] | $\overline{\beta} \pm \sigma_{\overline{\beta}}$ [ADIM] |
|---|---|---|---|
| Deep 128 $\alpha$25 | 0.812 | $430.911 \pm 1.468$ | $32.823 \pm 2.398$ |
| Shallow 128 $\alpha$25 | 0.800 | $285.873 \pm 0.728$ | $30.610 \pm 2.328$ |

**Optimization** We analyzed and investigated parameter behavior layer by layer. This was helpful to understand on which layers RAM and ROM memories reside and where MAC operations are carried out in greater numbers, but especially to denote the best trade-off time inference layers bottlenecks. Figure 3.3 illustrates this latter aspect: the horizontals bars represent the layers times inference on board from the first C net layer (with "0" index) to the last nineteenth (with "18" index), compiled from up to down on the right y-axis. The left y-axis specifies the type and output dimensions of each net layer: the first layer type is a 2D standard convolution with $(64 \times 64 \times 8)$ filters, while the last layer type is non-linearity and its output is a double neuron as binary classification expects.

In the Optimization Step section, specific pruning tests have been listed. The reason for these specific executions can be explained by observing figure 3.3 itself. Firstly, confirmation takes place: the layer time inference strictly depends on input and output dimensions and input and output filter quantity. However, there is another distinguishing factor: with these parameters being equal, the depthwise convolutions demonstrate their higher efficacy concerning the standard convolutions. This is visible by comparing the time's inference of layer 0 (standard) with layer 1 (depthwise) or layer 5 (depthwise) with layer 6 (standard) and 9 (depthwise) and 10 (standard). These same layers couples represent the principal dimensionality redundancies of the best trade-off MobileNet structure. In this way, the executed pruning steps not only have focused on the critical time inference net bottlenecks but even on the layer with feature maps dimensionality already present in the architecture. However, the same approach has not been implemented for layer 2, where no redundancy was highlighted. The last remaining peak was represented by layer 16: to reduce successfully the pooling impact over time, the strides of layer 15 have been increased from $(1 \times 1)$ to $(2 \times 2)$.

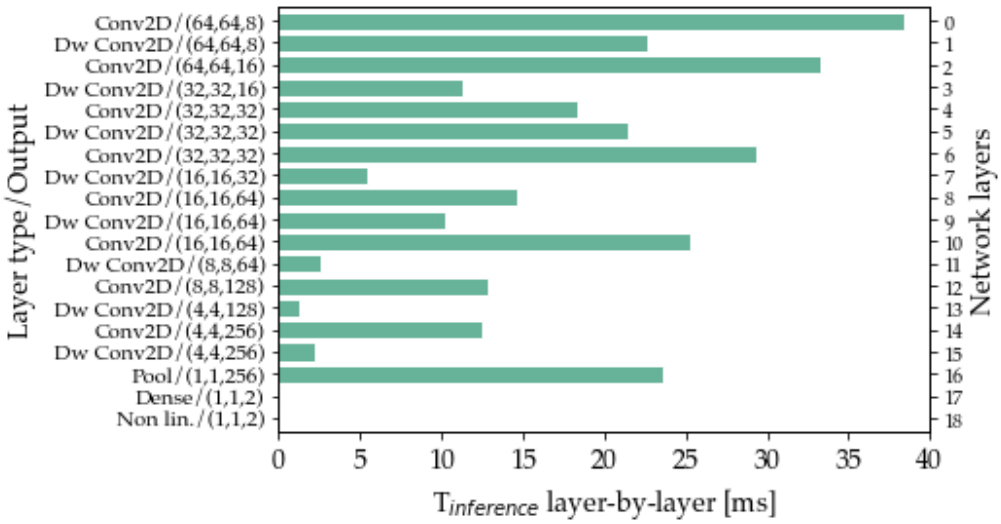


FIGURE 3.3: RGB Best trade-off: time inference layer by layer.

Figure 3.4 helps to understand how each optimization technique improves several study parameters. Except for accuracies, that have values already $\in [0, 1]$, each of these is normalized respect to all the 16 tests executed (8 of Keras and 8 of tflite) plus the best trade-off, namely starting point. The first key to the reading of this Figure 3.4 regards the pruning tests. The first five blue charts highlight remarkable progressive savings in terms of time inference and MAC operations, which reach

61.9% and 62.6%, respectively in P5, with a decrease of 6 C layers on the structure. Memory remains almost invariants to pruning applications, as well as accuracies evaluated on a random portion of 500 elements go from 0.832 of the starting point to 0.842 of P5, highlighting performance improvement in parallel to that computational. The three R1–R3 regularization tests have been carried out by adding definite values of L2 regularization following some state of art experiments and reporting only the interesting results among all the executions. In the study case, the weights regularization has been applied on dense layer 17, while the activations regularization on layer 15. The performances of these three tests are 0.822, 0.816, and 0.796, with respect to the 0.800 start accuracy on all the 6000 test set images. None of the computational parameters has been worsened by this technique, as can be seen on last three blue charts of Figure 3.4.

A further reading key of Figure 3.4 is represented by the "standard" and "quantized" charts. The gain on parameters, such as MACs, RAM, and ROM memories, and time inference, for both pruning and regularization techniques, is visible. Table 3.4 quantifies each variation in terms of means and standard deviations, normalized to the parameter's maximum values over the whole of the represented tests. Remarkable mean decreases of 73.3% and 43.2% are highlighted by ROM and RAM values, respectively, that establish very low variances over the tests. MAC operations meanly enhance by 29.3%, but most of all 8-bit quantization reduces tests time inference by $50.9 \pm 13.4\%$, by meanly losing only the 0.002% in accuracy terms, becoming a key technique for hard real-time application. The last expected data are the invariance of C layers to quantization.

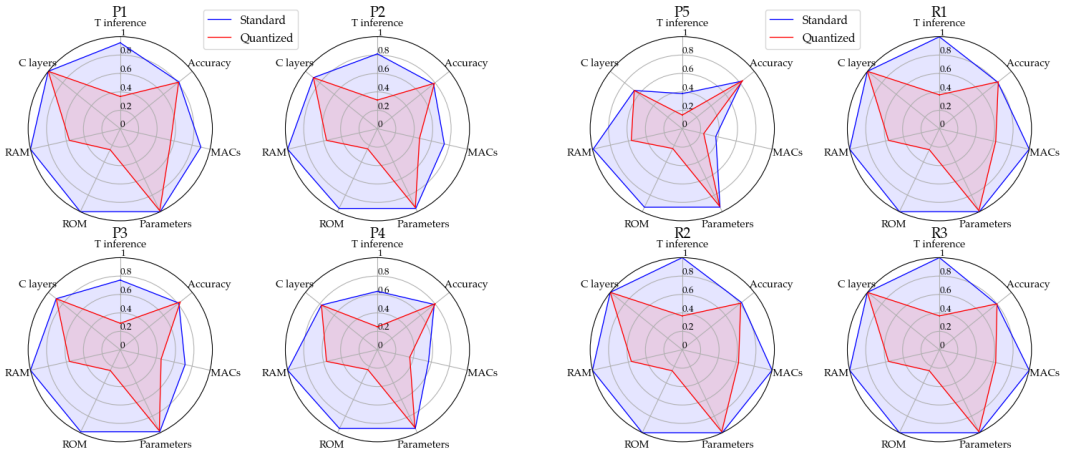


FIGURE 3.4: Optimization tests over best trade-off. P(i) means Pruning test while R(i) means Regularization tests, both executed on Keras (on blue) and quantized TensorFlow lite (on red) versions. (**a**) P1–P4 tests. (**b**) P5, R1–R3 tests.

TABLE 3.4: Relative gain for each investigation parameter due to quantization effect in terms of mean and standard deviation over the whole optimization tests.

| **Investigation Parameters** | $\overline{\%} \pm \sigma_{\overline{\%}}$ |
|---|---|
| Time inference | $0.509 \pm 0.134$ |
| Accuracy | $-0.002 \pm 0.001$ |
| MACs | $0.293 \pm 0.082$ |
| Number of Pars | $0.011 \pm 0.001$ |
| ROM | $0.733 \pm 0.017$ |
| RAM | $0.432 \pm 0.003$ |

**Explanation** Figure 3.5a,b show accuracies and intersections over unions for each test, for pruning and regularization, respectively. These graphs indicate two events: firstly, the trends of accuracy and IoU are largely similar to each other. Exceptions are steps P3 to P5, where trends are opposites. On all tests small variations are outlined, being able to observe substantial stability of both performance and interpretability factors. Table 3.5 reports these aspects for each execution. The low variances of these parameters between the eight tests mean not only having a certain safety margin in correct classification, but even LIME criteria remain the same for most of the test set images explainers. Finally, another aspect is represented by the low average value of IoU over the eight tests, equal to 0.394. Investigating a typical human detection problem through a classification task may involve not so high IoU values: LIME superpixels, differently by detection boxes, cannot isolate people one by one but act, including the densest areas of the image, without geometrical restrictive rules. This implies that the size that the superpixels union area of the two classifiers can reach over is large. Consequently, there are low values of IoU.

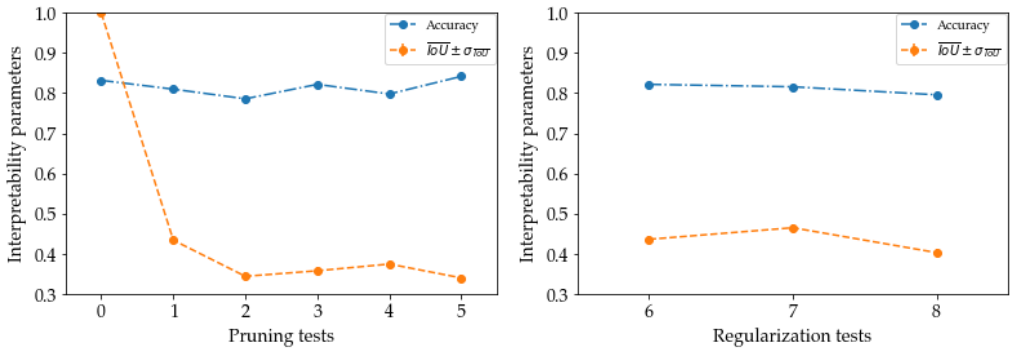


FIGURE 3.5: Interpretability results from over-optimization tests: accuracy and intersection over union trends. (**a**) P1–P5 tests. (**b**) R1–R3 tests.

TABLE 3.5: Explainability trends: accuracy and IoU (in terms of mean and standard error) on each pruning and regularization test on 500 images test set.

| **Test** | **Accuracy** | $\overline{IoU} \pm \sigma_{\overline{IoU}}$ |
|---|---|---|
| P1 | 0.810 | $0.434 \pm 0.009$ |
| P2 | 0.786 | $0.344 \pm 0.009$ |
| P3 | 0.822 | $0.357 \pm 0.009$ |
| P4 | 0.798 | $0.374 \pm 0.001$ |
| P5 | 0.842 | $0.340 \pm 0.009$ |
| R1 | 0.822 | $0.436 \pm 0.010$ |
| R2 | 0.816 | $0.466 \pm 0.009$ |
| R3 | 0.796 | $0.403 \pm 0.009$ |

## 3.4 Prediction of glucose concentration in children with type 1 diabetes using neural networks: an edge computing application

Despite the large amount of works presented for the forecasting of future glycemic levels and the noteworthy results they achieve, all the papers mentioned in 2.1.4 focus on the prediction of glycemic levels of adult subjects. Indeed, there are few works in the literature that aim to predict blood glucose levels specifically in pediatric patients. Children represent the most challenging diabetic population, because pediatric patients go through a period of rapid growth, physiological and hormonal changes along with complex individualization and socialization processes. This often results in a significant decline in the quality of disease management, treatment adherence, and glycemic control [100, 101]. Among the most remarkable studies, Mougiakakou et al. [102] tested two different neural network models on real data of four T1D pediatric patients, after pre-processing features with a glucose-insulin metabolism model. They achieved the best results (average RMSE $= 22.2 \pm 13.4\, mg/dl$) using a feedforward neural network. Dassau et al. [103] proposed a hypoglycemia prediction algorithm that combined five different predictors to assess the risk of incoming hypoglycemia in the following 35 minutes in children with T1D, validating the system on 22 subjects. The decisions of the five models were combined through a majority vote, and the ensemble model identified 91% of the hypoglycemic events with sufficient advance. Finally, De Bois et al. [104] tested six different data-driven models on data of ten virtual T1D children generated using the UVA/Padova simulator [44]. They generated for each patient 29 single days with a three-meal daily scenario, exploiting the simulator's built-in bolus calculator and treating each day as a standalone set of data. For a PH of 30 minutes, they achieved the best numerical performance using a Gaussian Process with a dot-product kernel (average RMSE $= 5.2 \pm 2.0\, mg/dl$). Conversely, the LSTM model resulted in the one with the greatest clinical accuracy, as 97.46% of its predictions fell into zones A and B of the Clarke Error Grid [105], corresponding to accurate predictions.

In the light of what is present in the literature, the contribution of this work, published in [106], is twofold. On the one hand, we implement two state-of-the-art models for the prediction of glycemic levels, and apply them to the specific task of the prediction in pediatric patients; such models improve the performance of the models currently studied in this field. On the other hand, we implement these models on an edge computing system, thus laying the foundations for the future creation of embedded devices capable of forecasting blood glucose levels in order to improve patients' quality of life and aid medical diagnosis; we evaluate the feasibility of such prediction-at-the-edge system on two different boards in terms of prediction accuracy and execution time. To our knowledge, this is the first attempt to implement a pediatric-specific glucose prediction model on an edge-computing system.

### 3.4.1 Edge system

For this work we have used the generated data set of simulated patients presented in 3.2 used to train predictive models.

**Optimization of network models** A Precision Medicine approach was used to tune the predictive models, which involves choosing the hyper-parameters optimally and individually for each different subject. In this work, we implemented

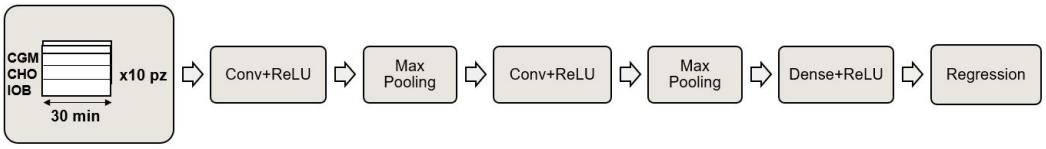


FIGURE 3.6: Schematic representation of the proposed Convolutional Neural Network.

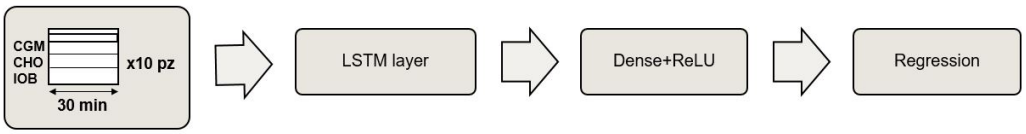


FIGURE 3.7: Schematic representation of the proposed LSTM Recurrent Neural Network.

and optimized a CNN and an LSTM recurrent neural network, because such models achieve the most promising performance in the literature [107]. Both networks were trained using a subset of the available data and then tested on subsequent data of the same *in silico* patient without being updated again. The networks have a sequence-to-label architecture, as the expected output is a single value corresponding to the expected blood glucose value in 30 minutes. After splitting the data into Training (70%), Validation (20%), and Test set (10%), the models were built.

The proposed CNN is a 1D-CNN, with one-dimensional kernel, consisting of two convolutional layers with ReLU activation function, each followed by a Max-Pooling that cuts the parameters in half by taking, in pairs, only the largest value. To complete the model, the convolutional layers are followed by a dense layer with ReLU activation function, and an output neuron that provides the final regression. A schematic representation of the proposed CNN model is reported in Figure 3.6. The choice of hyper-parameters was made by performing a grid search on the validation set, based on a range of parameters including values identified through preliminary tests and parameters reported in the literature [107]. The optimization was done with respect to the kernel size and the number of feature maps.

The proposed LSTM model consists of a first LSTM layer, a dense layer with ReLU activation function, and an output layer that returns the predicted CGM value. Also in this case, the model was optimized in terms of the number of neurons in the first LSTM layer and in the dense layer by investigating both parameters identified in preliminary tests and parameters reported in the literature [107]. A schematic representation of the proposed LSTM model is reported in Figure 3.7.

Both models take as input a $(3 \times 30)$ matrix of values, corresponding to the last 30 minutes of the 3 feature values. Such parameter was identified in preliminary tests, as it provides the models with enough information to capture the recent trend of the features. We found empirically that using longer monitoring periods did not improve performance. With regards to the strategy chosen to train both networks, the Stochastic Gradient Descent (SGD) optimizer is adopted, which requires a learning rate (0.0001), a momentum (0.9) and a clip Value (0.5), which is a necessary parameter to prevent the gradient explosion phenomenon in deep neural networks, improving the prediction quality. The training of both model was performed by splitting the data into mini-batches of 1400 samples (i.e., approximately one day of data) and setting the maximum number of epochs to 200. Finally, to prevent overfitting, the early stopping strategy was adopted, which stops training if the performance on the

validation set does not improve within a fixed number of consecutive epochs.

Two different evaluation metrics are used to thoroughly evaluate the performance of the models. Root Mean Square Error (RMSE) is utilized to assess numerical accuracy, as it provides a numerical estimate of how close the predicted values are to the real ones. Let us consider a prediction performed at timestamp $t$. Defined $P(t + PH)$ as the prediction performed at time $t$ regarding the future glucose value $CGM(t + PH)$, and considering a time series with a total of $T$ timestamps to be predicted, the RMSE is defined as:

$$RMSE = \sqrt{\sum_{t=1}^{T-PH} \frac{(CGM(t + PH) - P(t + PH))^2}{T - PH}} \tag{3.10}$$

where $PH$ is the considered prediction horizon. The smaller the RMSE value, the better the performance. In addition, we considered the Clarke Error Grid (CEG) analysis as a measure of the clinical accuracy of the predictions produced. The CEG consists in a grid which is divided into 5 zones, from A to E, which plots the actual and the predicted CGM values on the horizontal and the vertical plot axis, respectively. Values in zones A and B represent good or acceptable glucose predictions; values in zone C represent mistaken predictions that may lead to unnecessary treatment; values in zone D represent a dangerous failure to predict; finally, values in zone E represent a completely wrong prediction that would lead to erroneous treatment [105].

**Edge system description** In order to test the feasibility of the predictive models of being implemented and utilized on an edge system, we needed to identify the target hardware. Our choice fell on two different devices: a Raspberry Pi4, chosen for its low cost and high computational capability, and a Coral DevBoard, a developer kit containing a Tensor Processing Unit (TPU) processor which is useful for accelerating the execution of machine learning models. The Raspberry Pi4 has a Broadcom BCM2711 quad-core Arm Cortex A72 of 1.5GHz processor, with 4 GB of memory. Furthermore, in order to be able to carry out the tests, we chose to use Raspbian OS (a Debian-derived ISO) as operating system. Python and Mendel Development Tool (MDT) were also installed. The former is necessary to perform tests directly on the Raspberry; the latter is used to give commands to the Coral DevBoard, and therefore allows its set-up and use. The Coral Devboard has a quad Cortex-A53, Cortex-M4F CPU, with 1 GB LPDDR4 RAM, and it has a 4 TOPS (8bit) TPU accelerator for machine learning processes. The operating system running on the DevBoard is Mendel Linux. We installed and utilized all the dependencies necessary to run the model on the board using the Py CoralAPI.

**Edge system implementation** Both datasets were provided as input, as sequences of the last 30 minutes of values, for two models compared: CNN and LSTM. The models were implemented and trained on Google Colab through the use of the open source libraries of Keras and TensorFlow. Through this API, the networks were trained and the hyperparameters optimized.

Although the single models were trained on two different datasets, topologically the trained networks do not differ, in terms of hyperparameters. Therefore, the number of algebraic operations performed by a single network is invariant with respect to the dataset. Having made this consideration we decided to implement on the edge

device only the models trained on the dataset including more hypo/hyperglycemic events, as it is more similar to a real use case.

For the implementation of the models on edge computing architectures, it is necessary to perform a quantization step that differs depending on the architecture on which inference is going to be performed. In order to perform regression tasks on the Rasperry, we chose to use the quantization in *.tflite* format, that transforms the model keeping output variables in *float32* format. This optimization, namely dynamic range quantization, provides latency close to fully fixed-point inference. However, the outputs are still stored using floating point so that the speedup with dynamic-range operations is less than a full fixed-point computation, as reported on the official TensorFlow web page [108]. From now on we will refer to the model obtained with this quantization as *.tflite*.

For the implementation on the Dev Board, it was necessary to transform the models in their 8-bit representation, in order to execute them exploiting the full potential provided by the Coral's TPU. In this case, the quantization method to be used is known as full integer quantization. Applying this approach requires to provide a representative dataset, in order to calibrate variable tensors such as model input, activation functions, outputs of intermediate layers, and model output. As a representative dataset it would theoretically be sufficient to provide a set of 100-500 sample data, taken between the training and validation set. In our case, a dependence of the goodness of the quantization on the subset of data passed to the model as a representative dataset was noted. In fact, it was not sufficient to use data taken randomly from the training or validation set but it was necessary to use ordered data, given the time series forecasting nature of the task. At the end of this quantization procedure, all input and output values are taken to *uint8*. From now on we will refer to the model obtained with this quantization as *uint8*.

Due to the 8-bit nature of the quantization required to exploit the capabilities of the Coral Devboard TPU processor, a problem arose for our regression task. The range of values of the dataset varies between 10 and 600 $mg/dl$, whereas the values that can be represented with 8 bits are 256. Consequently, we pursued two approaches. The first consists in avoiding any pre-processing of the input data, and then reconstructs the possible overflow cases obtained in the output through a post-processing of the data, maintaining the granularity of the prediction at 1 $mg/dl$. The reconstruction was done following the procedure set out in the algorithm 1. It assumes that a decrease of glucose concentration of more than 50 $mg/dl$ in a single minute is very unlikely or impossible. In this case, we post-process the prediction and sum 255 to the predicted value.

The second approach consists in the application of a normalization step in the pre-processing phase, remapping the data values between 0 and 255. Such an approach avoids problems related to overflow, but it takes the granularity of the prediction to approximately 2.33 $mg/dl$. Then, we de-normalized the predicted values to compute the evaluation metrics. This could introduce inaccuracy in the predictions.

The Raspberry and DevBoard were used for the calculation of inference times, to be compared with the performance limits that our application requires (less than the sampling period of the sensor, i.e. 1 minute). At each timestamp, the edge system takes as input the 30 most recent values of the features (i.e., the data of the *in silico* patient produced by the simulator), computes the latest value of the IOB, and performs a prediction of the future blood glucose level. A representative schematic of the experimental system can be seen in Figure 3.8.

**Algorithm 1** Output reconstruction algorithm

```
reconstructed_pred = []                                   ▷ initialization of variables
overflow = False
deltaY = 50
For i,x in enumerate (tflite_uint8_model_prediction):     ▷ Start of the for loop
if x >= 240 then
    if overflow and (x - tflite_uint8_model_prediction[i-1]) >= deltaY: then
        overflow = False
    else if  not overflow and (x - tflite_uint8_model_prediction[i+1]) >= deltaY:
then
        overflow = True
    end if
end if
delta = 255 if overflow else 0
reconstructed_pred.append(x + delta)                      ▷ End of the for loop
```

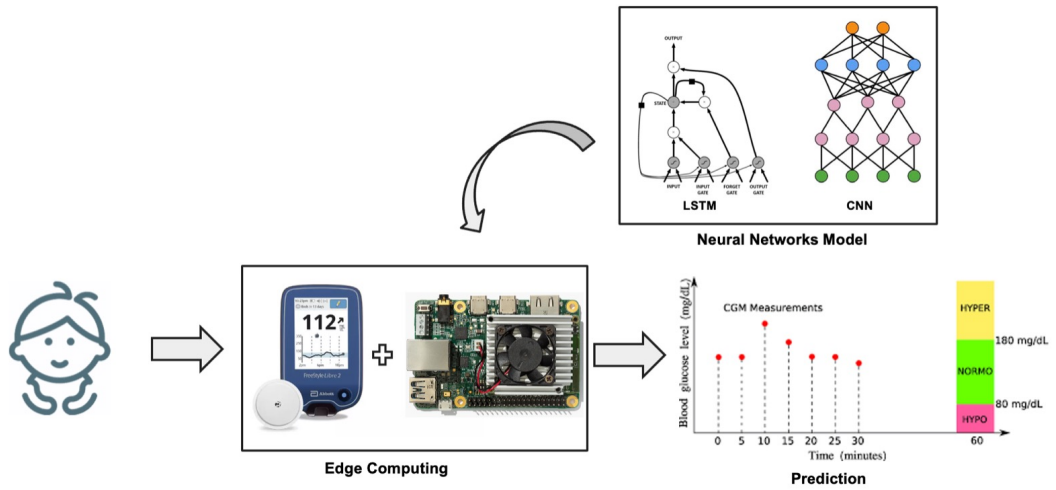


FIGURE 3.8: Schematic representation of the experimental setup during the test phase with edge systems.

### 3.4.2 Results and Discussion

As a result of the grid search performed on the Discovery set, the optimal configuration of the CNN comprises a number of filters equal to 26 for the first convolutional layer, 20 filters for the second convolutional layer, and a Kernel size equal to 1x5 on both. Note that, due to the shape chosen for the filters and to the structure of the input matrix, in the first CNN layer the convolutions are performed on different timestamps of the same feature. With regards to the LSTM model, the optimal configuration resulted in 64 neurons for both the LSTM and the fully-connected layer. Once the models were optimized, predictions were performed on the Test set, and the RMSE and the CEG were computed. With regards to the CEG values, only those from the second dataset were evaluated, as they present more hypo- and hyperglycemic values and are thus more similar to a real-life scenario.

Table 3.6 reports the average values and their standard deviation of the tests performed using the different versions of the models. As expected, the results achieved by the baseline model on the standard dataset are better than those achieved on the dataset with outliers. The LSTM model outperforms the CNN on both datasets, both in terms of average RMSE and CEG results. In particular, with regards to the realistic dataset, the LSTM achieves an RMSE of $16.3 \pm 4.7\, mg/dl$, which is noteworthy if compared to other studies presented in the literature concerning the prediction on pediatric T1D patients. Also, 99.0% of its predictions fall in zones A and B of the CEG and thus represent clinically accurate or acceptable predictions, whereas 1.0% of predictions fall in zone D. The latter mainly correspond to failures of predicting hypoglycemia. No predictions fall in zones C and E.

A comparison with the results achieved in the literature can be only partial, because, as explained in the Introduction, there are a limited number of studies addressing the prediction task on pediatric patients, and only one of them exploits the UVA/Padova simulator. The model tested on data from 4 real pediatric patients by Mougiakakou et al. [102] that achieves an average 22.1 $mg/dl$ RMSE is outperformed by both the proposed models; however, it is known that forecasting glycemia of real patients is though compared to virtual patient, because some unpredictable events might be present. De Bois et al. [104] tested the same 10 virtual children of the UVA/-Padova simulator we utilized; they achieved an average RMSE of 5.2 $mg/dl$ which outperforms both the proposed models in all configurations in terms of numerical accuracy; nonetheless, the clinical accuracy of their best model (zones A+B) is 97.5% and it is outperformed by our models, which both achieve accuracy above 99.0% in their best configuration. However, it must be considered that the two datasets have been generated with different meal and bolus schedules, so this comparison is just qualitative.

**Edge system results and discussions** The results reported in Table 3.6 refer to the models trained without having carried out the normalization of the input values. The expected increase in the RMSE values of the models implemented on the edge devices can be observed; however, this variation differs between the two quantized representations of the networks. With regards to models quantized using dynamic range quantization for implementation on the Raspberry, the RMSE values increase by a maximum of 0.4 $mg/dl$ for the CNN, whereas there is no difference for the LSTM. Again, the LSTM model outperforms the CNN in terms of numerical accuracy, achieving an RMSE of $16 \pm 4.7\, mg/dl$, and 98.9% of its predictions fall in zones A and B of the CEG. This result is of particular interest because it is similar to the performance achieved on datasets composed of data of adult T1D patients, and it

TABLE 3.6: Results of the tests performed with the proposed models CNN and LSTM. In this test, the normalization step was not performed in the pre-processing phase. The results refer to the RMSE [$mg/dl$] achieved on both the ideal (no-error) and the realistic (hypo-hyper) dataset. Such results are reported in terms of average RMSE $\pm$ standard deviation. The CEG results are referred only to the realistic dataset, and its results are reported as percentage on the total dataset. For each neural network, we reported the results for the model implemented on Google Colab, for the model implemented on Raspberry (*.tflite float32* format), and for the model implemented on the Dev Board (.tflite *uint8*).

| **Model** | RMSE (no-error) | RMSE (hypo-hyper) | CEG (A;B;C;D;E) |
|---|---|---|---|
| CNN | $22.2 \pm 2.5$ | $23.2 \pm 2.3$ | 87.0; 12.0; 0.0; 1.0; 0.0 |
| LSTM | $13.5 \pm 3.4$ | $16.3 \pm 4.7$ | 93.8; 5.2; 0.0; 1.0; 0.0 |
| CNN *.tflite* | / | $23.6 \pm 2.0$ | 85.7; 13.6; 0.0; 0.7; 0.0 |
| LSTM *.tflite* | / | $16.3 \pm 4.7$ | 93.7; 5.2; 0.0; 1.1; 0.0 |
| CNN *uint8* | / | $40.1 \pm 11.1$ | 75.4; 20.8; 0.0; 1.2; 2.5 |
| LSTM *uint8* | / | $35.0 \pm 13.3$ | 82.4; 12.5; 0.0; 1.5; 3.6 |

is achieved on the edge device, without resorting to cloud computing. A graphical example of the predictions is reported in Figure 3.9, where we report as an example data of two patients for whom the best and the worst performance is achieved in terms of RMSE. The LSTM prediction is closer to the true CGM value compared to the CNN, which produces more oscillatory predictions; however, the LSTM tends to overestimate both hyperglycemic and hypoglycemic peaks.

Nonetheless, it is worth noting that only 0.7% of predictions of the CNN model fall outside the A and B zones of the CEG, compared to 1.1% of the LSTM; conversely, the LSTM produces more predictions that fall in zone A (93.7% against 85.7% of the CNN). This may be explained considering that the LSTM is more capable of performing accurate predictions in the euglycemic range, which translates into better RMSE and a larger percentage of predictions in zone A, whereas it may miss some hypoglycemic events; on the contrary, the CNN has a larger RMSE and a larger amount of predictions in zone B of the CEG, corresponding to errors in the euglycemic range, whereas it is more capable to predict hypoglycemia. Examples of the CEG are shown in Figure 3.10, where we report as an example data of two patients for whom the best and the worst performance is achieved in terms of CEG percentage in zone A. In conclusion, the CNN may be more appropriate to predict critical hypoglycemic events when implemented in *.tflite*, although its average numeric accuracy is worse than that of LSTM. However, it should be taken into account that results achieved on virtual patients are, in general, slightly better than those obtained on real patients, thus performance may deteriorate when testing on a real dataset.

A different analysis applies to the models on which the full integer quantization was performed for implementation on the Coral DevBoard. Indeed, this quantization technique, that casts the values from *float32* to *uint8*, has more significant effects on the goodness of prediction. In particular, the overflow that is observed when glycemic values are above 255 $mg/dl$ considerably increases the RMSE scores, and generates some predictions that fall in the dangerous E zone of the CEG. For this reason, as explained in section 3.4.1, two different approaches were chosen. The second one, which involved an initial pre-processing of the data, gave considerably better results than the first one, and they are reported in Table 3.7. In particular, the re-

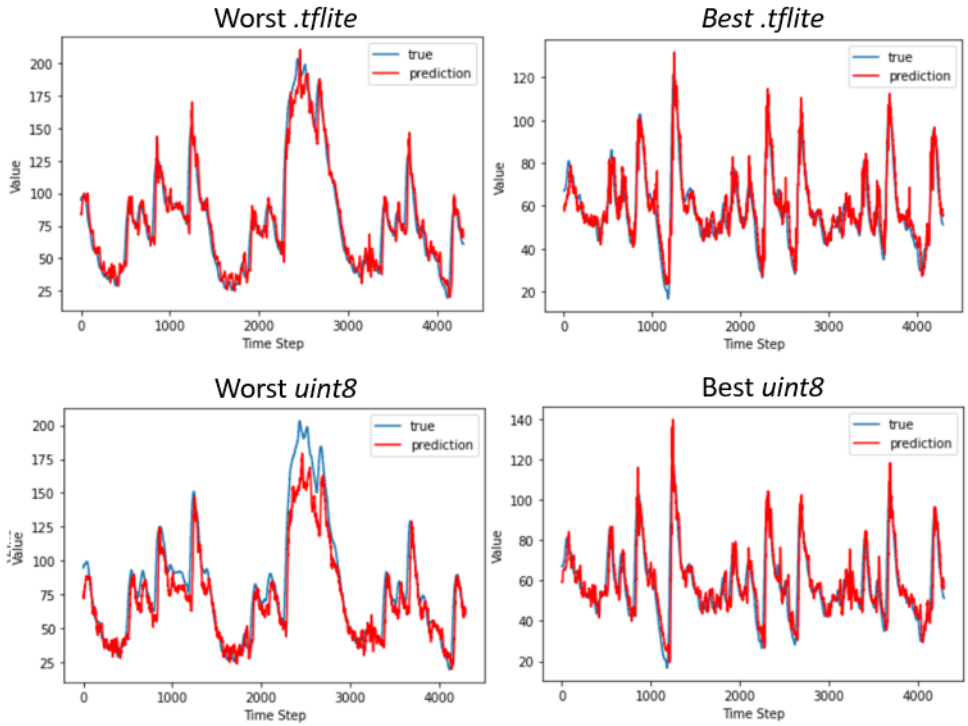


FIGURE 3.9: Graphical examples of the best and worst predictions performed by the CNN (left) and LSTM (right) using different edge devices. We computed the confidence interval for the predicted values, that are 2.01 for the worst *.tflite*, 2.14 for the worst *uint8*, and 1.09 for either the best *.tflite* and *uint8*, respectively. Nonetheless, we do not report such an interval in the figure because its values are too small to be observed in the graphics. The glycemic index values shown in the figure are normalized between 0 and 255, thus, to obtain the real glycemic values, we need to multiply by 2.33.

TABLE 3.7: Results of the tests performed with the proposed models CNN and LSTM, on which was carried the normalization step in the pre-processing phase. The results refer to the RMSE [mg/dl] achieved on the realistic (hypo-hyper) dataset. Such results are reported in terms of average RMSE $\pm$ standard deviation. The CEG results are referred only to the realistic dataset, and its results are reported as percentage on the total dataset. For each neural network, we reported the results for the model implemented on Google Colab, and for the model implemented on the Dev Board (.tflite uint8 format).

| **Model** | RMSE (hypo-hyper) | CEG (A;B;C;D;E) |
|---|---|---|
| CNN | $21.8 \pm 2.3$ | 87.8; 10.9; 0.0; 1.1; 0.0 |
| LSTM | $16.0 \pm 3.4$ | 93.7; 5.5; 0.0; 0.8; 0.0 |
| CNN *uint8*-normalized | $24.7 \pm 5.5$ | 87.6; 9.8; 0.0; 0.9; 0.0 |
| LSTM *uint8*-normalized | $21.2 \pm 8.6$ | 87.4; 7.5; 0.0; 5.1; 0.0 |

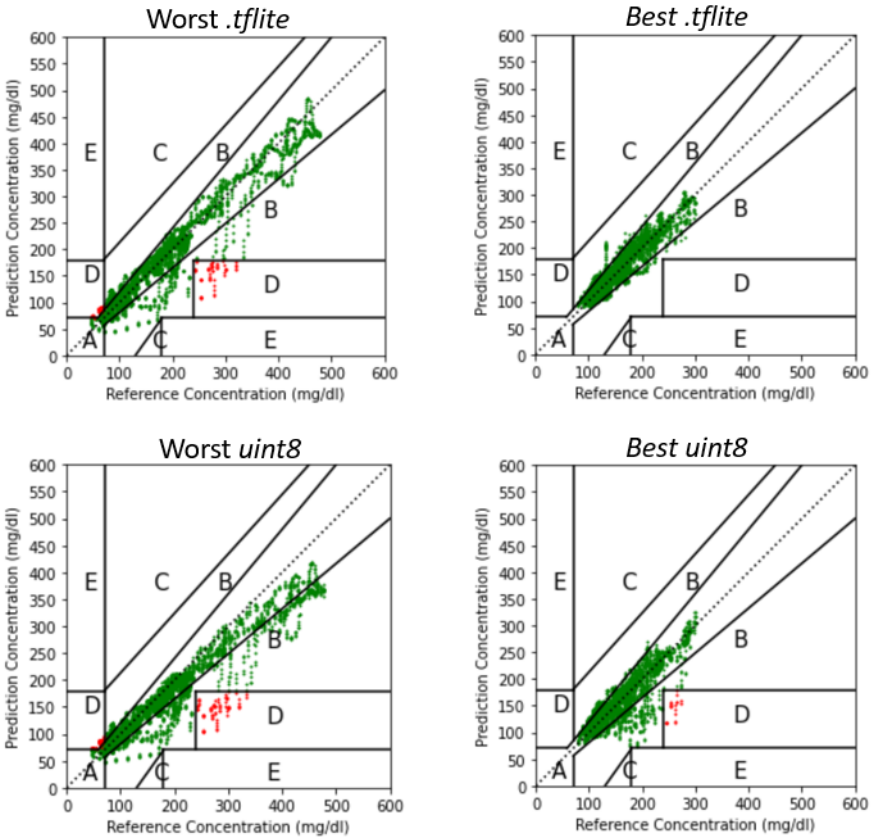


FIGURE 3.10: Clarke Error Grids resulted by the best and worst predictions of the CNN (left) and LSTM (right) using different edge devices. Predictions falling in the safe zones A and B are plotted in green; predictions in zone C are plotted in yellow; predictions falling in the dangerous zones D and E are plotted in red.

TABLE 3.8: Maximum inference time obtained in the test phase in milliseconds. The inference times are reported for each model, CNN and LSTM. They were calculated: for the models saved in TensorFlow saved model format over the Colab online TPU, for the *.tflite* model format over the Raspberry and for the *.tflite* format quantizated in *uint8* over the Coral DevBoard.

| **Model** | Colab TPU (TF Saved Model) | Raspberry (*.tflite*) | Coral DevBoard (*.tflite uint8*) |
|---|---|---|---|
| CNN | 0.085 | 101.56 | 18 |
| LSTM | 0.086 | 70.3 | 12 |

sults obtained for the models in Google Colab do not differ substantially from those achieved without the normalization; conversely, the *uint8* implementation of such models achieves considerably better performance than those obtained with the first approach. It must be considered that the granularity of the prediction increases from 1 $mg/dl$ to 2.3 $mg/dl$. In spite of this drawback, we can still consider this approach better than the first one, because the increase in granularity obtained is not critical from a clinical point of view. It is worth noting that, although the LSTM model outperforms the CNN in terms of RMSE ($21.2 \pm 8.6$ and $24.7 \pm 5.5\, mg/dl$, respectively), 5% of the predictions produced by the LSTM fall in the D zone of the CEG, corresponding to a failure of predicting dangerous events. This situation shows the LSTM model to be weaker to the *uint8* representation, which brings it a greater drop in accuracy. This is probably due to the narrowness of the model, which has only one LSTM plane. Given the limited number of mathematical operations required to achieve an output, the conversion step of the model to *uint8* fails to optimize the weights with the new integer values. On the contrary, only 0.9% of the predictions produced by the CNN fall in the D zone, proving that this latter model is more clinically accurate and reliable when implementing the models in *uint8*, despite the better numerical accuracy achieved by the LSTM model.

A further comparison between the different implementation concerns the actual inference times obtained, which returned largely satisfying results. We reported in Table 3.8 the worst-case results for each model and hardware to show compliance with the time constraints posed by the application. The inference times for both models in all three representations are far below the limit imposed by the application, i.e. 1 minute. However, the total times in the case of a real application should also take into account the times necessary for: signal collection by the sensors, preprocessing of the raw data, and displaying the results on an appropriate Graphic User Interface (GUI). Nonetheless, the time for a single inference operation to be summed are, in the worst case, the ones of the CNN performed in *.tflite* format by the Raspberry, corresponding to 101.56 *ms*. We can therefore assert that inference times, covering at most 0.17% of the total time limit imposed by the application, are not one of the parameters to be optimized in the case of a real implementation of the system. Furthermore, looking at Table 3.8 and comparing the data obtained in the tests of the two Edge systems, a consistent acceleration can be observed with the use of the Coral DevBoard when compared to the Raspberry's performance, although it does not reach the performance of Google Colab TPU. This result is in line with Google's own claims [109].

## 3.5 Image sensors and VPU acceleration for data analysis and classification

With the objective of evaluating the performance increases obtained on a Raspberry Pi device, through the use of VPU (Vision Processing Unit) Movidius™ Myriad 2 and Myriad X accelerators. We have developed a system for the classification of images of subjects with or without face masks [110]. Evaluating the average prediction times in a binary classification task using the face mask dataset presented in 3.2.

### 3.5.1 Model implementation on VPU

To perform the classification task we used the MobilnetV1 network, which is part of the family of networks developed specifically to perform predictions on devices with limited computing capability. Its main feature is to use instead of the normal

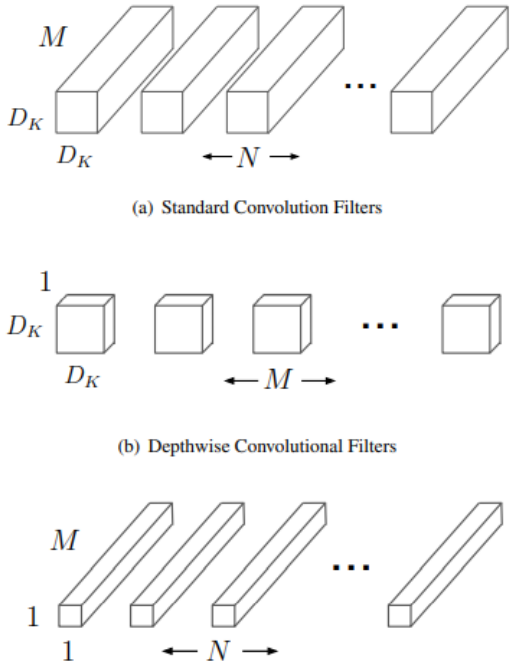


FIGURE 3.11: Standard convolution filter compared to Depthwise Separable Convolution filters [1].

convolution operation the Depthwise Separable Convolution [1].
The latter involves the replacement of the classic convolutional layer with two layers, thus redefining the convolutional filter used during the calculation as shown in the figure 3.11. The computational cost of an operation of convolution is:

$$C_{computational.conv} = (D_K \cdot D_K \cdot N \cdot M \cdot D_F \cdot D_F) \tag{3.11}$$

where M is the number of input channels N the number of output channels, $DK \cdot DK$ the size of the kernel and $DF \cdot DF$ the size of the feature map.

On the other hand, as far as Depthwise Separable Convolution is concerned, the computational costs are divided between the two layers as follows:

$$C_{computational.depth} = (D_K \cdot D_K \cdot M \cdot D_F \cdot D_F) + (N \cdot M \cdot D_F \cdot D_F) \tag{3.12}$$

Thus comparing the two approaches, the reduction in computational cost will be:

$$\frac{C_{computational.depth}}{C_{computational.conv}} = (\frac{1}{N} + \frac{1}{D_K^2}) \tag{3.13}$$

obtained as the ratio of (3.12) to (3.11).

All the training phase of the models has been made on the PC (HP Pavilion equipped with Intel® Core i7 9th Gen 2.60GHz CPU). As language of high level it has been chosen Python. In order to handle the dataset were used the functions of OpenCV to uniform the dimensions of the images in input to the net (resizing to 224x224x3). In order to execute the train of the network, the API (Application programming interface) of TensorFlow and Keras have been chosen because of their wide use and the excellent performances demonstrated in the field of machine learning. The technique used for the training is the Transfer Learning, a methodology that allows to reuse most of the parameters (weights) of a neural network, previously trained on a problem similar to the one we have to solve. Allowing us to concentrate our efforts only on the training of the last layers that are usually those dedicated to the classification. In our case the chosen network is the MobilinetV1 pre-trained on ImageNet, an open source dataset containing more than 14 million images. The model has been trained in three versions:

- MobilnetV1(H.1): steps.per.epoch=34, epoch=5, learning.rate=0.01;
- MobilnetV1(H.2): steps per epoch=34, epoch=20, learning.rate=0.001;
- MobilnetV1(FT): same training hyperparameters of the previous one but once completed the training it has been effected a phase of fine-tuning of 10 epochs that has carried the trainable parameters of the model from 2050 to 6146.

For the edge system, a Raspberry Pi Model 3B+ was used, equipped with an ARM Cortex-A53 CPU (1.4GHz 64-bit quad-core) and 1GB of RAM shared with the graphics card. The operating system installed on the Raspberry Pi was Raspbian OS, derived from Debian. To conduct the tests, the Raspberry Pi was configured with the necessary software for performing image classification tasks using the TensorFlow framework. The files in the SavedModel format, along with a Python script for image classification, were loaded onto the device. To enhance system performance, the same Raspberry Pi was augmented with two Intel Movidius™ Neural Compute Sticks, NCS1 and NCS2. These devices, leveraging Myriad 2 and Myriad X Vision Processing Unit (VPU) technologies, provided efficient support for artificial intelligence workloads. To prevent memory overload on the Raspberry Pi, only the essential files for the testing phase were transferred to the device. Specifically, the portion of the database containing images not used during the models' training phase was included, along with eight additional images unrelated to the generative dataset, to assess the networks' generalization capabilities. For classification tasks utilizing the NCS and NCS2 accelerators, the OpenVINO™ Software Developer Toolkit [111] was employed to generate the .xml and .bin files required for executing the network models. Using OpenVINO™'s Inference Engine, a program was developed to load the trained network models onto the NCS devices and retrieve classification result reports. For comparison purposes, the same computer used during the training phase was also employed for classification. To ensure consistency and comparability of results, the classification program used on the PC was identical to the one deployed on the Raspberry Pi.

### 3.5.2 Results and Discussion

As far as the classification task is concerned, table 3.9 shows the training results in terms of accuracy and loss. Such results are obtained through a 5-fold cross-validation using the face mask dataset presented in 3.2.

TABLE 3.9: The average values of accuracy (val.acc), and loss(val.loss).

| - | MobilnetV1 (H.1) | MobilnetV1 (H.2) | MobilnetV1 (FT) |
|---|---|---|---|
| **val.acc** | *0.8945* | *0.9709* | *0.9818* |
| **val.loss** | *0.4474* | *0.3615* | *0.3318* |

[a]calculated using 5-fold cross validation.

Tests were also performed to check that during the transformation of the models, from .pb files to their intermediate representation, there was no loss of information. In order to perform these tests, the classification results of the individual models in the various formats ( .pb and intermediate representation of OpenVINO™2020.3 and OpenVINO™2021) were compared. From the results obtained in the aforementioned tests it is possible to state that no loss of information occurred during the transformation of the models.

Classification tests on Edge system were performed by submitting a batch of 30 images, extracted with random logic from the dataset before the training phase. The test was repeated 25 times for each model and configuration. The average classification times and their coefficient of variance were then calculated for comparison. However, it must be taken into account that the Raspberry device having installed the Raspian operating system does not allow to control all processes that the machine is running. Therefore, in order to standardize the data obtained without introducing errors due to slowdown of the device caused by back processes, we followed a very precise workflow. The tests were carried out following these steps: (1) once the device is turned on, we disable the wi-fi internet connection and wait 1 minute before opening the command prompt, (2) once the command prompt is turned on we work for 15 minutes,(3) we wait 5 minutes before starting the operations again.

Table 3.10 shows the mean classification time values $\tau$ in milliseconds and the coefficient of variation $\sigma$ (standard deviation divided by the arithmetic mean). Comparing the NCS2 with the i9 CPU shows that in all three trained models the classification times between the two systems are comparable. Looking at the table about the MobilNetV1(H.2) and observing the elements in green, it is possible to see average classification times that are only one millisecond apart. Instead, as far as the coefficient of variation is concerned, the results obtained show a value that is very low for the networks executed on Neural Compute Stick devices, almost indifferent between the two versions.
While it results a higher value both for networks executed on CPU from pc and directly on Raspberry without the accelerators. During the test operations of the first model on a non-accelerated device there was a temperature rise that led the CPU to thermal throttling, a situation that has certainly affected the classification timing of the model. In fact, it can be seen that the first model shows a particularly high coefficient of variation, element in red, during the testing phase on Raspberry Pi.

TABLE 3.10: shows the mean classification times, $\tau$, expressed in milliseconds and the coefficient of variation, $\sigma$.

| **MobilnetV1 (H.1)** | | | | |
|---|---|---|---|---|
| **Hardware** | **CPU (TensorFlow)** | **Raspberry (TF)** | **NCS** | **NCS2** |
| $\tau$ **(ms)** | 30.08 | 440.28 | 62.412 | 40.102 |
| $\sigma$ | 0.080 | 0.261 | 0.014 | 0.003 |
| **MobilnetV1 (H.2)** | | | | |
| **Hardware** | **CPU (TensorFlow)** | **Raspberry (TF)** | **NCS** | **NCS2** |
| $\tau$ **(ms)** | 38.93 | 396.50 | 62.774 | 38.975 |
| $\sigma$ | 0.079 | 0.060 | 0.004 | 0.003 |
| **MobilnetV1 (FT)** | | | | |
| **Hardware** | **CPU (TensorFlow)** | **Raspberry (TF)** | **NCS** | **NCS2** |
| $\tau$ **(ms)** | 33.77 | 401.45 | 62.782 | 40.078 |
| $\sigma$ | 0.090 | 0.095 | 0.003 | 0.004 |

## 3.6 Efficient Detection of Microplastics on Edge Devices with Tailored Compiler for TinyML Applications

Microplastics, defined as synthetic particles or polymeric matrices ranging from 1 $\mu m$ to 5 mm, are an escalating environmental concern due to their resistance to biodegradation and prevalence across marine, freshwater, and terrestrial ecosystems [112–114]. Addressing their detection aligns with the United Nations' Sustainable Development Goals to conserve marine ecosystems [115]. Advanced detection methods, such as Fourier Transform Infrared (FTIR) and Raman spectroscopy [116], combined with AI-driven approaches [117, 118], provide robust frameworks for water quality monitoring. However, implementing these technologies on IoT devices in resource-constrained environments necessitates lightweight and efficient AI models.

TinyML, a research field dedicated to adapting machine learning models for low-power microcontrollers, plays a pivotal role here. Techniques like quantization and pruning enable the deployment of efficient neural networks (NNs) on devices with limited computational and memory resources [119, 120]. TinyML applications span diverse areas, including wearable health monitors and industrial anomaly detection [121, 122], where on-device processing reduces latency and improves privacy [123].

To advance this field, we propose a novel framework based on the ONNX API for compiling high-level, quantized models directly into native C code. This approach circumvents reliance on inference engines like ONNX Runtime or TensorFlow Lite, enabling:

- Lower latency and energy consumption;
- Reduced memory usage by eliminating inference engine overhead;
- Enhanced portability across diverse microcontrollers.

The framework supports quantization and is benchmarked across multiple microcontrollers, showing superior performance in terms of responsiveness, memory efficiency, and energy consumption.

### 3.6.1 NeuralCasting Compiler

We present NeuralCasting, a Python-based compiler designed to convert ONNX format models into native C code. This approach optimizes the deployment of NNs on edge devices, particularly bare metal boards with limited memory and computational power, where the lack of an operating system complicates the deployment. Frameworks like Google's `TFLite` and Microsoft's `ONNX Runtime` allow the deployment of NNs on embedded systems; nevertheless, they require an inference engine, introducing an overhead in memory usage and inference time on edge devices. Moreover, compatibility issues often arise on bare metal microcontrollers, where an additional abstraction layer is undesirable and challenging to deploy.

NeuralCasting's architecture utilizes the ONNX format as its foundation, i.e., a Direct Acyclic Graph (DAG). In ONNX, nodes take the form of tuples containing the node's name, the name of the incoming edges, and the name of the outgoing edges. NeuralCasting parses these nodes, converting them into an internal data structure holding the metadata of the operators and connecting the network using memory references. This enables graph traversal and sequential C code generation based on the operator type (e.g., Gemm, ReLU, Convolution) and metadata (e.g., the kernel size of the Convolution).

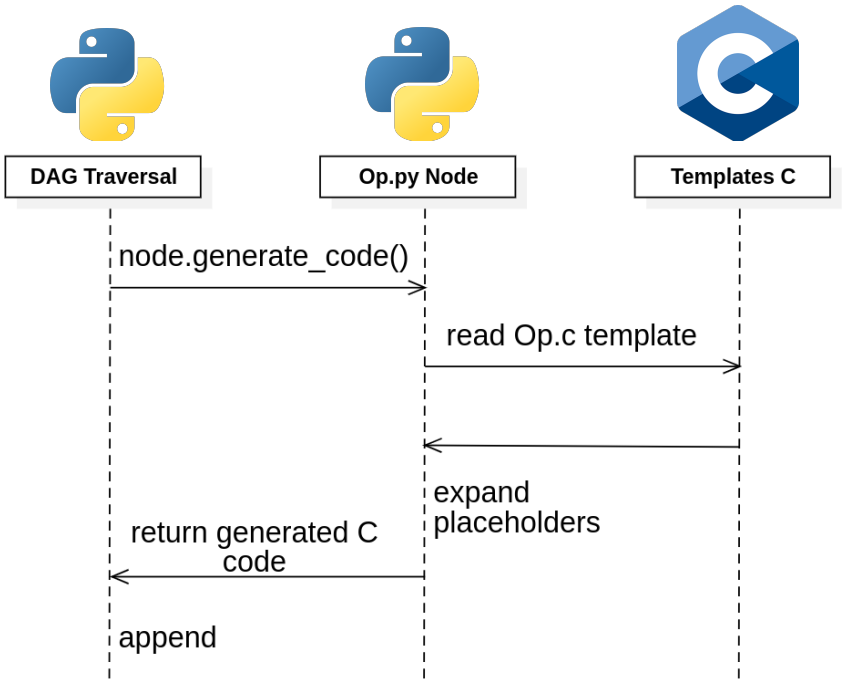


FIGURE 3.12: The workflow of NeuralCasting for the C code generation. The kernel is implemented in Python. During the DAG traversal, native C code is generated from templates. Indeed, the templates are expanded using the information stored in Op.py (acquired from the original ONNX model). Finally, the C code generated for the operator is appended to the current generated code of the entire network.

The C code generation occurs through the information stored inside the internal DAG, used as an intermediate representation by the compiler. Successively, the kernel accesses a hand-crafted C template, univocally related to the operator. The templates contain *placeholders* to expand into generated C code, ranging from simple macros to more complex sections of code, such as indexing to ensure broadcasting between matrices and tensors of different sizes. The information to expand the placeholders is inside the node. Finally, the kernel appends the generated C to the main generated code, created from the previous operators analyzed during graph traversal. Figure 3.12 shows the pipeline used by the compiler for generating C code.

NeuralCasting provides flexibility in memory allocation regarding the weights and biases of the models, allowing allocation either in the process's data segment or the heap, depending on requirements. It relies on a compiler flag *-alloc*, which can be set to *heap* for heap allocation or *data* for data segment allocation as global variables. Concerning the allocation in the heap, C code and binary files containing weights and biases are generated and read within the program through a specific *nn_alloc* function that allocates memory in the heap. Similarly, an *nn_free* function deallocates memory. To prevent memory fragmentation, allocate weights once at the program's start and deallocate them at the end. For data segment allocation, weights are compiled internally in the data segment, eliminating the need for external sources. Considering our microplastic detection task, which demands high performance with relatively small matrices, we chose to allocate in the data segment to maximize performance. Moreover, deploying on bare metal chips without an operating system introduces additional difficulties and risks related to manual dynamic memory allocation.

Also, NeuralCasting provides the generation of parallelized C code using *OpenMP*, through a *-parallel* flag settable to *omp*, for the automatic generation of OpenMP clauses. Furthermore, a *-n_threads* flag allows to define the number of threads used. Considering our small-sized models, using a single thread was optimal, as parallelization did not enhance performance due to the overhead of thread management.

Compared to other state-of-the-art frameworks like `TFLite` and `ONNX Runtime`, NeuralCasting currently offers fewer optimizations. It lacks inter-layer optimizations, node fusion, and architecture-specific optimizations. However, NeuralCasting generates code for bare metal edge devices, making it optimal for small models where advanced parallelization and vectorization are unnecessary. Generating native C code enhances portability across architectures, and the absence of an inference engine eliminates overhead, improving performance in terms of both memory usage and inference time.

**Integration of Quantization in NeuralCasting** Due to the growing need to compress models and optimize performance on edge devices, NeuralCasting presents an extension to support ONNX quantization units, such as QGemm, QAdd, QSigmoid, and other fundamental components of quantized NNs.

A quantized ONNX model can be exported via the ONNX API, using a representative dataset and performing post-training quantization. The ONNX Q units provide necessary quantization data, such as scaling factors and zero points.

Supporting quantization in a compiler significantly enhances the deployment of machine learning models on resource-constrained devices. It reduces model size and computational requirements, resulting in faster inference, lower latency, and improved performance, essential for real-time applications. Additionally, quantized

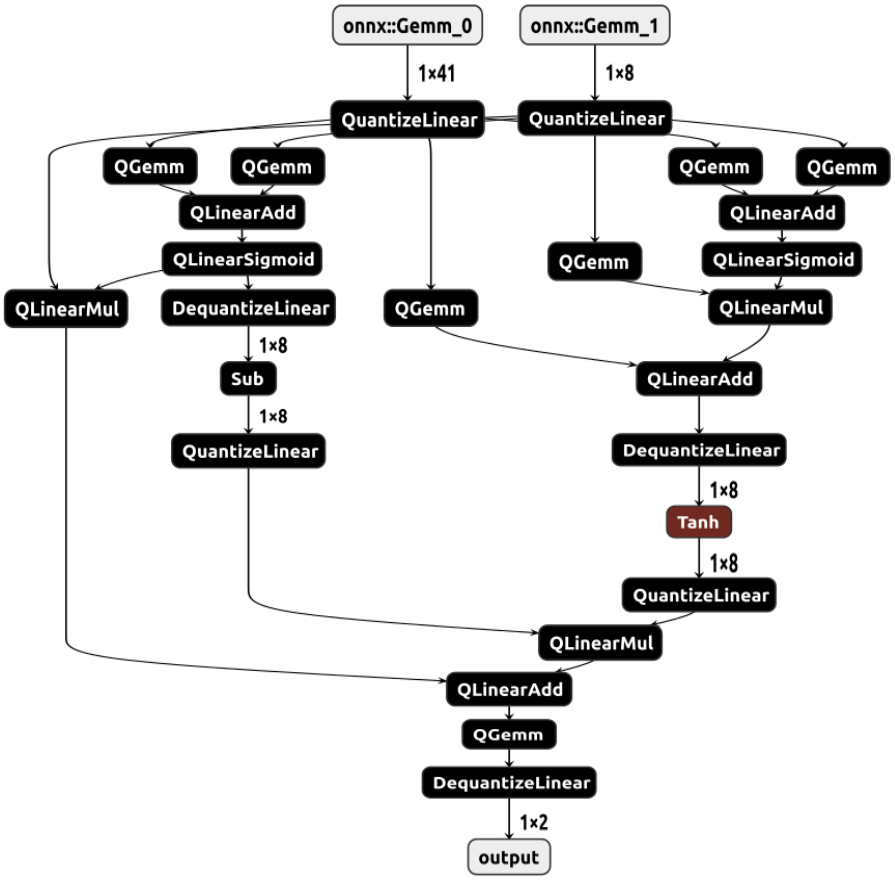


FIGURE 3.13: The model based on quantized GRU, re-implemented on its single elements.


FIGURE 3.14: The ONNX graph of the Quantized MLP model. The QGemm, as a Q-Unit, presents attributes specific to linear quantization, consequently used by the compiler for generating C code.

models consume less energy, extending battery life in portable devices. Integrating quantization into a compiler allows developers to focus on model development while ensuring efficient, high-performance deployment across various hardware platforms.

Quantization was integrated into NeuralCasting similarly to the integration of traditional float operators. Indeed, NeuralCasting is based on the ONNX format, generating native C code using the information inside each unit. ONNX provides units specifically designed for quantized operators, named Q-Units. For example, *QLinearConv*, *QLinearMatMul* and *QGemm*. Moreover, it provides a specific *QuantizeLinear* Q-Unit for the quantization of weights and activations, and a Q-Unit for de-quantization (i.e., to re-convert the integers to floating point), called *DequantizeLinear*. These units present specific attributes for quantization, such as zero points and scaling factors. The quantization hyperparameters are used by NeuralCasting to generate C code for the quantized operators. Figure 3.14 reports a Quantized MLP model, where the quantized input feeds a series of Quantized Generalized Matrix Multiplications (QGemm). Finally, the resulting quantized tensor is de-quantized to provide the float32 output tensor. As presented in Figure 3.14, each quantized operator provides the necessary scaling factors and zero points for the inputs and outputs. For example, considering a QGemm operator, the linear quantization used applies the linear quantization definition to a floating point Gemm defined in 3.16, resulting in the equation:

$$\mathbf{q_y} = \frac{s_W \cdot s_x}{S_y} \cdot (\mathbf{q_W} \cdot \mathbf{q_x} + \mathbf{q_{bias}}) + z_y \tag{3.14}$$

with the pre-computed factor $\mathbf{q_{bias}}$:

$$\mathbf{q_{bias}} = \mathbf{q_b} - z_x \cdot \mathbf{q_W} \tag{3.15}$$

where $s_W$ is the scaling factor of the weights matrix, $s_x$ is the scaling factor of the input tensor, $s_y$ is the scaling factor of the output, $\mathbf{q_W}$ is the quantized weights matrix, $q_x$ is the quantized input tensor, $z_y$ is the zero point of the output tensor, $\mathbf{q_b}$ is the quantized bias vector and $z_x$ is the zero-point of the input tensor.

ONNX does not provide a quantized unit for the GRU. As a result, it was implemented in NeuralCasting by combining elementary operators based on the equation 3.17 (see Figure 3.13). Finally, lookup tables based on 256 points have been implemented for the activation functions to improve inference time.

**Models Description** We employed two distinct NN architectures to analyze and process our data: Multilayer Perceptrons (MLPs) and Gated Recurrent Units (GRUs). These models were selected due to their proven effectiveness in handling different data types. Below, we provide a detailed description of each model, including its structure, functioning, and the specific methods used for training.

- **Multilayer Perceptrons (MLPs)**: MLPs represent a foundational class of artificial NNs extensively utilized for a wide range of pattern recognition and classification tasks. An MLP is structured with an input layer, one or more hidden layers, and an output layer, where each node, apart from the input nodes, functions as a neuron equipped with a non-linear activation function. This layered architecture enables MLPs to approximate complex, non-linear functions, making them particularly effective in scenarios that require the learning of complex patterns and relationships within data. The training of MLPs

is typically conducted using back-propagation, a gradient-based optimization technique that minimizes the error between the predicted and actual outcomes. The output of a neuron in an MLP can be represented as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.16}$$

where $f$ is the activation function, $w_i$ are the weights, $x_i$ are the inputs, and $b$ is the bias. Despite their relatively simple architecture compared to more advanced models, MLPs remain a cornerstone in the field of machine learning due to their robustness and versatility in handling diverse datasets.

- **Gated Recurrent Units (GRUs)**: GRUs are a specialized form of recurrent neural networks (RNNs) designed to address some of the limitations inherent in traditional RNNs, such as the vanishing gradient problem [124]. GRUs incorporate gating mechanisms. Specifically, *update* and *reset* gates, which modulate the flow of information across the network. This allows GRUs to maintain and update information over long sequences more effectively, making them highly suitable for tasks involving sequential data, such as time-series forecasting and natural language processing. The equations describing a GRU are:

$$\begin{aligned}
\mathbf{z_t} &= \sigma(\mathbf{W_z} x_t + \mathbf{U_z} \mathbf{h_{t-1}} + \mathbf{b_z}) \\
r_t &= \sigma(\mathbf{W_r} x_t + \mathbf{U_r} h_{t-1} + \mathbf{b_r}) \\
\tilde{h}_t &= \tanh(\mathbf{W_h} x_t + r_t \odot \mathbf{U_h} \mathbf{h_{t-1}} + \mathbf{b_h}) \\
\mathbf{h_t} &= (1 - \mathbf{z_t}) \odot \mathbf{h_{t-1}} + z_t \odot \tilde{\mathbf{h}}_\mathbf{t}
\end{aligned} \tag{3.17}$$

where $\mathbf{z_t}\mathbf{z_t}$ is the update gate, $\mathbf{r_t}$is the reset gate, $\tilde{\mathbf{h}}_\mathbf{t}$ is the new hidden state, $\mathbf{h_t}$ is the final hidden state, $\mathbf{W}$ and $\mathbf{U}$ are weight matrices, $\mathbf{b}$ is the bias vector, concatenation of $(\mathbf{b_z}; \mathbf{b_r}; \mathbf{b_h})$, and $\odot$ denotes element-wise multiplication. The streamlined architecture of GRUs facilitates faster training times and reduces computational demands while still providing competitive performance. The ability of GRUs to capture temporal dependencies with greater efficiency underlines their importance in modern NN applications, particularly in contexts where data exhibits sequential or temporal characteristics.

**Experimental Setup** A Neural Architecture Search (NAS) process was executed to select the network designated as the final model to be trained and deployed on devices. For the MLP, a greedy grid-search approach was applied, considering the number of layers (1, 2, 3) and the number of neurons per layer (8, 16, 32, 64) as parameters. Similarly, for the GRU, the same approach was applied to determine the most effective hidden size (8, 16, 32, 64). The steps involved included:

- **Parameter Initialization**: Setting the initial values for the number of layers and neurons per layer as per the grid search ranges;
- **Model Generation**: Constructing various network architectures by systematically varying the number of layers and neurons;
- **Training**: Each generated model was trained using the designated dataset;
- **Evaluation**: The performance of each model was assessed based on specific metrics, i.e., accuracy and computational efficiency. In cases where multiple

models achieved similar accuracy, the architecture with the lower computational cost was chosen;

- **Selection**: The model with the optimal performance metrics was selected as the final architecture for deployment.

This approach ensured a comprehensive exploration of potential network configurations, allowing the identification of the most effective model for the given application.

Once we identified the architectures designated for porting to edge systems, we proceeded to quantize them. We chose to perform float32 to int8 quantization using either the native functions provided by `ONNX Runtime` and via NeuralCasting.

The models were quantized via Post-Training Quantization (PTQ), i.e. the quantization was carried out at the end of the model training phase. Usually, an approach via Quantization Aware Training (QAT) is preferable, as including the quantization process within the training allows a higher accuracy recovery. Nonetheless, in this case, the PTQ was preferred for several reasons. First, the microplastic detection task relies on a binary classification, which is little affected by small approximations of the output tensor. Consequently, the difference between the accuracy via PTQ and QAT is negligible. Furthermore, QAT is notoriously expensive from a computational point of view as it involves a simulation of the quantized model in the forward pass of training. Considering an expensive NAS algorithm such as Grid Search, this approach would have proven to be unnecessarily expensive from a computational point of view, with minimal gains in terms of accuracy.

After quantizing the models, we conducted 100 tests using 100 synthetically generated datasets (according to parameters reported in Section 3.2) to compare the performance of both the original and quantized versions of the MLP and GRU models. This comparison was performed by implementing a Student's t-test on the populations of the accuracies obtained from the 100 tests between the original models and their quantized version. Moreover, the comparison in terms of accuracy between the two quantized versions of the model was not carried out since the two models perform the same mathematical operations exploiting the same quantization ranges as explained in subsection 3.6.1, thus resulting in two perfectly equal models at the level of inference results.

**Deployability on MicroController Unit (MCU)** The quantized versions of the MLP and GRU models were tested on edge devices. In particular, the C implementation of the code is flashed into a set of microcontrollers with characteristics that cover a varied performance range. The following MCUs were chosen: STM32H747XI -M7/M4 with a dual-core ARM Cortex-M/ at 480 MHz and Cortex-M4 at 240 MHz processor, nRF52840 with an ARM Cortex-M4 CPU with FPU running at 64 MHz processor, and ESP32-PICO-D4 with a dual-core Xtensa 32-bit LX6 microprocessor at 240 MHz. We tested the microcontrollers using their respective development boards: Portenta H7 for the STM32H747XI, Arduino Nano 33 BLE for the nRF52840, and M5StickC Plus for the ESP32-PICO-D4. Tests were also performed on a single-board computer, Raspberry Pi 4, which has a Broadcom BCM2837B0 processor, Cortex-A53 (ARMv8) 64-bit SoC, with a frequency of 1.4 GHz. The technical specifications of the devices are shown in Table 3.11. Each microcontroller and single-board computer was chosen to represent a broad spectrum of processing power and memory capabilities, providing a comprehensive assessment of the model's scalability and adaptability across different hardware platforms. For each MCU, inference was performed on

both models 10,000 times, and the execution time of each inference was measured. Finally, the mean, maximum, and standard deviation of the execution times were calculated. To better compare performance on different device power consumption, memory occupation was evaluated.

TABLE 3.11: Technical Specification of the devices tested powered at 5V

| Board | MCU | Memory | Frequency [MHz] | Current [A] | Power [W] | Voltage [V] |
|---|---|---|---|---|---|---|
| Portenta H7 | STM32H747XI-M7 | 2MB Flash, 1MB RAM | 480 | 0.12 | 0.6 | 5 |
| | STM32H747XI-M4 | | 240 | 0.12 | 0.6 | 5 |
| Arduino Nano 33 BLE | nRF52840 | 1MB Flash, 256KB RAM | 64 | 0.02 | 0.1 | 5 |
| M5StickC Plus | ESP32-PICO-D4 | 4MB Flash, 520KB SRAM | 240 | 0.5 | 2.5 | 5 |
| Raspeberry Pi 4 | SoC BCM2711 | 8GB LPDDR2 SDRAM | 1800 | 0.48 | 2.4 | 5 |

### 3.6.2 Results and Discussion

The NAS process resulted in the best parameters identified for the MLP being one layer and eight neurons. Instead, the optimal hidden size for the GRU was determined to be eight. After quantization, 100 tests were performed to compare the original and quantized models: the original MLP model and the quantized MLP model achieved the same average accuracy of $0.991 \pm 0.001$. The conducted Student's t-test indicated that the difference between the distributions was not significant ($p$-value = 0.733). For the GRU, the original model showed an average accuracy of $0.993 \pm 0.001$, whereas the quantized model demonstrated a slight drop in accuracy ($0.985 \pm 0.001$). This difference was statistically significant ($p - value < 0.05$).

TABLE 3.12: Performance in terms of latency time relating to the MLP model, using `ONNX Runtime` and NeuralCasting for deployment, on different HW architectures. The acronym N.A. means *Not Available*, i.e. the framework is not supported by the platform.

| Hardware Platform | Inference Time $[\mu s]$ | | Maximum Time $[\mu s]$ | |
|---|---|---|---|---|
| | NeuralCasting | ONNX Runtime | NuralCasting | ONNX Runtime |
| x86_64 Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz | $0.63 \pm 0.59$ | $32.20 \pm 4.97$ | 400.00 | 624.00 |
| Portenta H7 - STM32H747XI -M7 | $46.28 \pm 3.01$ | N.A. | 56.00 | N.A. |
| Portenta H7 - STM32H747XI -M4 | $46.16 \pm 2.91$ | N.A. | 54.00 | N.A. |
| Arduino Nano 33 BLE | $407.62 \pm 19.51$ | N.A. | 474.00 | N.A. |
| M5StickC Plus | $63.55 \pm 1.86$ | N.A. | 71.00 | N.A. |
| Raspeberry Pi 4 | $6.72 \pm 2.29$ | $45.85 \pm 1.81$ | 33.00 | 87.97 |

TABLE 3.13: Performance in terms of latency time relating to the GRU model, using `ONNX Runtime` and NeuralCasting for deployment, on different HW architectures. The acronym N.A. means *Not Available*, i.e. the framework is not supported by the platform.

| Hardware Platform | Inference Time $[\mu s]$ | | Maximum Time $[\mu s]$ | |
|---|---|---|---|---|
| | NeuralCasting | ONNX Runtime | NuralCasting | ONNX Runtime |
| x86_64 Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz | $1.10 \pm 0.90$ | $13.20 \pm 1.76$ | 133.00 | 63.20 |
| Portenta H7 - STM32H747XI -M7 | $211.60 \pm 3.86$ | N.A. | 247.00 | N.A. |
| Portenta H7 - STM32H747XI -M4 | $211.89 \pm 3.95$ | N.A. | 252.00 | N.A. |
| Arduino Nano 33 BLE | $1964.81 \pm 11.16$ | N.A. | 2001.00 | N.A. |
| M5StickC Plus | $255.59 \pm 3.35$ | N.A. | 262.00 | N.A. |
| Raspeberry Pi 4 | $9.74 \pm 3.71$ | $109.42 \pm 3.53$ | 55.00 | 194.78 |

Upon deploying the software on the different devices chosen, the results in terms of power consumption, memory occupation, and inference time were evaluated. Considering the power consumption of the algorithms, as shown in Table 3.11, it

can be seen that there is a variation based on the characteristics of the chosen platform. For the sake of brevity, the consumption figures in a single column for both the GRU and the MLP implementation as the values were found to be identical. The lower power consumption during the task execution is around 0.02 W for the Arduino Nano 33 BLE Sense platform, while the higher consumption is reached by the M5StickC Plus, which is 0.44 W. It was seen, as shown in Table 3.14, that the RAM and Flash used are a bit higher for the GRU model compared with the percentage required for the MLP for all the platforms. Overall, the STM32H747XI-M7 had the lowest resource usage, whereas the nRF52840 exhibited the highest RAM consumption, indicating that both models show consistent resource usage patterns across different microcontrollers, which is crucial for optimizing model deployment in resource-constrained environments.

In addition, it is worth mentioning that directly compiling the network in C is advantageous compared to using an inference engine on embedded systems. Besides the additional abstraction layer, the inference engine requires no negligible memory space on small edge devices. Instead, directly compiling the network into C requires only the storage relating to the weights and biases of the model and the algorithms of the operators, respectively, in the data segment and the text segment of the program.

TABLE 3.14: Resource Utilization of GRU and MLP Models on various Microcontrollers, for each cell, is reported the ratio between the used MB and the total available MB

| MCU | GRU | | MLP | |
|---|---|---|---|---|
| | RAM [MB/MB] | Flash [MB/MB] | RAM [MB/MB] | Flash [MB/MB] |
| STM32H747XI -M7 | 0.15/0.52 | 1.51/7.86 | 0.14/0.52 | 0.14/0.78 |
| STM32H747XI -M4 | 0.12/0.29 | 0.08/1.04 | 0.12/0.29 | 0.07/1.04 |
| nRF52840 | 0.12/0.26 | 0.08/0.98 | 0.12/0.26 | 0.08/0.98 |
| ESP32-PICO-D4 | 0.10/0.33 | 0.27/1.31 | 0.10/0.33 | 0.27/1.31 |

A more in-depth analysis is required for the evaluation of inference.

Table 3.12 shows the results for the quantized MLP model in terms of mean inference time, standard deviation, and maximum time. Where possible, results for the Neural Casting and ONNX Runtime implementation were compared. Using the native C code generated by NeuralCasting, there is an evident increase in inference time with the degradation of the hardware characteristics. The result achieved with Raspberry Pi 4 is about an order of magnitude smaller than the results obtained by the reference computer. Considering the MCU set, the best inference time is $41.16 \pm 2.9$ $\mu s$ reached by Portenta H7 (core M4). When analyzing the maximum times, to avoid high out-of-range values of the inference time, a washout operation was carried out to eliminate possible initial fluctuations in the execution of the first 100 tests. The same behavior was found for the GRU model (showed in Table 3.13). Clearly, the inference time increases as the hardware characteristics degrade, using the C code generated by NeuralCasting. The result obtained with the Raspberry Pi 4 is approximately an order of magnitude lower than the results from the reference computer. Among the MCU set, the best inference time is $211.60 \pm 3.86$ $\mu s$, achieved by the Portenta H7 (core M7). Furthermore, for all MCUs tested, it was not possible to implement the ONNX runtime version of the algorithms due to incompatibilities.

In the case of the x86_64 experiments, the standard deviation is relatively high compared to the mean due to the distribution of the measurements. In fact, out of 100,000 experiments, 99,989 measurements were in the order of $10^{-7}$, 6 in the

order of $10^{-6}$, 4 in the order of $10^{-5}$, and 1 in the order of $10^{-4}$. Such sporadic outliers influence the standard deviation considerably, inflating it by several orders of magnitude. A similar discussion concerns the measurements of the Raspberry Pi 4.

In Figure 3.15, a bar plot shows the result previously discussed, in this way was possible to highlight the inference time distribution among the selected devices.

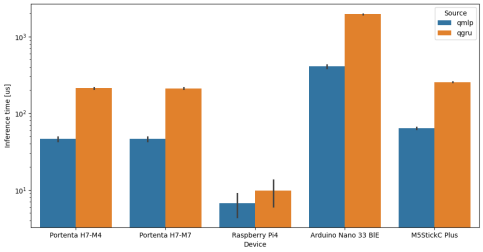


FIGURE 3.15: The latency of the MLP and GRU models on different hardware platforms are reported in the current bar chart, where blue bars showcase the average and standard deviation related to the quantized MLP, while orange bars refer to the quantized GRU model. For each microcontroller, the experiments were repeated 10,000 times for statistical significance. The boards considered for the experiments are reported in Table 3.11, providing technical specifications.

This approach leverages a custom TinyML compiler to enable energy-efficient AI models on low-power edge devices, ensuring real-time detection of microplastics with minimal latency crucial for timely environmental monitoring.

# Chapter 4

# Interpretation of results and explainable AI for healthcare

## 4.1 State of the art

In medical applications, explainability and interpretability of Artificial Intelligence (AI) models are paramount. Clinicians must not only rely on the predictive performance of AI systems but also understand the reasoning behind these predictions to make informed decisions and foster trust. Explainable AI (XAI) addresses this necessity by providing insights into the model's decision-making processes, either through statistical methods or by embedding interpretability into the model design.

Statistical approaches, such as SHAP, LIME, CAM etc., are widely used to explain AI outputs by quantifying the importance of individual features. These methods help clinicians understand which variables most influence predictions, offering transparency and increasing trust in AI systems. However, these post hoc techniques are computationally expensive, limiting their deployment in real-time or resource-constrained environments.

By contrast, designing AI systems with built-in interpretability offers an alternative. For example, layered meta-learning models divide complex tasks into sub-tasks managed by specialized models and aggregate their outputs through a meta-learner, ensuring traceability of decisions. Similarly, the Tandem framework enhances medical image segmentation by combining segmentation predictions with confidence maps, enabling clinicians to evaluate the reliability of results directly. These "by design" methods embed interpretability within the model architecture, ensuring transparency while reducing the overhead of additional explanatory computations.

A critical innovation in combining explainability and efficiency is Knowledge Distillation (KD). KD allows smaller, more efficient *student* models to replicate the performance of larger, more complex *teacher* models. This technique is particularly valuable in healthcare, where computational resources may be limited. According to [125], KD methods can be categorized as follows:

- **Response-based Knowledge:** This approach focuses on aligning the *student* model's outputs with the *teacher*'s predictions using techniques like minimizing Kullback-Leibler (KL)-Divergence between logits [126, 127].
- **Feature-based Knowledge:** Here, the *student* model learns from intermediate feature representations of the *teacher*, as introduced in Fitnets [128]. While this method often enhances performance, it incurs higher computational costs [129].

- **Relation-based Knowledge:** This focuses on capturing relationships between data elements, such as instance-to-instance or feature-to-feature relations, helping the *student* model internalize complex patterns [130, 131].

Feature-based Knowledge Distillation (FD) is particularly relevant in healthcare for its ability to transfer both performance and interpretability. FD has been extended to explainability distillation paradigms, integrating KD principles with XAI requirements. Such methods allow smaller models to inherit the *teacher*'s explainability alongside its performance, making them suitable for practical applications in critical fields like healthcare.

Despite these advancements, challenges remain, particularly the computational expense of combining KD and XAI. Methods like CAM, Grad-CAM [132, 133], and auxiliary models [134] often require significant resources. Recent approaches address these limitations: Rao et al. [135] introduced adaptive teacher-student learning to reduce model parameters, while Sha et al. [136] employed KD in federated learning to enhance privacy and computational efficiency. Similarly, Kim et al. [137] optimized KD for high-resolution pathological images, balancing performance and computational cost.

In the healthcare context, FD has demonstrated potential in creating efficient, interpretable models. For example, transferring intermediate features from a *teacher* to a *student* model has proven effective in reducing training complexity while preserving transparency [138]. This capability is crucial in medical scenarios, where models often operate in resource-constrained environments or directly on edge devices, as seen in real-time glycemic event prediction.

In conclusion, explainability and Knowledge Distillation are complementary strategies addressing the dual challenges of performance and transparency in healthcare AI. By integrating interpretability into model design and leveraging KD for efficient knowledge transfer, these methods are paving the way for scalable, explainable AI systems in medical applications.

## 4.2 Dataset

In this section, a brief overview of the datasets employed in the subsequent analyses is provided. Each dataset will be described in terms of its origin, scope, and key characteristics. Presenting these datasets at the outset will help establish a clear context for the analyses to follow and underscore their relevance to the research questions at hand.

**Ohio T1DM** The Ohio T1DM dataset was initially available to participants in the first and second Blood Glucose Level Prediction (BGLP) Challenge in 2018 and 2020 and then became publicly available to other researchers. In this work, we consider the original format [139] and its expansion [140] as a single dataset. It contains eight weeks of data concerning continuous glucose monitoring, insulin, physiological sensor, and self-reported life-events of twelve adults suffering from T1D (five females and seven males, aged between 20 and 60, each using the Medtronic $Enlite^{TM}$ CGM sensor and a fitness band), all following a Continuous Subcutaneous Insulin Infusion therapy(CSII). More detailed information about the dataset can be found in [139, 140].

We decided to pursue a univariate approach, so CGM sensor data is used alone as an input feature of the proposed model. In order to test the multivariate variant of the models, and provide a fair comparison between different approaches, we

utilized only the features that are in common between the datasets; furthermore, in order to develop a system as autonomous as possible and to reduce the burden on the patient, we only considered the features collected by sensors and without the direct involvement of the user. After this selection, the four considered features are CGM sensor read values, injected insulin, skin temperature, and galvanic skin response.

**Private Validation Dataset (UCBM)** The Unit of Endocrinology and Diabetology of Campus Bio-Medico University (UCBM) Hospital provided anonymized CGM data of five T1D patients (all males), all using Dexcom G5 CGM sensor, aged between 32 and 43 (average $38.6 \pm 5$), HbA1c between 5.7 and 8.4, weight between 67 and 95 kg, daily insulin requirement per kg between 0.07 and 0.85 UI/Kg/die (average $0.49 \pm 0.29$). Three patients use CSII, whereas two follow a Multi-Injection Therapy (). Every patient was monitored for a period ranging from 3 to 14 days (average $8 \pm 3.8$), for a total of 40 days, during which they regularly performed physical activity. Predicting glucose levels of T1D patients during physical activity is particularly tough due to quick variations occurring [36]. It is worth noting that the patients from the UCBM dataset utilize a different CGM sensor than patients from the public dataset.

**2D immunofluorescence ADPKD** Is a collection of RGB immunofluorescence images of human tubules engineered from epithelial cyst-lining cells affected by Autosomal Dominant Polycystic Kidney Disease (ADPKD) [141]. Each image is paired with a binary image indicating the shape and size of cysts within the tissues. We adhered to the preprocessing methods described in the paper. For simplicity, we divided the dataset into a fixed train-validation-test split, ensuring that images from the same tubule were kept separate as recommended by the authors. The dataset providers committed to making it available to anyone upon request for reproducibility.

**3D liver Tumor LiTS** Liver Tumor Segmentation Benchmark (LiTS) [142], consists of 3D CT scans of liver tumors. For this dataset, we followed the preprocessing steps proposed in [143]. Both datasets exhibit high variability in the size of detectable objects, and each acquisition may contain many target objects.

**Lung X-rays** It is a publicly available clinical dataset of lung X-ray images, consisting of four classes: Covid (3,616 samples), Normal (10,192 samples), Lung Opacity (6,012 samples), and Viral Pneumonia (1,345 samples), for a total of 21,165 samples [144, 145]. Each image initially has a resolution of 299×299 pixels and is paired with a 256×256 pixel segmentation mask of the lung.

**UVA/Padova simulator** The UVA/Padova simulator [146] is a widely recognized *in silico* tool for Type 1 Diabetes (T1D) research, providing a virtual environment in which researchers can test and refine glycemic control strategies before progressing to clinical trials. By incorporating validated mathematical models of glucose–insulin dynamics in the human body, it captures critical factors such as insulin sensitivity, carbohydrate metabolism, and daily variability. Users can specify individualized meal scenarios, defining the carbohydrate content and timing of meals, while the simulator automatically administers a basal–bolus insulin regimen based on each

subject's insulin-to-carbohydrate ratio. This level of granularity allows for rigorous testing of various glycemic control approaches, including conventional therapies and advanced closed-loop algorithms, under standardized and reproducible conditions. Furthermore, the simulator has gained broad acceptance as a reliable pre-clinical framework, bridging the gap between purely theoretical work and real-world patient studies, and thereby accelerating the development of safer, more effective treatments for T1D.

## 4.3 Tandem: a Confidence-based Approach for Precise Medical Image Segmentation

The advancement of AI technologies and machine learning algorithms and their integration into medical imaging and decision support systems is ushering in a new era of precision medicine, enabling more accurate diagnoses and treatments [147].

Furthermore, this technology has shown exceptional proficiency in detecting complex medical conditions, such as cysts in tissues affected by Autosomal Dominant Polycystic Kidney Disease (ADPKD), as well as tumor masses. There are still challenges because these objects come in a wide range of sizes, which makes it difficult for neural networks. In this situation, small objects have little impact on the traditional cost function used to train the models, which makes them likely to ignore clinically significant objects.

Based on previous studies that have used U-Net-like models to segment variable-sized clinical targets in tissues affected by ADPKD, we aim to enhance these models with an automatic confidence estimation tool. Our approach in the work presented in [148] involves a flexible architecture integrating prediction refinement into deep segmentation models. This aims to reduce prediction errors and assist clinicians by providing a mechanism to measure the model's confidence in its predictions, regardless of the size of the objects being analyzed and across different regions of the images.

In our study, we emphasize the limitation of the current solutions by focusing on the segmentation of cysts engineered using cells from ADPKD patients and the detection of tumor masses in liver tissues. For the first target, we utilized a dataset from the *Istituto di Ricerche Farmacologiche Mario Negri IRCCS*, containing RGB immunofluorescence images of human tubules engineered from cyst-lining cells. This research builds upon the work of Monaco et al. [149]. For the second target, we selected the well-known Liver Tumor Segmentation Benchmark (LiTS) [150], which consists of CT liver 3D images.

Summing up, this work aims to improve segmentation accuracy for clinical objects by adapting deep learning segmentation solutions to provide a size-invariant self-assessment of their predictions. The main contributions of this study are summarized as follows:

- Introduction of a novel segmentation solution, Tandem (**T**andem **A**nalysis for **N**eural **D**etection and **E**valuation **M**odel), which integrates a classifier with a segmentation model jointly trained to produce a confidence map alongside the usual model prediction.
- Evaluation of the proposed strategy on two different medical image segmentation tasks involving 2D and 3D acquisitions.
- Practical evaluation of the Tandem procedure to generate a reliable confidence map of the predictions to assist clinicians in model evaluation.

We apply our framework to two datasets, encompassing 2D and 3D medical images presented in the section 4.2.

### 4.3.1 Tandem architecture and experimental setup

To improve medical segmentation models and ensure reliable self-evaluation of their predictions, we propose Tandem, an end-to-end pipeline for enriching the output of

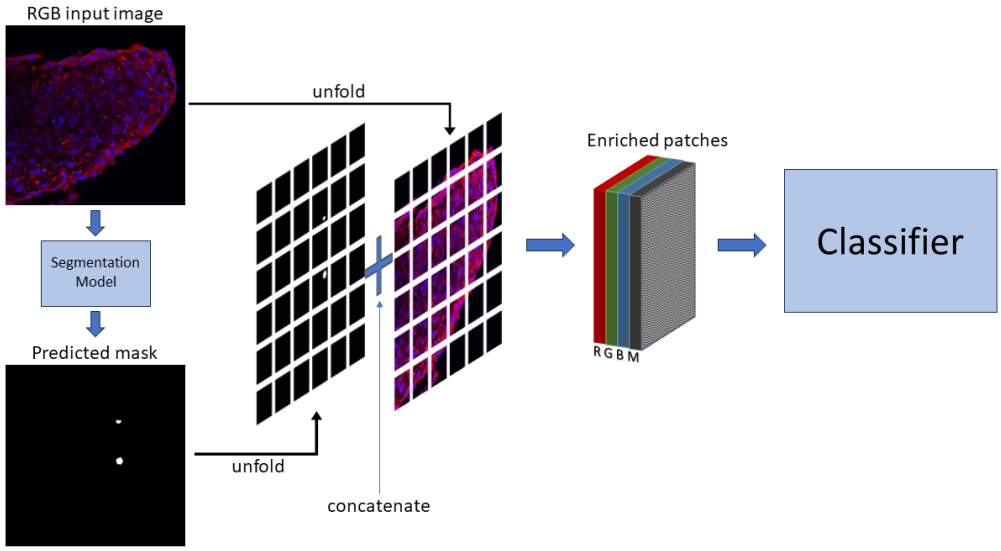


FIGURE 4.1: Pipeline diagram of Tandem

a neural network tailored to segmentation tasks. This is achieved by a *self-evaluation* head that provides a confidence measure for different parts of the prediction. Figure 4.1 illustrates the main components of our framework, which include a segmentation neural network and a classifier to estimate the error of the different parts of the original prediction. The key concepts of this architecture are explained in more detail in the following section.

**Tandem architecture** As previously mentioned, our approach uses classification to improve segmentation results. Our pipeline is not architecture-specific, so the two neural networks for segmentation and classification can be selected according to the specific requirements of the use case.

Starting with an input image of size $D \times H \times W \times C$ (where $D$ is included for 3D images, $H$ and $W$ are the other spatial dimensions, and $C$ is the number of channels), the first model $\mathbf{S}(\cdot, \theta_s)$ produces a prediction with the same spatial dimensions and $C'$ channels, representing the number of classes to segment.

The classifier $\mathbf{C}(\cdot, \theta_c)$ acts then as a sliding window, scanning the segmentation prediction along with the associated input image to detect inconsistencies. It refines the segmentation output by merging its predictions, thus contributing to the final segmentation result. To scan the input image with the classifier, we divide it into non-overlapping square patches of dimensions $d \times h \times w$ (where $d$ is included for 3D images) such that the dimensions along each axis are divided evenly into the corresponding original image dimensions.

The classifier's input is a tensor of size $d \times h \times w \times (C + 1)$, where the channels consist of the original image channels concatenated with the first step's prediction. We refer to these 4-dimensional patches as enriched patches. These enriched patches are then passed to the classifier, which predicts a label for each patch, considering the segmentation prediction. The classifier's output is then a binary prediction for each patch of the input image, which is compared with a ground truth label according to the following rule:

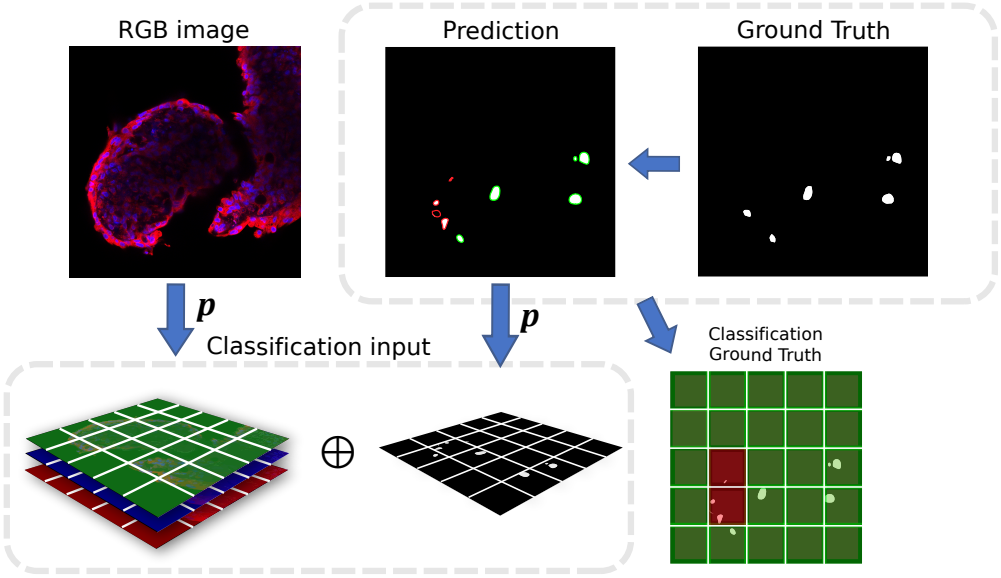


FIGURE 4.2: Visual representation of the generation process of classifier inputs and ground truths.

- **True**: If either the prediction detects a target that is present in the patch ground truth, or if the model finds no target and there is indeed nothing in the ground truth.
- **False**: If the prediction detects a target that is not present in the ground truth, or if the model fails to detect a target.

A target is marked as *detected* if the prediction patch has an Intersection over Union (IoU) of at least $t_{\text{dt}}$ with the corresponding ground truth patch. In our experiments, we set this threshold to 0.4, but this value can be adjusted depending on the specific requirements of the problem. To reduce the prediction noise, we also introduced a size cutoff below which targets within the patch are not considered. This means that any patch containing only targets with a total number of pixels below the cutoff value is treated as empty. Although this cutoff value can be arbitrarily small, we found in our experiments that a well-chosen value helps the model to focus on relevant targets, improving overall performance. The ground truth labelling process used during training is illustrated in Figure 4.2.

The overall model is finally trained on both tasks simultaneously. Let $x$ represent the input image, $y$ the ground truth segmentation, and $y_{\text{cl}}$ the classification ground truth based on the segmentation prediction. The global loss function is defined as:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{s}}(\hat{y}_{\text{s}}, y) + \alpha \mathcal{L}_{\text{cl}}(\hat{y}_{\text{cl}}, y_{\mathbf{cl}}) \\ \hat{y}_{\text{s}} &= \mathbf{S}(x, \theta_{\text{s}}) \\ \hat{y}_{\text{cl}} &= \mathbf{C}(\mathbf{p}[x \oplus \hat{y}_{\text{s}}], \theta_{\text{c}}), \end{aligned} \tag{4.1}$$

where $\mathbf{p}[\cdot]$ denotes the operation of splitting the input into patches and $\alpha$ is a tunable coefficient balancing the contributions of the two-loss components. Since the classification labels $y_{\text{cl}}$ depend on the segmentation model's predictions, the second term of the loss can directly influence the first model, leveraging the additional

TABLE 4.1: Segmentation results for dataset and target size

| | | Pr | | | Re | | | IoU | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Small | Medium | Large | Small | Medium | Large | Small | Medium | Large |
| cyst | Base | 0.485±0.056 | **0.764**±0.046 | **0.799**±0.023 | 0.664±0.109 | 0.748±0.049 | 0.685±0.017 | 0.381±0.022 | **0.605**±0.013 | 0.584±0.011 |
| | Tandem | **0.494**±0.022 | 0.737±0.032 | 0.795±0.044 | **0.691**±0.058 | **0.773**±0.030 | **0.698**±0.039 | **0.403**±0.010 | **0.605**±0.005 | **0.59**±0.022 |
| LiTS | Base | 0.049±0.011 | 0.224±0.016 | **0.537**±0.042 | 0.363±0.052 | 0.556±0.030 | 0.781±0.035 | 0.045±0.010 | 0.190±0.010 | **0.467**±0.035 |
| | Tandem | **0.056**±0.015 | **0.256**±0.039 | 0.483±0.059 | **0.435**±0.082 | **0.600**±0.046 | **0.809**±0.019 | **0.052**±0.014 | **0.217**±0.026 | 0.433±0.045 |

penalty for its errors. The two loss functions $\mathcal{L}_s$ and $\mathcal{L}_c$ can be any losses typically used for segmentation and classification tasks.

Finally, this pipeline produces the first step of segmentation prediction and the pathed correctness classification as output.

**Experimental Setup** **ADPKD dataset.** For the Tandem backbones, we used a U-Net++ as the segmentation model and a ResNet18 as the classifier. The model was trained for up to 30 epochs until convergence on the validation set, using the Adam optimizer with a learning rate of $1 \times 10^{-4}$, tuned with a cosine annealing with a warm restart scheduler. The loss function $\mathcal{L}_s$ combined binary cross-entropy (BCE) and Jaccard loss, utilizing their inverse-weighted versions as proposed in [143] to make the network focus more on small-sized targets. We employed Binary Focal Loss with parameters $\alpha = 0.1$ and $\gamma = 2$ for the classification component. These parameters were determined by a grid search to correct the imbalance of labels in this subtask. The patch size used for the classification inputs is 128 pixels. This value was chosen to be large enough to capture larger cysts, but not too large to be meaningful for smaller cysts.

**LiTS Dataset.** For this dataset, the tandem backbones consisted of a 3D U-Net and a 3D ResNet18 [151]. The training pipeline for the baseline model was inspired by [143]. Specifically, we trained the models for 100 epochs using Focal Loss ($\alpha = 0.25$, $\gamma = 2$) with an inverse weight variation for $\mathcal{L}_s$, and Focal Loss ($\alpha = 0.1$, $\gamma = 2$) for $\mathcal{L}_c$. We employed the Adam optimizer with a learning rate of $1 \times 10^{-4}$, which was reduced by 20% at epochs 50 and 80. The classifier input patch size for this dataset is 32 pixels.

Following these setups, we trained the segmentation models in their baseline and tandem fashions to compare the effects of our method on segmentation performance. Then, we analyzed the classifier's performance to measure its strength in providing insights into segmentation output errors. The following section presents the results, averaging over 5 repetitions to improve statistical significance. All the experiments were conducted on a fixed train-validation-test split as provided by the datasets authors.

All experiments were implemented using the PyTorch framework on an Intel Core i9-10980XE CPU @ 3.00GHz and two NVIDIA RTX A6000 GPUs. Each experiment took approximately 2 hours for the ADPKD dataset and 12 hours for the LiTS dataset. We observed no significant variation in the training time required for baseline models and their tandem counterparts. The code used for the experiments is available online [1].

### 4.3.2 Results and Discussion

Table 4.1 reports the segmentation performance comparison of the models trained alone or within our pipeline for the two datasets. The baseline models mentioned in

[1] https://github.com/simone7monaco/tantem-segmentation

each dataset's respective papers are compared with their Tandem-enhanced counterparts.

Tandem models generally improve recall (Re) and intersection over union (IoU) across most size categories, with particular regard on smaller sizes, with particularly notable enhancements for small and medium-sized objects. In the ADPKD dataset, Tandem slightly improves precision (Pr) for small objects (1.85% increase), performs marginally worse for medium (3.53% decrease) and gets roughly equivalent results for large objects compared to the Base model. For recall, Tandem consistently improves performance for small (4.07% increase), medium (3.34% increase), and large objects (1.9% increase). IoU results also indicate that Tandem outperforms the Base model for small objects (5.78% increase), matches the Base model for medium objects, and slightly improves for large objects (1.03% increase).

In the LiTS dataset, Tandem shows better precision for small (14.29% increase) and medium-sized objects (14.29% increase), but a lower precision for large objects (10.05% decrease) than the Base model. Recall improvements are significant for small (19.83% increase), medium (7.91% increase), and large objects (3.58% increase) in Tandem. IoU results are better for small (15.56% increase) and medium-sized objects (14.21% increase), but slightly lower for large objects (3.38% decrease) with Tandem than the Base model. Notably, the performance on this dataset is significantly poorer than on the previous one, particularly for small and medium targets, highlighting the task's increased complexity; while this dataset is commonly used for liver detection benchmarks where models perform well, few studies focus on tumor detection.

Overall, the Tandem model effectively enhances recall and IoU, which are crucial for medical segmentation tasks, particularly for smaller and medium-sized objects. However, there is a trade-off with a slight reduction in precision for larger objects, suggesting a need for further tuning or model adjustments to achieve optimal performance across all size categories. Finally, considering segmentation capability alone, it is important to note that some of the observed variations are relatively minor, indicating that the performance of a network trained with the Tandem procedure is quite comparable to one trained with a baseline approach. The crucial distinction, however, is that the Tandem method not only avoids performance degradation but also offers slight improvements, while additionally providing a self-evaluation of the model predictions.

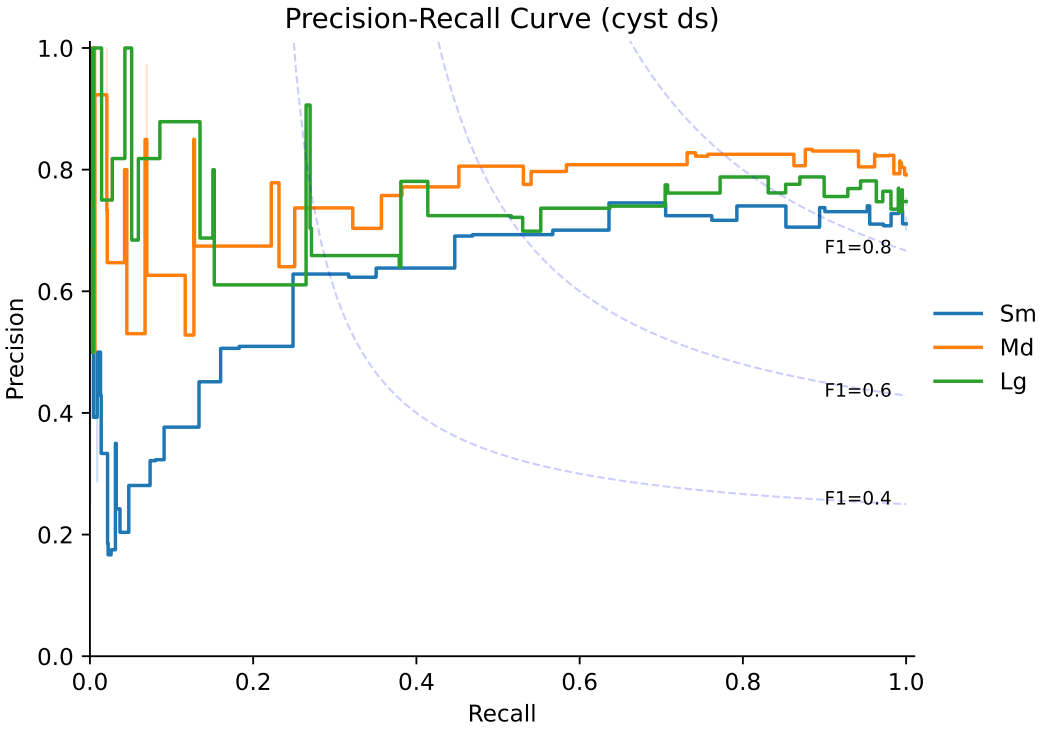


FIGURE 4.3

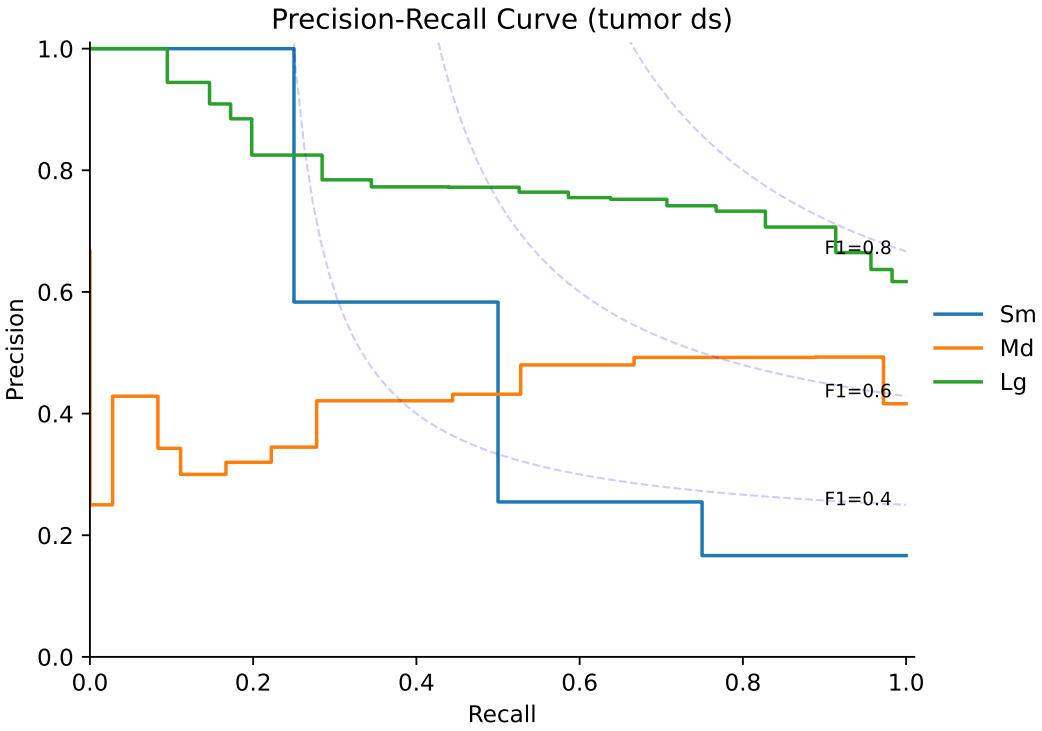


FIGURE 4.4

FIGURE 4.5: P-R curves for the Tandem confidence maps both on LiTS and ADPKD datasets.

To quantify the Tandem method's capacity for providing reliable confidence maps,

Figure 4.5 displays the precision-recall curve of the Tandem models on the classification subtask, categorized by target sizes. This plot illustrates the tradeoff between the precision and recall metrics on test patches as the classifier threshold varies to determine whether a prediction is considered *correct* or *wrong*. The dashed lines represent the ISO-F1 curves, indicating points on the plane associated with specific F1 scores. Each patch is labelled according to the size category of the object it contains, as per the ground truth.

In the ADPKD dataset, the model performs best for medium-sized targets, followed closely by large and small sizes. The performance across all three sizes is comparable, indicating that the confidence maps generated by the Tandem method are stable and reliable regardless of the detected cyst size. The three curves exhibit a slightly horizontal orientation, with the medium-sized targets achieving a maximum precision of 0.8 as they approach the best recall values. Large and small targets follow closely, reaching around 0.7 maximum precision. This horizontal orientation of the PR curves suggests that the classifier maintains relatively high precision across a wide range of recall values, indicating a balanced performance in detecting positive and negative instances. The high F1 values achieved in these regions of the plot further underscore the overall effectiveness of the classifier.

In the tumor dataset, the model shows the highest performance for large-sized targets, with the corresponding PR curve positioned at the top. This curve is not completely flat but slopes downward slightly. When the F1 score reaches 0.8, the recall is approximately 0.9, and the precision is around 0.8. This indicates robust performance for large tumors, maintaining a high balance between precision and recall, and effectively managing both true positive detections and minimizing false positives.

The PR curve for medium-sized targets is slightly lower and exhibits a moderate increase. Although it does not achieve an F1 score of 0.8, it reaches its best performance with an F1 score above 0.6, with both precision and recall around 0.6 at the highest recall values. This indicates a reasonable performance for medium-sized tumors, but highlights that there is still room for improvement to achieve higher precision and recall.

For small-sized targets, the PR curve shows a steep decline, with the best performance, or "knee," occurring at an F1 score of about 0.5, with recall at 0.5 and precision at 0.6. This suggests that detecting small tumors remains a challenging task for the model, as indicated by the lower F1 scores and the sharper drop in performance metrics.

Overall, the variations in performance across different target sizes in the tumor dataset highlight the increased complexity of this detection task. Despite these challenges, the high F1 scores achieved in specific regions of the plot further reinforce the classifier's overall effectiveness.

## 4.4 Development of an Explainable Deep Learning-Based Decision Support System for Blood Glucose Levels Forecasting in Type 1 Diabetes Using Edge Computing

Two main issues still limit the real-life utilization of a machine learning-based decision support system that predicts blood glucose on an edge device. First, the predictive algorithms that perform best are usually those that are more complex and take as an input several heterogeneous types of data in addition to CGM, such as insulin injections or amount of carbohydrates ingested by the patient [152]. This makes it difficult to implement these models on a microcontroller and, above all, takes for granted the integration of the device to be developed with an insulin pump that provides information on basal insulin, insulin boluses and ingested carbohydrates, in addition to the CGM device [153]. Second, there is no guarantee that a patient is willing to make therapeutic decisions based on information provided by a deep learning algorithm, which is by nature a 'black box'. User acceptability of machine learning algorithms is one of the challenges of eXplainable AI (XAI), a branch of Artificial Intelligence whose aim is to develop methods that make machine learning algorithms interpretable.

This work, presented in [154], aims to overcome the practical limitations of real-life utilization of machine learning decision support system to predict the blood glucose level by introducing the following main contributions:

1. the development of an LSTM-based deep learning model that, taking a 30-minute time sequence of CGM values as an input, predicts the glycemic value in advance of PH = 30 minutes with a univariate approach. In addition, in order to test the limits of the implemented framework, 11 additional PHs, ranging from 5 to 60 minutes, are investigated;

2. the utilization of Local Interpretable Model-Agnostic Explanations (LIME) [12], one of the most prevalent algorithms for explainability, which aims to conduct an analysis of model explainability. This is done to enhance the interpretability of the model for the user, making it easier to understand how and why the model arrives at its conclusions;

3. the implementation of the model on a microcontroller to simulate the communication and reception of data from a CGM sensor and perform inference on an edge device.

In the following sections, we have first introduced the data used to validate our approach, detailing the deep learning model employed and outlining the experimental procedure followed for its training. Subsequently, we described the tools implemented for conducting the explainability analysis. Finally, we elaborated on the devices and the experimental setup utilized for edge inference.

### 4.4.1 Data generation and Model Training

Using the UVA/Padova simulator [146] presented in section 4.2 we have generated 30 days of CGM data of 10 adult T1D subjects. In order to make the dataset more realistic, we followed the approach presented in [155]: for each simulated day, we randomly varied the number of meals between 3 and 5, taking as a reference the amount of carbohydrates presented in the Dietary Reference Intakes for Carbohydrate [156] and varying this amount by a random quantity of $\pm 10\%$, while randomly

varying the consumption time by $\pm$ 30 minutes. In order to simulate human error on carbohydrate counting and to force hypo/hyperglycemic events and, thus, to make data more difficult to predict [157], we randomly modified the optimal insulin bolus computed by the simulator by adding random noise sampled from a uniform distribution in the interval $[-3, 3]$. Finally, we adopted a simulated *Guardian RT* sensor, in order to make the generated CGM data even more similar to those observed in real life.

The structure of the proposed model and the choice of hyperparameters was designed and optimized in our previous study [155] and consists of an input-layer taking in a time series of 30 CGM-data, corresponding to 30 minutes, an LSTM-layer consisting of 30 units with hyperbolic tangent (tanh) activation function, and a dense layer consisting of 15 units with Rectified Linear Unit (ReLU) activation function. The output, depending on the PH chosen in the training phase, is a single numerical value corresponding to the CGM value predicted in advance of PH minutes. It is worth stressing that the goal of this study is not to develop the best possible glycemic predictor, but rather to investigate the explainability and the edge implementation of a well-established method.

The training of each model was performed in order to specialize each of them on a specific subject, and consists of two phases. The first is a Leave-1-Patient-Out training in which the model is trained using data from all but one subject (the data being divided into 70% Training and 30% Validation), while the second is a Precision Medicine training in which the model is re-trained on the previously excluded subject on which the test will also be performed (the data being split into 50% Training, 20% Validation and 30% Test). It is proven in the literature that this methodology yields the best results in terms of performance [155]. Before being given as an input to the neural networks, the data were pre-processed according to the Z-Score standardization; in particular, for the first training, mean and standard deviation were calculated on all the input data, whereas for the second training they were calculated only on 70% of the data of the subject of interest, corresponding to the Training and Validation data (50% + 20%). Furthermore, in order to prevent overfitting, three techniques were adopted: Early Stopping, L2 Regularization and Dropout. Thus, a total of 120 models were trained, each specialized in predicting the blood glucose of a specific subject with a PH of 5 to 60 minutes (thus, 12 models for each of the 10 patients). The model was implemented in Python using the open-source libraries of Tensorflow and Keras.

The performance of the model was evaluated in terms of Root Mean Square Error (RMSE) and Clarke Error Grid Analysis (CEGA). The former is a metric that evaluates the numerical accuracy of the predictions, and is defines as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}[y(t_i + PH) - \tilde{y}(t_i + PH|t_i)]^2}{N}}$$

being $y$ and $\tilde{y}$ the true glycemic value and its prediction performed with a prediction horizon $PH$ at time $t_i$, averaged over all the $N$ predictions. The latter is a metric specific to the blood glucose prediction task that relates model predictions to actual values, discerning predictions that would lead to correct clinical decisions from those that would lead to incorrect or seriously incorrect clinical decisions, such as not recognizing a hypo/hyperglycemic event or recognizing a hypo/hyperglycemic event that is not really occurring; it can be regarded as a measure of clinical accuracy of the predictions.

### 4.4.2 Explainability Analysis

LIME [12] is an agnostic Explainability algorithm, i.e., applicable to all types of machine/deep learning models, which provides "local" explanations, i.e., referring to a single model prediction and not to the model itself. Specifically, it generates a linear model around the input associated with the prediction and provides the "Feature Importance" (FI) as an explanation, i.e., the ranking of the features, namely the inputs, most relevant to the model in making that prediction. In particular, to each feature (namely the input CGM data) it assigns a positive weight if this pushed the prediction towards a high blood glucose value, whereas it assigns a negative weight if this pushed the prediction towards a low blood glucose value. The greater the absolute value of the weight, the greater the contribution of the input data to the prediction. The analysis carried out in this study consists of calculating, for the models with PH = 30 and 60 minutes, the average FI of the input separately in two situations: in the 2 hours following each meal and in the rest of the dataset (i.e., fasting), in order to highlight any recurring patterns.

### 4.4.3 Inference on Edge Device

In order to investigate the feasibility of the implementation on the edge device, models predicting blood glucose with a PH = 30 minutes were converted into *tflite* format in order to be executed by the microcontroller. Specifically, the device selected to perform inference is the Arduino Portenta H7 [158], based on a dual Arm Cortex M4 and M7 processor. This microcontroller was chosen for its ability to perform calculations on data in float32 format, which allows the converted models to be loaded without having to quantize them and thus preserving their performance. In addition, this device was also chosen for its integrated Bluetooth Low Energy (BLE) module, which is essential for simulating the device's communication with a device providing CGM data. The BLE architecture involves two main actors: the Peripheral, i.e., the device that has the data (e.g., a sensor) and the Central, i.e., the device that requests the data. Specifically, in our experimental setup BLE communication is implemented between a personal computer (simulating the CGM sensor, the role of which is to provide a CGM value when requested), Arduino Portenta H7 and an application to visualize the prediction. The personal computer sends data and simulates the behavior of the CGM sensor; however, for reasons related to the use of libraries, it acts as a Central entity, modifying the Feature defined on Portenta H7, which acts as a Peripheral. We leveraged the Python library Bleak for generating Central entities. This enables establishing connections to multiple Peripherals, facilitating the reading and updating of characteristics. Specifically, the PC writes the CGM values to a specific characteristic defined by Portenta H7, called the "Receive Characteristic", while reading the prediction made on the edge on another characteristic, also defined by Portenta H7, called the "Send Characteristic". The library used for BLE communication is ArduinoBLE [159]. In order to handle the different data rate existing in commercially available CGM sensors, a linear interpolation function was implemented on the Portenta H7 which, once the number of data needed to cover the 30-minute time interval has arrived, generates a sequence of 30 values to be given as an input to the model to perform inference. Finally, M5StickC Plus is the microcontroller used to simulate the data reception application. The implemented code contains a callback function that is triggered every time new data is written to the Characteristic. It has its own screen, which is used to display the read value. A schematic of the architecture can be found in Figure 4.6.

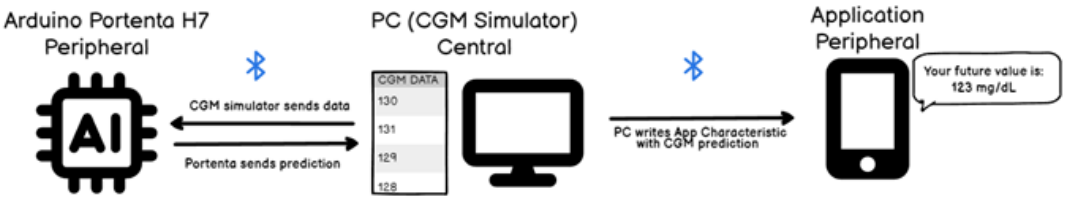


FIGURE 4.6: Schematic diagram of the simulation performed with PC (CGM device simulator), Arduino Portenta H7 for prediction calculation, and App to visualize the data

### 4.4.4 Results and Discussion

Table 4.2 shows the results in terms of RMSE (mg/dL) and CEGA percentages on all subjects at varying PHs considered.

| PH | RMSE | Clarke Error Grid Analysis | | | | |
|---|---|---|---|---|---|---|
| [min] | [mg/dL] | A (%) | B (%) | C (%) | D (%) | E (%) |
| 5 | 2.13 | 99.99 | 0.01 | 0 | 0 | 0 |
| 10 | 4.19 | 99.92 | 0.07 | 0 | 0.01 | 0 |
| 15 | 6.84 | 99.29 | 0.62 | 0 | 0.09 | 0 |
| 20 | 9.59 | 97.41 | 2.26 | 0 | 0.33 | 0 |
| 25 | 12.09 | 94.58 | 4.71 | 0 | 0.70 | 0 |
| 30 | 14.22 | 91.95 | 7.05 | 0 | 1.00 | 0 |
| 35 | 16.10 | 88.94 | 9.86 | 0 | 1.20 | 0 |
| 40 | 17.98 | 85.61 | 12.76 | 0 | 1.63 | 0 |
| 45 | 19.61 | 82.82 | 15.50 | 0 | 1.68 | 0 |
| 50 | 21.28 | 79.55 | 18.54 | 0 | 1.90 | 0 |
| 55 | 22.34 | 77.62 | 20.27 | 0 | 2.11 | 0 |
| 60 | 23.56 | 74.60 | 22.94 | 0 | 2.46 | 0 |

TABLE 4.2: RMSE and average CEGA percentages on all subjects, varying PH

Competitive results are shown (14.22 mg/dL on average for PH = 30 min) when compared to the literature [152, 160], especially considered that better results were obtained from much more complex models with a multivariate approach. A graphical depiction of the glucose forecasting with a PH of 30 minutes compared to the reference glycemic values of the 10 in silico patients is presented in Figure 4.7. Furthermore, very good results were obtained with regard to the percentages belonging to the different zones of the Clarke Error Grid: as can be observed in the tables, most of the predictions are within zones A and B, with very few predictions (always less than 2.5%) within zone D. In the cases of $PH \leq 10$ minutes, it can be observed that all hypo/hyperglycemic events are predicted: although the short notice may not allow them to be avoided, their recognition can certainly help preventing severe hypo/hyperglycemic episodes [161].

Table 4.3 focuses on the performance of the models predicting CGM values with a PH of 30 minutes, comparing the results obtained by performing inference on Arduino Portenta H7. With regard to the inference performed on the microcontroller, as expected since no quantization was carried out, the performance in terms of RMSE is in line with that obtained on the personal computer: the differences are due to the fact that the inference results are calculated only on a portion of the test set data.

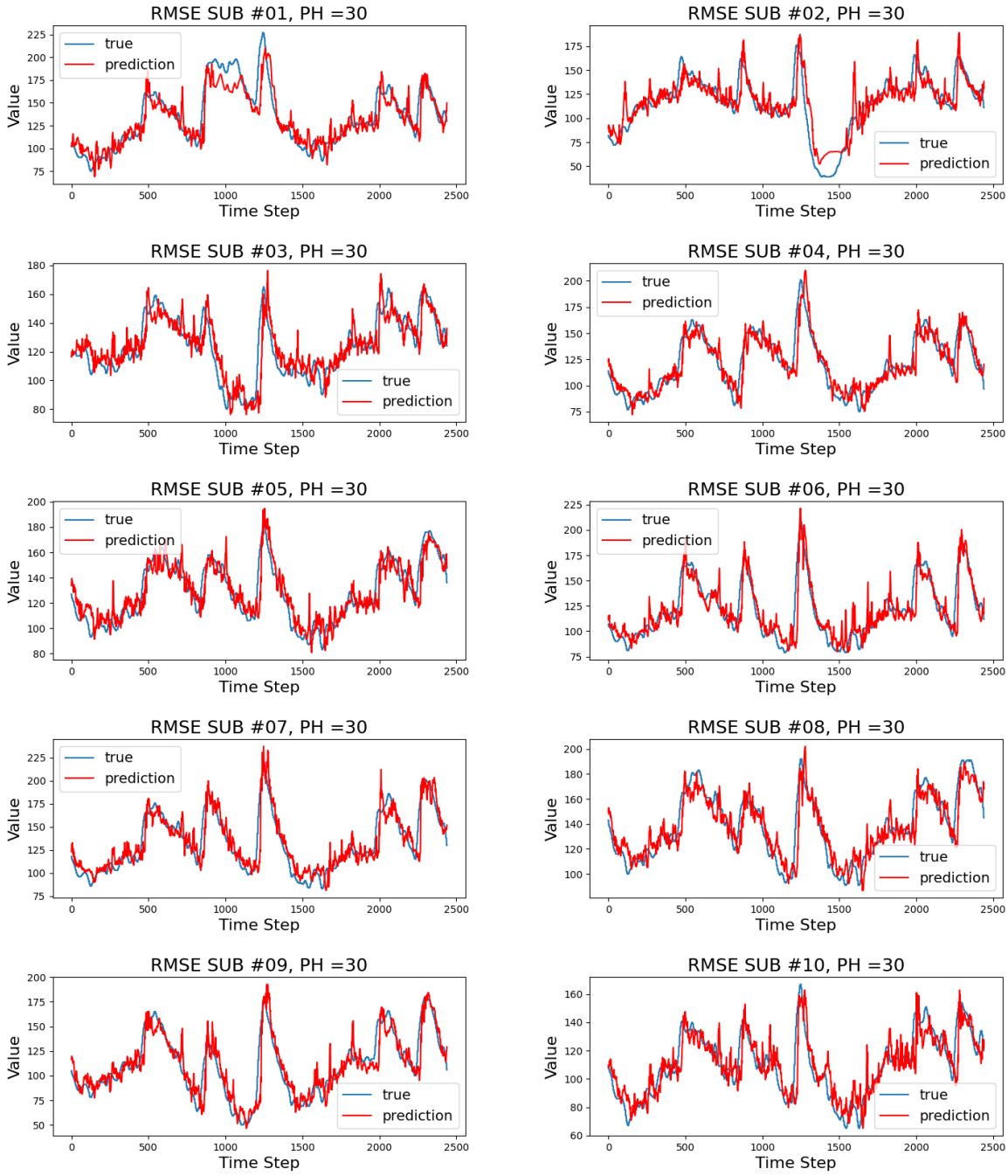


FIGURE 4.7: Graphic examples of glucose levels forecasting for the 10 in silico patients.

| Subject | RMSE | RMSE Portenta | Diff | Clarke Error Grid Analysis | | | | |
|---|---|---|---|---|---|---|---|---|
| | [mg/dL] | [mg/dL] | [mg/dL] | A (%) | B (%) | C (%) | D (%) | E (%) |
| #01 | 15.79 | 15.04 | 0.75 | 90.98 | 7.77 | 0 | 1.24 | 0 |
| #02 | 14.72 | 11.52 | 3.20 | 90.29 | 8.93 | 0 | 0.78 | 0 |
| #03 | 12.57 | 9.88 | 2.69 | 95.28 | 4.52 | 0 | 0.20 | 0 |
| #04 | 13.71 | 10.31 | 3.40 | 93.05 | 6.09 | 0 | 0.86 | 0 |
| #05 | 12.79 | 9.9 | 2.89 | 94.27 | 4.92 | 0 | 0.81 | 0 |
| #06 | 15.51 | 10.65 | 4.86 | 90.23 | 8.92 | 0 | 0.85 | 0 |
| #07 | 15.65 | 11.45 | 4.2 | 92.11 | 7.18 | 0 | 0.71 | 0 |
| #08 | 13.22 | 9.89 | 3.33 | 94.84 | 5.01 | 0 | 0 | 0 |
| #09 | 15.16 | 12.16 | 3.00 | 87.78 | 10.00 | 0 | 2.23 | 0 |
| #10 | 13.11 | 9.82 | 3.29 | 90.69 | 7.16 | 0 | 2.15 | 0 |

TABLE 4.3: RMSE of models with PH = 30 minutes by inference performed on PC and Arduino Portenta H7, absolute difference between respective performances; CEGA percentages of models with PH=30 minutes

With regard to the communication and inference time, they are always around 1 second. This is largely suitable in a practical application, since most commercial CGM devices have a sampling period of at least 1 minute [162].

Figure 4.8 presents the result of the Explainability analysis for Subject #01, who is also representative of the remaining subjects. On the top left are the average FIs after meals with a PH of 30 minutes and, as can be observed, the most recent values have strongly positive FIs. The reason could be that the most recent CGM values serve the model to understand the trend in blood glucose, which on average tends to rise after meals. In contrast, older values have strongly negative FIs: the reason could be that, since the corresponding CGM input values (center of Figure 2) are still high, the model has been receiving high values for at least 10 minutes, so it expects that after 30 minutes blood glucose will start to fall and therefore associates a negative FI with these values. A mirror-like behavior is shown on the bottom left, where average fasting FIs with a PH of 30 minutes are shown. In this case, the most recent input values to the model have negative FIs, as they tend to be lower values than the average. Slightly less recent input values, on the other hand, have positive FIs as the model, having received more than 10 values (corresponding to 10 min) of CGM tending to be low, expects blood glucose to rise again within 30 min. On the right side, two graphs that mirror those depicting a PH of 30 minutes are reported, reflecting a PH of 60 minutes. In this case, a more uniform distribution of FIs was expected. Actually, this is not the case, as for both PHs the models show a very similar behavior, with only the most recent 10 minutes of CGM showing recurring patterns in terms of FIs.

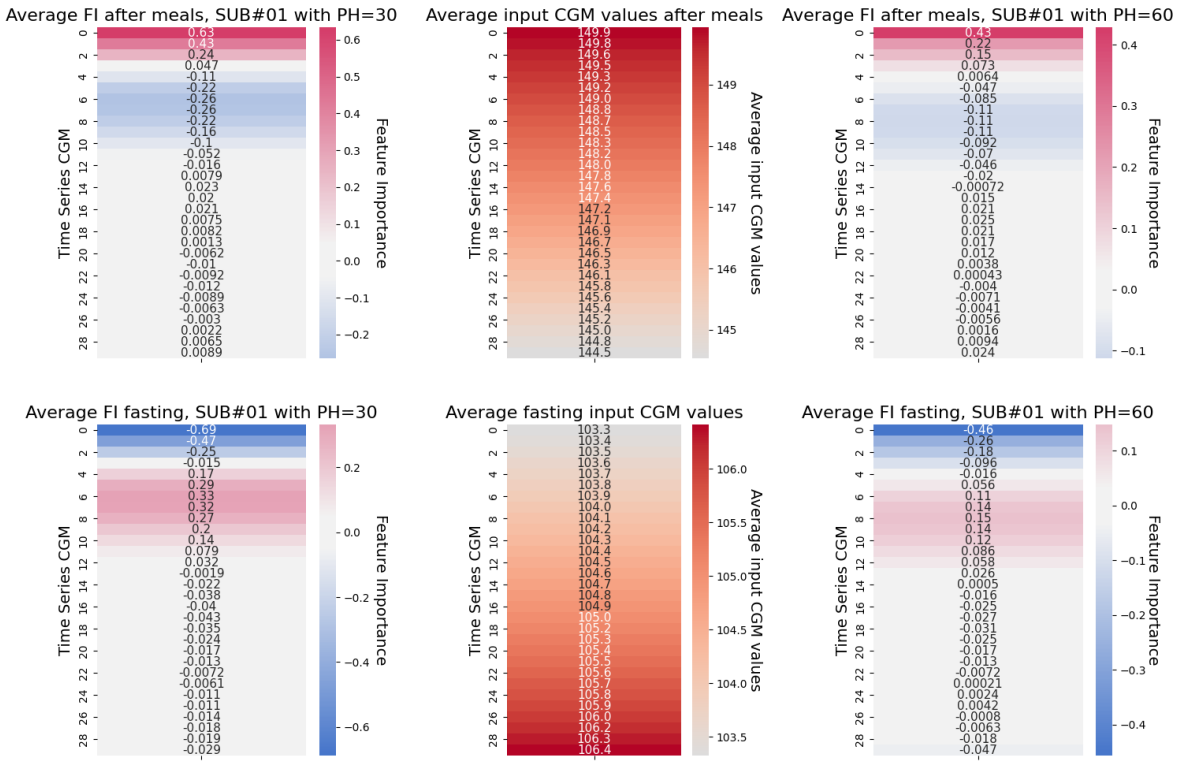


FIGURE 4.8: Explainability Analysis for Subject #01. Top: average FI during fasting; bottom: average FI after a meal.

## 4.5 Feature-Based Knowledge Distillation for Explainable Detection of Pulmonary Diseases

The computational cost of AI systems in healthcare is a significant challenge for resource-constrained environments where traditional, expensive technologies are not feasible. In order to lower the inference computational burden of any NN, Hinton et al. [126] introduced the Knowledge Distillation (KD) paradigm. This technique compresses the knowledge learned by a large *teacher* model into a smaller, more efficient *student* model. By enabling high performance with lower resource requirements, KD aligns with the principles of frugal innovation, making advanced AI solutions accessible and sustainable in low-resource settings.

However, especially in the health sector, transparency is a key factor. The opacity of AI systems in healthcare, often seen as "black boxes", raises significant concerns about reliability, accountability, and ethical practices, hindering trust and safety. In this context, numerous studies in the field of eXplainable Artificial Intelligence (XAI) [163] have proposed methods to enhance model explainability through Explainability Distillation (ED) techniques [134, 164–167]. Nonetheless, these techniques often consider *direct* approaches, necessitating to generate and target heatmaps or other explainable representations during the training process. These approaches entail computational costs, contrasting with the principles of *frugal innovation*, which aim to reduce complexity and lower the costs of implementation and development.

The aim of this work is to introduce an unprecedented approach leveraging Feature-based Distillation (FD) to effectively transfer explainability from a sophisticated, large-scale model to a more compact one. Our innovation lies in the simultaneous distillation of intermediate features and logits, a method not previously explored in the domain of XAI. Unlike previous approaches, we do not directly compare the explanations during training, thus maintaining low computational costs. Traditional methods generate explanations using computationally intensive techniques, such as Class Activation Maps (CAM) [132] or Grad-CAM [133], at each training step. Instead, explanations in our approach are used exclusively to test the explainability transfer of our model, resulting in significantly reduced computational expenses. This approach aims to maintain the alignment of the student model's explainability with that of the teacher model, pushing the boundaries of current methodologies in the field.

Crucially, our contribution does not lie in introducing a new method for KD per se, but in the innovative application of existing KD techniques to specifically preserve the explainability of the teacher model within the student model, tailored to our task. By leveraging standard KD methods to maintain explainability features, our framework is versatile and has the potential to be extended to other contexts where explainability is of paramount importance.

To validate our approach, we implemented it for detecting lung pathologies from chest X-ray (CXR) images using a comprehensive public dataset from [144,145]. This dataset offers two significant advantages: (1) it comprises four categories of labeled CXR images — COVID-19, lung opacity, viral pneumonia, and absence of disease — which have not been extensively analyzed in previous research, highlighting the scalability and adaptability of our method; (2) it includes segmentation masks of lung regions, meticulously validated by expert radiologists. These masks not only enhance the dataset's reliability but also enable a unique opportunity to systematically analyze their impact on classification accuracy and model explainability. Such radiologist-validated segmentation masks are rare in the field and provide a robust

basis for exploring how localized lung features contribute to both predictive performance and interpretability, further demonstrating the dataset's utility in advancing AI-driven medical diagnostics.

Our key contributions can be summarized as follows:

- Developing and training a *VGG19* model [168] for a four-category classification task, achieving superior performance over current state-of-the-art methods.
- Introducing and analyzing the novel impact of segmentation masks on both the classification and explainability outcomes of our models.
- Transferring the knowledge and feature maps from the *VGG19* model [168] to a significantly smaller *MobileNet* model [169] (approximately 5.4 times smaller), and assessing its performance with various sizes of training data (10%, 25%, 50% or 100% of the dataset), evaluating whether distillation-based methods offer advantages when data is scarce.

In essence, our distillation technique demonstrates exceptional proficiency in transferring both the knowledge and explainability from a larger model to a smaller one, surpassing existing methods in terms of both efficiency and effectiveness, particularly under conditions with limited training data.

The remainder of this paper is structured as follows: Section 2 reviews related works, providing the context and background for our approach. Section 3 describes the materials and methods used, including the dataset, model architectures, and the proposed framework. Section 4 presents the experimental results and discusses their implications in detail. Finally, Section 5 concludes the paper with a summary of key findings and potential directions for future research.

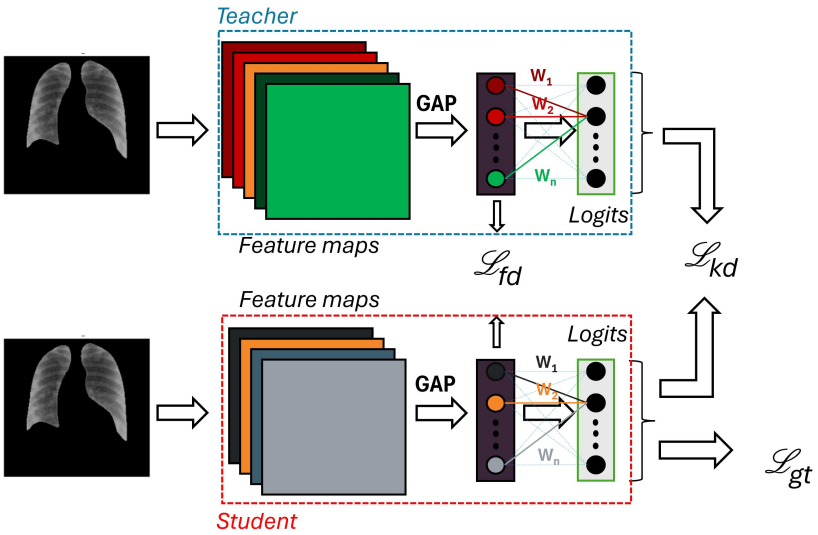


FIGURE 4.9: Proposed Method's Pipeline.

### 4.5.1 Materials and Framework description

**Dataset preprocessing** We used the public clinical dataset of lung X-rays presented in 4.2. We have resized the original images to 256×256 pixels. Each segmentation

mask was then applied to its corresponding resized image, resulting in the creation of a new, *masked* dataset. This dataset stands out as a robust and challenging benchmark due to its inclusion of expert-validated lung segmentation masks and the presence of four distinct categories, offering a rare opportunity to explore underrepresented aspects in prior studies. These samples were divided into three sets: 15,000 samples for the training set, 3,000 samples for the test set, and the remaining samples for the validation set. To balance the dataset and enhance model performance, we applied data augmentation only on the training set by increasing the number of samples for each class, as explained in sections 4.5.1 and 4.5.2.

The augmentation involved random transformations such as horizontal and vertical flips, zooming (30% to 80%), and 90-degree rotations.

**Models** For training *teacher-student* models, we used two main architecture families: VGG19 [168] as the *teacher*, and MobileNet [169] as the *student*.

**VGG19:** The convolutional architecture of VGG19 consists of a total of 16 convolutional layers, structured in a series of blocks. Each block contains multiple convolutional layers with 3x3 filters, followed by ReLU activation functions to introduce non-linearity. After each set of convolutional layers, a max-pooling layer is applied to reduce the spatial dimensions of the feature maps. This pattern is repeated across the blocks, with the number of filters increasing as the network goes deeper. Only the convolutional part of the network was retained. Following the convolutional layers, the architecture includes a batch normalization layer, a Global Average Pooling (GAP) layer, and a classification layer. The model's weights, with the exception of the classification layer, are initialized using a pre-trained model on ImageNet.

**MobileNet:** The convolutional architecture of MobileNet consists of a series of depthwise separable convolutional layers. In total, MobileNet has 28 layers that include both standard and depthwise separable convolutions. The key innovation in MobileNet is the use of depthwise separable convolutions, which break down a standard convolution into two separate layers: a depthwise convolution, which applies a single filter per input channel, followed by a pointwise convolution, which uses a 1x1 convolution to combine the outputs from the depthwise layer. This approach drastically reduces the number of parameters and computational cost compared to traditional convolutions, resulting in only $\approx 3M$ trainable parameters versus the $\approx 20M$ in VGG19, which is about 18.65% of the parameters. Each convolutional layer is then followed by a batch normalization layer and a ReLU activation function. As for the VGG19, after the convolutional layers, the architecture includes a batch normalization layer, a GAP layer, and the classification one. The model's weights are initialized randomly, but in a consistent manner to ensure that experiments are replicable.

**Proposed Framework** Let $P_t$ and $P_s$ denote the probability distributions obtained by applying the softmax function to the logits of the teacher and student models, respectively, scaled by a temperature $T > 0$:

$$P_j = \text{softmax}\left(\frac{\text{logits}_j}{T}\right) \tag{4.2}$$

where $\text{logits}_j$ are the raw, unnormalized outputs of the final layer of the model j. The temperature $T$ is used to smooth the softmax outputs, making it easier for the student model to learn from the teacher's knowledge. Similarly, let $f_t$ and $f_s$

denote the feature maps from the last convolutional layer of the teacher and student models, respectively.

Building on this, our proposed knowledge distillation method aligns intermediate features alongside probability distributions derived from logits in a convolutional neural network. This approach suggests that compressing and transforming information in intermediate layers can potentially lead to a more semantically rich representation [170]. This indicates that leveraging intermediate features in the knowledge distillation process could enhance explainable knowledge transfer between *teacher* and *student* models. Our framework employs the Kullback-Leibler divergence as the loss function to compare the probability distributions $P_\mathrm{t}$ and $P_\mathrm{s}$:

$$\mathscr{L}_{kd} = \left( \sum_i P_{\mathrm{t,i}} \cdot \log \left( \frac{P_{\mathrm{t,i}}}{P_{\mathrm{s,i}}} \right) \right) \cdot T^2 \tag{4.3}$$

The use of the KL divergence as a loss function to compare the teacher and student probability distributions was introduced by Hinton et al. [126]. The $T^2$ term ensures gradient scaling remains consistent despite the modified probabilities. In our experiments, we tested temperatures ranging from $T = 1$ to $T = 10$ with a step size of 1 and selected $T = 5$ as it yielded the best performance.

Additionally, another loss function aligns the outputs of the last convolutional layer of both models by applying Global Average Pooling and comparing them using cosine similarity:

$$\mathscr{L}_{fd} = 1 - \frac{\sum_i (\mathrm{GAP}(f_\mathrm{t})_i \cdot \mathrm{GAP}(f_\mathrm{s})_i)}{\sqrt{\sum_i (\mathrm{GAP}(f_\mathrm{t})_i)^2} \cdot \sqrt{\sum_i (\mathrm{GAP}(f_\mathrm{s})_i)^2}} \tag{4.4}$$

where GAP denotes the Global Average Pooling function and $f_\mathrm{t}$ and $f_\mathrm{s}$ are the feature maps of the last convolutional layer of the teacher and student models, respectively.

By aligning the outputs of the last convolutional layer, which is also utilized by CAM [132] to generate heatmaps, we ensure interpretability consistency for efficient Explainability Distillation. To make the outputs comparable, an additional convolutional layer was added to the student model to create the same number of feature maps.

Finally, the ground truth loss is computed using the Cross-Entropy function:

$$\mathscr{L}_{gt} = -\sum_i y_{\mathrm{true},i} \log(y_{\mathrm{pred},i}) \tag{4.5}$$

where $y_\mathrm{true}$ represents the true labels (as integers) and $y_\mathrm{pred}$ represents the predicted probability distribution over the classes.

The final loss function is thus defined as:

$$\mathscr{L} = \frac{\mathscr{L}_{gt} + \mathscr{L}_{kd} + \lambda \cdot \mathscr{L}_{fd}}{2 + \lambda} \tag{4.6}$$

The loss comprises three components: the ground truth loss ($\mathscr{L}_{gt}$), the knowledge distillation loss ($\mathscr{L}_{kd}$), and the feature distillation loss ($\mathscr{L}_{fd}$). $\lambda$ is employed to weigh the different losses. Specifically, $\lambda$ at the numerator assigns weight to the feature distillation loss, while the denominator $(2 + \lambda)$ normalizes the total contribution of the three loss components to 1. This ensures that the losses are weighted and no single component dominates the training process. We tested $\lambda$ values ranging from 0.1 to 5 with a step size of 0.5, with $\lambda = 2$ yielding the best performance.

Our method differs from prior approaches, such as the framework proposed by Sun et al. [164]; instead of comparing heatmaps generated by CAM for each sample, which is computationally intensive, we only compare the outputs from two layers of the models, thus reducing the computational load. This defines an *indirect* method of Explainability Distillation.

**Experiments** For all the experiments conducted, the following parameters were used: the Adam optimizer with an initial learning rate of $1 \times 10^{-4}$, a batch size of 32, and a maximum number of epochs set to 400. The learning rate was scaled by a factor of 0.8 whenever the validation loss did not improve for 10 consecutive epochs, progressively decreasing (e.g., from 1 to 0.8, then 0.64, and so on) until reaching a minimum value of $1 \times 10^{-9}$. Training was stopped early if the validation loss did not improve for 20 consecutive epochs, with the best weights observed during training restored.

Our experimental phase follows the steps reported below:

**Preliminary Augmentation Experiments** Preliminary experiments were conducted to determine the most effective data augmentation strategy. Various trials were performed, both with and without augmentation, to train the *teacher* model. The percentage of augmentation applied to each individual class was varied to assess its impact. Specifically, the augmentation percentage for each class ranged from +0% to +100%, with a step of 20%, for all classes except class 3 (Viral Pneumonia), which was augmented up to +400% due to its significantly smaller sample size compared to the other classes.

Combinations of augmentation percentages for the different classes were tested to identify the optimal configuration, ensuring a balanced and effective augmentation strategy that maximized model performance.

**Segmentation Masks Evaluation** Secondly, we conducted an experiment to evaluate whether to use the original *unmasked* dataset or to apply segmentation masks to the images to create a new *masked* dataset. This experiment aimed to assess the impact of segmentation masks on both $F_1$-Score and S-IoU. For this purpose, we trained two VGG-based models, one on each dataset. Furthermore, this experiment provided an opportunity to compare our *teacher* model directly with the state-of-the-art model trained on the same unmasked dataset [2]. It is important to note that this comparison strictly evaluates the teacher model against the state-of-the-art and does not involve any KD or XAI techniques from our proposed framework.

**Comparative Analysis of Distillation Methods** Once the *teacher* model was obtained and the dataset usage was determined, we proceeded to test various distillation methods. We compared FD with models trained from scratch, models trained with Knowledge Distillation alone (i.e., only distilling the logits), and models trained by re-implementing the method proposed by Sun et al. [164]. All models were initialized with the same weights to ensure the experiments' replicability.

Due to the substantial computational overhead required by explainability-based distillation approaches—such as CAT-KD [165], $e^2KD$ [166], or the method proposed by Alharbi et al. [134]—we refrained from including them in our direct comparisons. Implementing these methods would not only involve generating per-sample explanations via CAM [132] or GradCAM [133] or training additional modules (e.g., autoencoders), but also conducting multiple runs to ensure a statistical evaluation.

With our available hardware, each set of experiments already required more than weeks of computation. Undertaking such extensive efforts for multiple approaches would have been prohibitively time-consuming. As a result, we opted to focus on a single explainability-based method, i.e., Sun et al. [164], to contextualize our proposed approach in a limited yet feasible setting.

All the models were trained using different portions of the dataset: with the test and validation sets fixed, we replicated the experiments using 10%, 25%, 50%, and 100% of the training set. For each portion of the dataset, the models were trained using k-fold cross-validation, with k depending on the percentage of the dataset used—10, 4, 2, and 1 fold, respectively—to ensure the entire dataset was utilized. This approach was designed to maintain a constant computational cost by ensuring the same number of samples was processed across all scenarios. This was made to evaluate whether distillation-based methods offer advantages when data is scarce.

**Computational Cost Evaluation** Finally, we compared the computational cost of our Explainability Distillation method with Exp-KD, the method introduced by Sun et al. [164]. To do this, we calculated the time required to perform 30 epochs on 10% of the dataset, without applying data augmentation, using both the FD method and our implementation of the Exp-KD method [164]. All experiments were conducted under the same conditions, using a NVIDIA Tesla K80 (2014) GPU with 12 GB of SDRAM.

**Metrics**

**$F_1$-Score** Despite prior literature usually employing accuracy to evaluate the performance of their models, we also use the $F_1$-Score metric to handle the imbalance nature of the dataset.

**Explainability Metrics** To evaluate the explainability of the models post-training, we utilized CAM [132]. This method allowed us to determine whether the transfer of interpretability from the teacher model to the student model was successful. CAM exploits the feature maps from the last convolutional layer of the model, combined with the output from the classification layer, to generate heatmaps. These heatmaps are crucial for identifying the segments of an image that the model considers most significant for classifying a specific sample.

We decided to use CAM for our comparisons with other methods proposed in the literature, such as those by Sun et al. [164], and due to its low computational cost required to generate explainability.

We calculated the Intersection over Union (IoU) between the heatmaps, generated by the model using CAM, and the lung segmentation masks. This metric, called *Segmentation-based IoU* (S-IoU), measures how well a model focuses on the lung region for classification purposes, and is defined as the average IoU calculated across all instances. We use this metric because the lung area is the most critical region for the task at hand [144]. The IoU is defined as follows:

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{4.7}$$

To evaluate the distillation of explainability, and thus the difference between the heatmaps generated by the *teacher* and *student* models, we employed the *Teacher-based IoU* (T-IoU): an IoU between the two heatmaps, averaged across all samples.

While both metrics are based on IoU, their application differs: S-IoU compares the heatmaps computed using CAM with the lung segmentation masks, investigating whether the model focuses on the relevant lung areas during prediction; T-IoU compares the heatmaps generated by the student model with those produced by the teacher model, examining how similar the interpretability of the two models is, ensuring that the student model inherits the explainability characteristics of the teacher.

To apply these two metrics, it is essential that the heatmaps have binary values. Thus, the heatmaps were binarized using different thresholds (0.4, 0.6, 0.8) to determine whether a pixel is assigned a value of 0 or 1 based on whether its value falls below or above the threshold. Higher thresholds in binarization can effectively isolate the most critical regions in images, providing clearer insights into the areas deemed important by the model [171, 172]. Therefore, we decided to present results only for the 0.8 threshold, as it highlights the pixels most crucial for classification, thereby providing the most significant information.

### 4.5.2 Results and Discussions

**Teacher Evaluation** After training several models with varying levels of augmentation, the best performance for the *teacher* model was achieved with 300% augmentation for the Viral Pneumonia class and 80% for the other classes. These augmentation percentages were maintained for all subsequent experiments.

The results, in terms of $F_1$-Score and accuracy, for the two *teacher* models trained on the unmasked and masked datasets respectively, are compared with another work from the literature on the same unmasked dataset [2]. However, it should be noted that [2] does not report F1-Score values for this dataset, nor does it provide a S-IoU metric, as explainability is not considered in their approach. The comparison is presented in Table 4.4.

Although the model used is the same, our results are better. This improvement could be attributed to the use of different data augmentation techniques, to the different random splits of the dataset, or to the different layers following the convolutional part. Several studies [173–175] have utilized subsets of the dataset, focusing on different class combinations. The first two studies employ three classes, omitting the Lung Opacity class, while the third study uses two classes, comprising 822 samples for the Covid class and 1577 samples for the Normal class. These approaches yield test set accuracies of 98.64%, 98.24%, and 99.96%, respectively.

TABLE 4.4: Performance comparison between our teacher models trained on unmasked and masked datasets and the state-of-the-art work [2] on the same unmasked dataset. Metrics such as $F_1$-Score, Accuracy, and S-IoU are reported for our models, while [2] does not provide $F_1$-Score or S-IoU values due to the lack of focus on explainability in their approach. S-IoU results highlight the focus areas of the models, emphasizing the importance of explainability in healthcare tasks.

| | **Ours** | **Ours** | **Brima et al** [2] |
|---|---|---|---|
| **Dataset** | Unmasked | Masked | Unmasked |
| **Model** | VGG19 | VGG19 | VGG19 |
| **Accuracy** | **96.35%** | 93.29% | 93.99% |
| **F1-Score** | **96.34%** | 93.27% | Not reported |
| **S-IoU (0.8)** | 26.66% | **71.05%** | - |

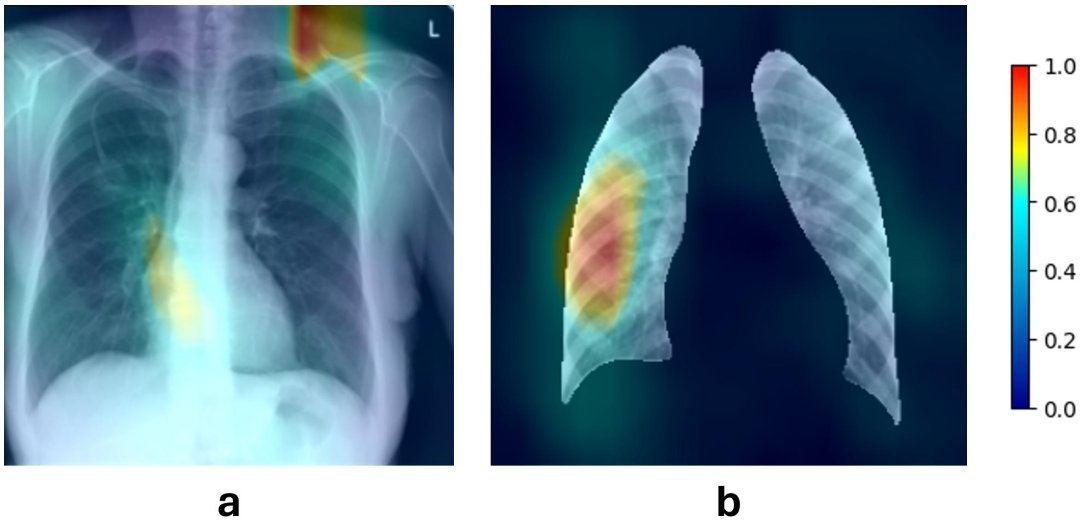


FIGURE 4.10: Heatmap examples of the *teacher* models trained on the unmasked dataset (a) and on the masked dataset (b) extracted with CAM. The color scale ranges from 0 to 1, where 1 indicates the regions of highest importance for the model's classification, and 0 represents areas considered irrelevant. The model trained on the unmasked dataset (a) shows focus on non-lung regions, while the model trained on the masked dataset (b) demonstrates a stronger alignment with the lung areas, resulting in higher S-IoU.

Table 4.4 also shows the results in terms of S-IoU for the two *teacher* models. Although the model trained on the unmasked dataset achieved state-of-the-art results in terms of $F_1$-Score on the entire dataset, the S-IoU results indicate minimal overlap with the lung areas, which are considered more relevant for the task at hand. This means the model tends to focus on non-lung regions of the image for classification, as illustrated in Fig. 4.10a. Fig. 4.10b instead illustrates the area of focus for the model with higher S-IoU. Therefore, considering the importance of explainability in healthcare [176], we decided to proceed with the masked dataset for all the following experiments. The final results for the model were obtained from the average of a 5-fold cross-validation: 68.95% of S-IoU, 92.13% of Accuracy and 92.11% of $F_1$-Score on the test set. Despite the dataset's imbalance, the remarkably close values of Accuracy and $F_1$-Score highlight the model's high robustness and reliability in handling uneven class distributions.

**Distillation Results** Figure 4.11 illustrates the variations in $F_1$-Score, T-IoU and S-IoU achieved with the four methods (Scratch, KD, FD, Exp-KD [164]) across different training set sizes. Training a model with 10% of the dataset using FD results in an over 40% improvement in $F_1$-Score compared to scratch and approximately 8% improvement over vanilla KD. When compared to Exp-KD, we achieve an improvement of around 4% while maintaining a lower standard deviation (2.88% with FD vs. 7.93% with Exp-KD). In terms of S-IoU and T-IoU, there are also improvements: approximately +22% and +11% respectively compared to scratch, around +10% and +2% compared to vanilla KD, and about +4% and +1.5% compared to Exp-KD, with similar standard deviations for both metrics when compared to vanilla KD and Exp-KD. This demonstrates the validity of FD as an ED technique in scenarios with limited data availability.

The results obtained with FD remain superior to the other methods even when using 25% of the dataset, except for S-IoU when compared to Exp-KD. For larger

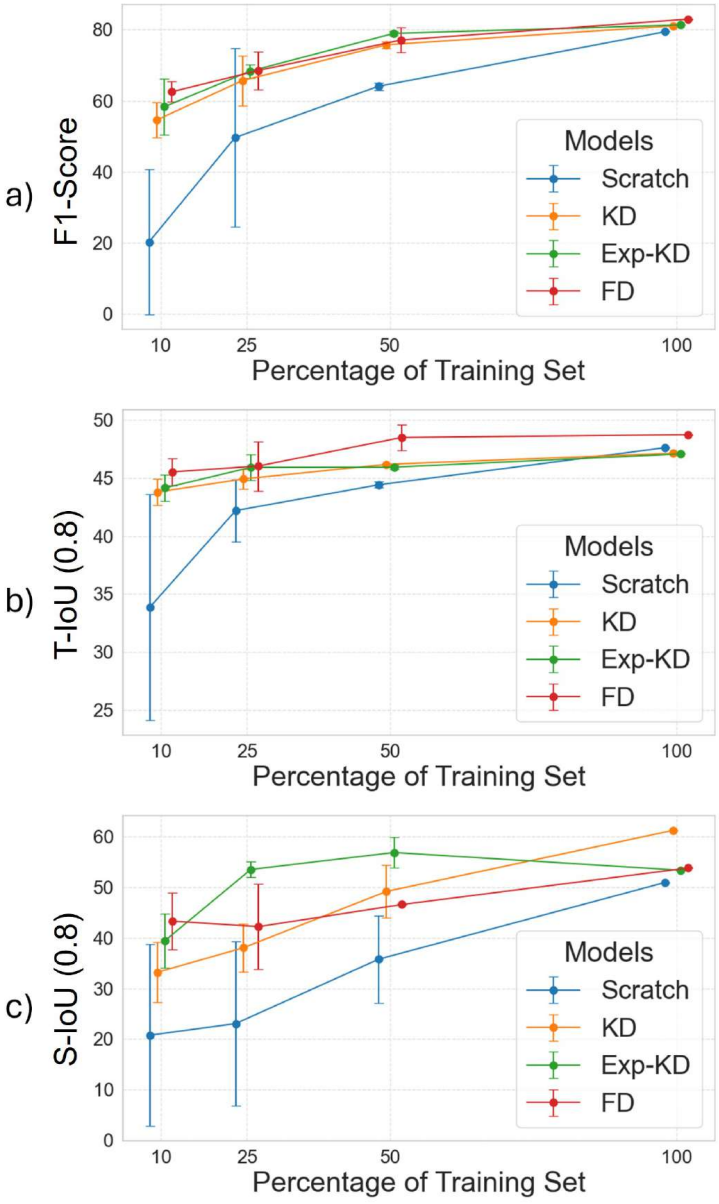


FIGURE 4.11: Variation of $F_1$-Score (a), T-IoU (b) and S-IoU (c) with Different Training Set Percentages: A Comparison of Scratch, KD, Exp-KD and FD Models

data percentages, such as 50% and 100% of the dataset, FD maintains consistent performance deltas for T-IoU relative to smaller data subsets, although no improvements in $F_1$-Score over the other methods are observed.

In general, models trained using KD outperform those trained from scratch across most metrics, highlighting the benefits of knowledge transfer. However, in terms of segmentation performance (S-IoU), our approach does not yield substantial advantages, as segmentation was not the primary objective of our training process. Instead, the segmentation analysis was performed as a supplementary evaluation to better understand the model's behavior. Our primary goal is to align the student model's explainability with that of the teacher, rather than to optimize segmentation performance. This is supported by the statistically significant improvements observed for T-IoU ($p < 0.05$), which directly reflects the model's ability to replicate the teacher's explainability patterns.

These results indicate that our FD approach more effectively mimics the teacher model compared to alternative methods. The slightly lower performance in S-IoU can be attributed to the fact that our method aligns more closely with the teacher, which itself exhibited suboptimal S-IoU in segmentation tasks. Taken together, the metrics demonstrate that FD achieves a stronger transfer of the teacher's explainability, making it particularly well-suited for applications where explainability consistency is paramount.

To further validate these findings, we conducted statistical significance tests using the Wilcoxon signed-rank test. While no statistically significant differences were observed for the $F_1$-Score or S-IoU, the T-IoU metric revealed a significant improvement ($p < 0.05$) for the FD model compared to Exp-KD [164]. This result highlights that, even when performance levels are similar across several metrics, the FD method enables a more effective transfer of the teacher model's explainability to the student model. Critically, this advantage is achieved while maintaining a substantially lower computational cost, paving the way for the use of *indirect* methods as ED techniques.

Indeed, we compared the time to complete each epoch for the four methods: Scratch, KD, FD, and Exp-KD [164]. As shown in Table 4.5, the FD method requires significantly less time than direct ED methods, while it slightly increases the computational time compared to vanilla KD. However, the difference between FD and KD is considerably smaller than the gap between FD and Exp-KD.

The computational gap can be explained by the additional steps involved in each method. FD introduces an intermediate step compared to KD, as it requires the distillation of intermediate features, which adds a moderate computational overhead. On the other hand, Exp-KD significantly increases the computational cost because it requires generating a CAM [132] for each sample and for each class during the distillation process. This additional requirement leads to a substantial increase in time complexity, making Exp-KD much slower compared to FD and KD.

TABLE 4.5: Epoch completion time for different methods.

| **Method** | **Time (Mean ± Std. Dev.) [s]** |
|---|---|
| Scratch | 30.50 ± 6.26 |
| Vanilla KD | 119.63 ± 24.72 |
| FD [Ours] | 235.97 ± 21.38 |
| Exp-KD [164] | 584.72 ± 18.50 |

# Chapter 5

# Secure system for the learning

## 5.1 State of the Art

Ensuring robust security and privacy is of paramount importance in the healthcare field, given the highly sensitive nature of patient data and the legal, ethical, and societal implications of its misuse. Consequently, medical institutions seek technological solutions that enable secure data handling while maintaining analytical capabilities. Among these solutions, Federated Learning (FL) [177] has gained substantial attention since its introduction by Google [178], particularly in clinical [179] and financial sectors. FL's decentralized paradigm ensures that sensitive data remains on local devices, exchanging only training-related information such as gradients, thereby minimizing direct exposure of patient details [180, 181]. This decentralized framework has proven effective in scenarios like blood glucose level monitoring [182] and disease treatment tasks ranging from electronic health records to cardiovascular imaging [179,183]. Beyond FL, Distributed Ledger Technologies (DLT) offer tamper-proof mechanisms to record and audit data exchanges, bolstering integrity and accountability, while Zero-Knowledge Proofs (ZKP) provide a way to verify computations without revealing private information. These three technologies can operate independently or, combined, can form a privacy-preserving ecosystem that supports secure data sharing, enhances collaboration among healthcare entities, and advances patient care outcomes.

### 5.1.1 Blockchain-Enabled Federated Learning for IoT-based Healthcare

Blockchain-based FL solutions for healthcare have emerged to address privacy concerns and incentivize collaboration. For instance, Gao *et al*. [184] developed a blockchain-based incentive mechanism to reward participants based on model contribution and quality. However, in some cases, architectures do not sufficiently address identity verification or the verifiability of computations. Astillo *et al*. [185] proposed a diabetes management system leveraging FL and deep learning models; although effective in privacy preservation, their framework used simulated datasets, lacking components to verify participant identities or check computation correctness. Similarly, Islam *et al*. [186] demonstrated how FL could preserve privacy when dealing with real-world clinical data but did not include modules to verify computations or authenticate participants. Another blockchain-based healthcare service was proposed by Singh *et al*. [187], wherein FL-trained model data were stored on the blockchain, but the scheme still exposed parameter updates to the network and omitted mechanisms for detailed participant identity and computation verification.

### 5.1.2 Fault Tolerance and Attack Detection in Federated Learning

A second category of works focuses on embedding fault-tolerance and attack-detection mechanisms into FL, frequently employing blockchain-based frameworks to protect the integrity of the learning process [188, 189]. Shayan *et al*. [188] proposed a decentralized peer-to-peer FL approach that leverages blockchain and cryptographic primitives for secure coordination, effectively mitigating some known attack vectors. Similarly, Chang *et al*. [189] presented a FL method for smart healthcare, where IoT devices perform learning and edge nodes maintain a blockchain. Through a gradient-verification scheme, they attempted to detect poisoning attacks. Despite these progressions, most approaches under this category lack comprehensive identity authentication features and do not offer verifiable proofs of the computations performed by each node.

### 5.1.3 Integration of FL with Zero-Knowledge and Identity Verification

Several recent works incorporate advanced cryptographic tools such as Zero-Knowledge Proofs (ZKPs) to reinforce data confidentiality, correctness, and privacy in FL. For instance, Xing *et al*. [190] proposed a ZKP-based FL scheme that verifies local computations and parameter aggregations on-chain. Almashaqbeh *et al*. [191] took this a step further by introducing anonymity and differential privacy, allowing participants to join and leave FL processes dynamically while preserving confidentiality. Meanwhile, Yang *et al*. [192] proposed an iZKP scheme for FL, guaranteeing robustness against various attacks yet omitting explicit participant identity checks.

In contrast, we have combined ZKP for computation verification with decentralized identifiers (DIDs) for FL participants, enabling both secure computation and transparent identity management. An external authority (acting as a certified entity) can further validate participant credentials, ensuring heightened trustworthiness in the learning environment. Table 5.1 summarizes these approaches and their comparative features.

TABLE 5.1: Comparison of the architecture presented in section 5.3 with the state-of-the-art solutions in terms of Identity verification, computation verification, attack resilience, and external auditor verification.

| Architecture or framework | Identity verification | Computation verification | Attack resilience | External auditor verification |
|---|---|---|---|---|
| [185] Astillo *et al.* | Not implemented | Not implemented | Low | Not Permitted |
| [186] Islam *et al.* | Not implemented | Not implemented | Low | Not Permitted |
| [187] Singh *et al.* | Not implemented | Not implemented | Low | Partially permitted |
| [188] Shayan *et al.* | Not implemented | Not implemented | Medium-high | Partially permitted |
| [189] Chang *et al.* | Not implemented | Not implemented | Medium-high | Partially permitted |
| [190] Xing *et al.* | Implemented | Implemented | High | Partially permitted |
| [191] Almashaqbeh *et al.* | Implemented | Implemented | High | Partially Permitted |
| [192] Yang *et al.* | Implemented | Implemented | High | Not Permitted |
| **Our architecture** | Implemented | Implemented | High | Permitted |

### 5.1.4 ELM for Online Tasks

An emerging line of research involves Online Learning (OL), particularly in the context of physiological signals such as BGL. These signals are heavily influenced by lifestyle variations, necessitating models that can adapt to new data in real time. Extreme Learning Machine (ELM) has proven to be a powerful candidate for OL scenarios [193] because of its speed, simplicity, and strong generalization.

ELM eschews backpropagation, making it suitable for underpowered or edge devices typical in FL scenarios. Various ELM variants (K-ELM, OS-ELM, KOS-ELM) have been applied in regression tasks, including BGL monitoring [194, 195]. For instance, [196] leveraged Vanilla ELM for diabetes onset classification, demonstrating high accuracy with minimal computational overhead. However, studies applying ELM specifically to BGL regression remain relatively sparse, underscoring the novelty of such approaches.

## 5.2 Datasets

In this section, a brief overview of the datasets employed in the subsequent analyses is provided. Each dataset will be described in terms of its origin, scope, and key characteristics. Presenting these datasets at the outset will help establish a clear context for the analyses to follow and underscore their relevance to the research questions at hand.

**Campus Bio-Medico Dataset** : It is a private dataset containing BGL level data and metadata collected from 20 patients at the University Polyclinic Campus Bio-Medico. The dataset was selected for analysis due to its size, the quantity of metadata extracted for each patient, and the high prevalence of comorbidities.

**Ohio T1DM Dataset** : As explained in 4.2, the Ohio T1DM dataset was introduced during the 2018 and 2020 Blood Glucose Level Prediction (BGLP) Challenges [139, 140]. It aggregates eight weeks of continuous glucose monitoring, insulin, physiological sensor, and self-reported life-events from twelve T1D adults (five females, seven males, aged 20–60), each using a Medtronic *Enlite*ÎM sensor, a fitness band, and Continuous Subcutaneous Insulin Infusion (CSII). Originally split into training/test sets per patient, we merged them to enable Leave-1-Patient-Out Cross Validation. Note that there are data interruptions, and two different fitness bands were used in the dataset's two releases.

**Approaches for BGL Prediction with Ohio T1DM Dataset** Deep learning strategies span BiLSTM [32], LSTM-based RNNs [45], ensembles of MLP and LSTM [46, 47], and even Transformer-based models [48], achieving varying degrees of performance on the Ohio T1DM dataset [197]. Notably, FedGlu [198] improves glycemic excursion detection by 35% with a FL approach, although it relies solely on MLPs, and GluADFL [199] employs an asynchronous FL framework, demonstrating robust performance under different communication topologies. Given the centrality of the Ohio T1DM dataset in this domain.

A summary overview of the described approaches can be observed in Table 5.2.

| Model | Approach | Based on | Computation | Dataset |
|---|---|---|---|---|
| [199] | GLuADFL | LSTM | Decentralized | Ohio T1DM |
| [198] | FedGlu | MLP | Decentralized | Ohio T1DM |
| [47](1) | Ensemble | Linear+RNN | Centralized | Ohio T1DM |
| [47](2) | Ensemble | Linear+RNN | Centralized | Ohio T1DM |
| [47](3) | Ensemble | Linear+RNN | Centralized | Ohio T1DM |
| [46] | Ensemble | MLP+RNN | Centralized | Ohio T1DM |
| [32] | Single NN | RNN | Centralized | Ohio T1DM+Private |
| [45] | Single NN | RNN | Centralized | Ohio T1DM |
| [48] | Single NN | Transformer | Centralized | Ohio T1DM |

TABLE 5.2: Summary of the proposed approach and the comparison models. The term Triple regressor refers to the simultaneous use of three sub-models specializing in three different situations (euglycemia, hypoglycemia, and hyperglycemia).

## 5.3 dRAIN: A distributed Reliable Architecture for IoT Networks

This first work is part of a broader effort aimed at designing an architecture capable of securely managing the exchange of information among connected devices. The rapidly growing IoT ecosystem generates an ever-increasing volume of data, which in turn accentuates the need to ensure features such as resilience, reliability, and traceability. Distributed Ledger Technologies (DLTs) can address these needs by providing data with robust security guarantees. However, conventional DLTs (e.g., blockchain) often struggle to scale efficiently and may incur high operational costs when applied to IoT [200].

An alternative class of DLTs based on Directed Acyclic Graphs (DAGs) offers performance advantages comparable to blockchain while mitigating several limitations that hamper blockchain's integration with IoT [201, 202]. Since 2019, the scientific community has devoted increasing attention to merging DAG-based DLTs with IoT systems. Studies have illustrated the potential benefits of this combination [203–205], ultimately identifying IOTA [206] as a particularly suitable DLT for IoT requirements due to its experimentally verified scalability [207–209].

Building upon these insights, the goal of this work is to present a comprehensive IoT architecture grounded in a DAG-based DLT that emphasizes security, reliability, traceability, and decentralization. We introduce dRAIN (a distributed Reliable Architecture for IoT Networks), in which each device self-certifies its digital identity on the DLT according to the Decentralized Identifiers (DIDs) standard. This approach not only ensures robust identity management but also facilitates communication, control, supervision, and updates for interconnected IoT devices that may not be known a priori. Starting from our previous work [210], which presents our proposed communication framework , in this work we present the complete architecture and a fully functional Proof of Concept (PoC) enabling to assess pros and cons of the proposed solution [211].

### 5.3.1 Architecture Description

In this section, the network architecture of dRAIN is described in a general way. In the presented architecture all critical operations will be performed by the secure part, while the non-secure part has the main task of acting as an interface to and from the outside world. Firstly, the structure of the network will be examined. The elements required for this approach to work will also be defined, furthermore these elements will be identified within the associated necessary technologies. Next, the roles and characteristics of the main network entities that interact in the system will be defined. Finally, the functional aspects will be analyzed, describing how to perform certain operations.

**IoT architecture with DLTs** In the IoT field, a system is composed of three main components: edge, fog and cloud. Following a DLT approach, the cloud could host the DLT itself, thus bringing all the properties of the latter such as reliability, total traceability, immutability and auditability. The fog could include the DLT nodes, which are interchangeable, distributed, and highly redundant, providing a decentralized access point to the network. In fact, there is almost no real difference in using one node over another to access the DLT network, the only one being the reliability of the individual node. Owning a reliable node solves the trust problem, however it introduces a single point of failure into the system. Therefore, it would

be wise to own a set of trusted nodes and rotate the Tangle access point from time to time. Furthermore, reputation mechanics allow one to quantify the trustworthiness of a generic node in the network. This allows the use of non-proprietary nodes by relying on a global reliability metric with some degree of confidence. The devices together represent the edge of the system. They are the source of the data that is forwarded to the nodes and subsequently to the DLT. Therefore, all the characteristics of security and reliability of the network are lost if they are compromised. It is evident that the devices represent the first element of a chain of trust that is reinforced by the characteristics of the DLT technologies. However, if the first link of the chain is broken, the chain itself is not able to guarantee anything. Therefore it is necessary for the devices to behave as a Root of Trust (RoT), sources of a reliability that can then be propagated throughout the system. This can be achieved by coupling the computational capabilities of the devices to a *trusted execution environment* (TEE) using technologies such as the *Arm TrustZone* or a *Trusted Platform Module* (TPM). A TEE ensures that all critical operations handled by the device are performed from a secure element that is not directly accessible from the outside. Once the devices have been secured, the network must provide services to enable their operations. There are four main components that are required for the system to function properly:

- *Data communication layer*: to enable traceable and customizable communication between devices and/or entities;
- *Digital identity provider*: to facilitate authentication between devices, enhance traceability and make clear the identity of a device on the network, preventing any identification bias;
- *Digital signature*: to ensure ownership and integrity of the published data, is also mandatory to achieve digital identity;
- *Distributed list of devices*: to enable tracking and classification of devices roles in the network, it acts like an access point to the network allowing devices to know who should the connect with; it also bind in an unambiguous way every device ID to its digital identity ID.

In DAG-DLTs digital identity could be easily provided by the DIDs (Decentralized Identifiers) standard by W3C (World Wide Web Consortium). The data communication layer, on the other hand, should be specifically designed on an higher level of abstraction than the transactions' ledger. In the proposed framework communication is modelled to be asynchronous. Each one of the devices possess a level-2 data channel where it can publish and read any data, much like a *topic* in MQTT. When the channel identity is uniquely bound to the device identity through DID, communication can be easily achieved. The DIDs standard states that each identity is associated with a document distributed on a DLT, which is accessible through an URL. To enhance automation and discovery between devices, the distributed list of them should also be published and constantly updated on the ledger by a trusted entity with a certain degree of authority. Thus, devices will be able to interact autonomously with dRAIN without the need to know each other beforehand without the need to engage a direct communication, as depicted in Figure 5.1.

**Network entities** Following the introduction of the proprieties owned by the elements composing the network architecture we now shift our attention to network entities, which may or may not have a corresponding physical element in the architecture. The main entities envisioned by the proposed framework are:

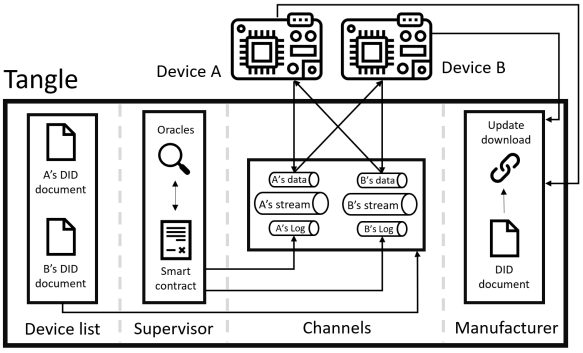


FIGURE 5.1: dRAIN Architecture

- **Edge devices**: physical devices that possess a digital identity distributed over the DLT. They operate by collecting data and publishing it through the specific data communication layer defined on the DLT. Each device has its own communication channel through which it can communicate with other devices and publish diagnostic logs or informations about itself. These are secure devices that possess cryptographic capabilities in order to manage the authentication processes related to their digital identity.

- **Supervisor**: is a decentralized entity identifiable in a smart contract or a non-static committee of devices (different from the edge devices). It is responsible for evaluating the operational state of edge devices based on rules defined at the outset, publishing its output on the ledger. It provides non-centralized information management regarding devices, thus operating a first stage of decentralized data analysis. This is very important to add value and usability to the data directly on the ledger. It possess a digital identity distributed over the DLT and can also have its own communication channel. By providing an opinion about the operational state of the devices, the supervisor indirectly introduces authority in the network. However, being a decentralized authority, it doesn't represent a single point of failure.

- **Authority of the system**: abstract entity that administers the system by managing its operating rules. It can represent the owner of the IoT service or the owner of the devices. Talking about authority may seem contradictory in a decentralized system as it seems to expose an obvious single point of failure in the system. However, in this case, the authority is not an active component in the processes, but is only responsible for setting the rules without participating in the operations. It is not an always-online component, it cannot be reached by others and it is controllable only by those who possess its private cryptographic keys, i.e. the owner of the system or the person acting on their behalf. It interacts with the system by entering instructions directly on the DLT ledger and has a DID to claim its identity. It can also have its own communication channel on the DLT;

- **HW/SW producer**: it is the manufacturer and manager of the hardware or firmware of the edge devices. In general, it does not coincide with the owner of the system authority, since the network services may not necessarily be offered by the same institution that manufactures the edge devices. It holds a DID on the DLT with which it certifies its identity unambiguously, offering services to

TABLE 5.3: Table containing the list of actors and their role as envisioned in the final architecture

| **Actors** |
|---|
| • `Devices` represent the physical devices that interface on the network. Their role is to connect to the channels and send or read data. In a single interaction a device can assume the role of author (`Authdevice`) or subscriber (`Subdevice`), similar to the MQTT protocol. In this scheme each device publishes data relevant to itself, so it will always play the role of author. Sometimes devices may be interested in data produced by others, in this case the device plays as a subscriber. The list of all devices is recorded in the Devices-list and each of them is uniquely associated to a `DID-Document`. |
| • `Supervisor`: it receives the rules from the system authority and applies them in the periodic control of the devices. It is also in charge of cutting off from the list of devices the ones that show inconsistency in their logs and of notifying it to the system `authority`. |
| • `Authority`: it is not an active component of the system and must not remain online throughout the process. Its role is to define the rules and to inform the `supervisor` of them. It is also the actor in charge of receiving alerts due to malfunctioning or tampering of distributed `devices`. |

update components in a verifiable and authentic manner. It may have its own communication channel on the DLT, but this is not a critical feature;

- **Stream Channels**: they are the channels on the DLT in which the devices post and read information, following a stream protocol, and it is identified trough the use of the `ChannelID`. They represent the basic structures that constitute the communication layer on the DLT. In order for the device to get granted the access to the channel, the channel author has to verify if the device is in the devices list and then give the `ChannelID` to the device that wants to log.

**Functional description** In this section a functional description of the architecture is given, explaining the the system operations and the interactions between the various entities of the network. The following schemes of operation are presented as possible options, in other real case implementations they could be defined in different ways and still be functional. Therefore, the purpose of this subsection is to demonstrate the feasibility of the operations and to abstractly describe the interaction algorithms developed in the proof of concept. The functions of communication between devices, supervision and management of devices and updating of devices will be analyzed. But in order to better understand them a brief description of the actors of the proposed framework is reported in table 5.3.

- **Communication**: the communication between devices is managed through an asynchronous logic of subscription to the channels of individual devices. A subscriber who wants to receive data from an author must first find the coordinates of his `DID-Document` from the list of devices on the DLT, and then proceed to send a subscription request. The author will then verify the identity of the subscriber and will only accept the request if the device is on the list.

After the subscription, the subscriber will be able to receive data directly from the author's channel, until the possible revocation of the reading rights. The procedures of subscribing and exchanging messages is represented in Figure 5.2.

- **Supervision and managment**: supervision and management of the devices is achieved thanks to the `supervisor` and authority entities. In a first phase the authority will configure the supervision rules adopted by the `supervisor` and will provide the devices with the permissions to use the system (e.g. special tokens). The device, authorized by the authority, will ask the `supervisor` to be added to the device list, consuming its special permission. Once in the system, the device will publish diagnostic logs on its channel that can be read by the `supervisor`. The `supervisor` will proceed to analyze the logs, confirming the good operational status of the device. If the logs are absent or invalid (according to the rules set by the authority), the `supervisor` will disapprove the operational status of the device, possibly sending an alert to the authority owner. The procedures of device initialization and supervision is represented in the diagram depicted in Figure 5.3.

- **Update**: regarding the updates, the firmware manufacturer is expected to possess and manage its own DID, providing an endpoint for certified and secure download of the update. This `DID-Document` provides also information on the latest firmware version. This information can be as simple as an alphanumeric code, or as complex as the hexadecimal output generated by hashing firmware binary itself. By occasionally checking this `DID-Document` and comparing the firmware version indicated with the one owned, devices will understand if they are running an outdated version. At this point it is sufficient to use the download link from the manufacturer's `DID-Document` to be sure that downloading is achieved directly from the official and reliable source. A secure *end-to-end* protocol like TLS can be used to ensure privacy and security for the communication, as the provided link could direct to an *off-ledger* source. Once the file has been downloaded, signature verification will be performed with the public key present on the manufacturer's DID, installation will follow if the signature is valid. Should the device be compromised and willingly not updating its firmware, the `supervisor` will notice this by analyzing the device logs. Once a certain time window has passed, a log with an outdated firmware version will be considered invalid, with all the consequences described in the management and supervision function. The supervision rules must hence be updated by the authority everytime a new FW package is released. The procedure of FW update is represented in Figure 5.4.

### 5.3.2 Proof of Concept development

In this section we analyse how the PoC of the architecture, described in the section 5.4.1, has been implemented. Our PoC consists of two phases: the development of a testbed for the communication and the supevision aspect and the development of a simulation of the system in order to test its scalability potential.

**Testbed** Starting with the testbed of the communication and supervision. Our implementation has been carried out using the IOTA DLT. This DLT, besides having the advantage of offering great scalability and allowing free data publication, offers

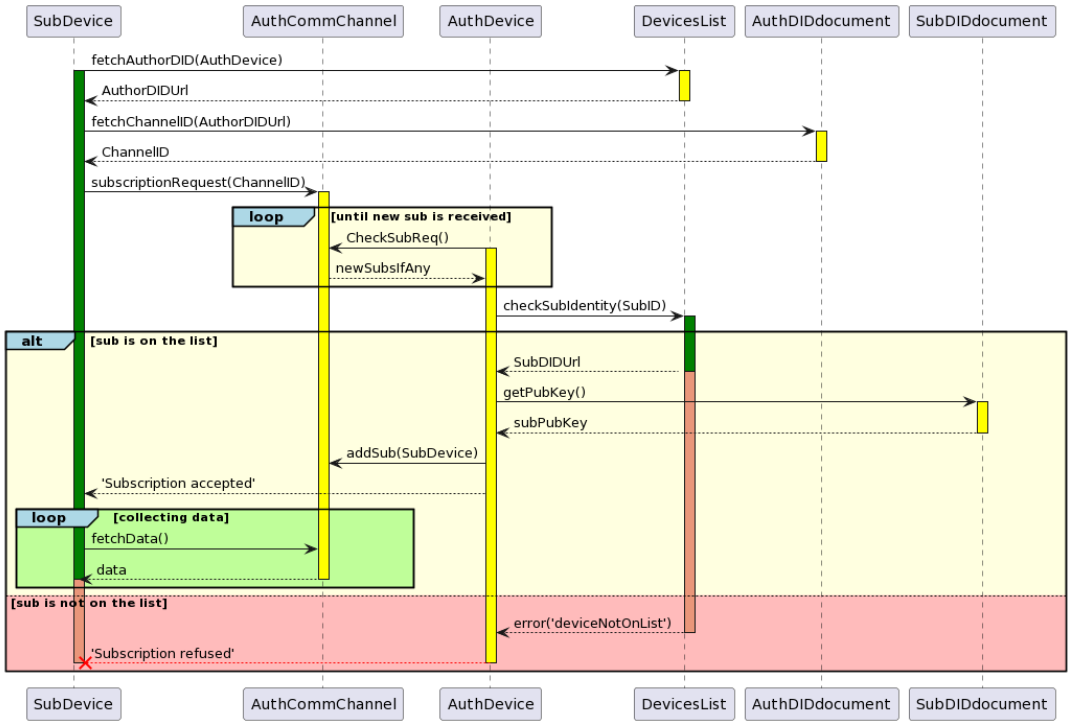


FIGURE 5.2: Communication sequence diagram

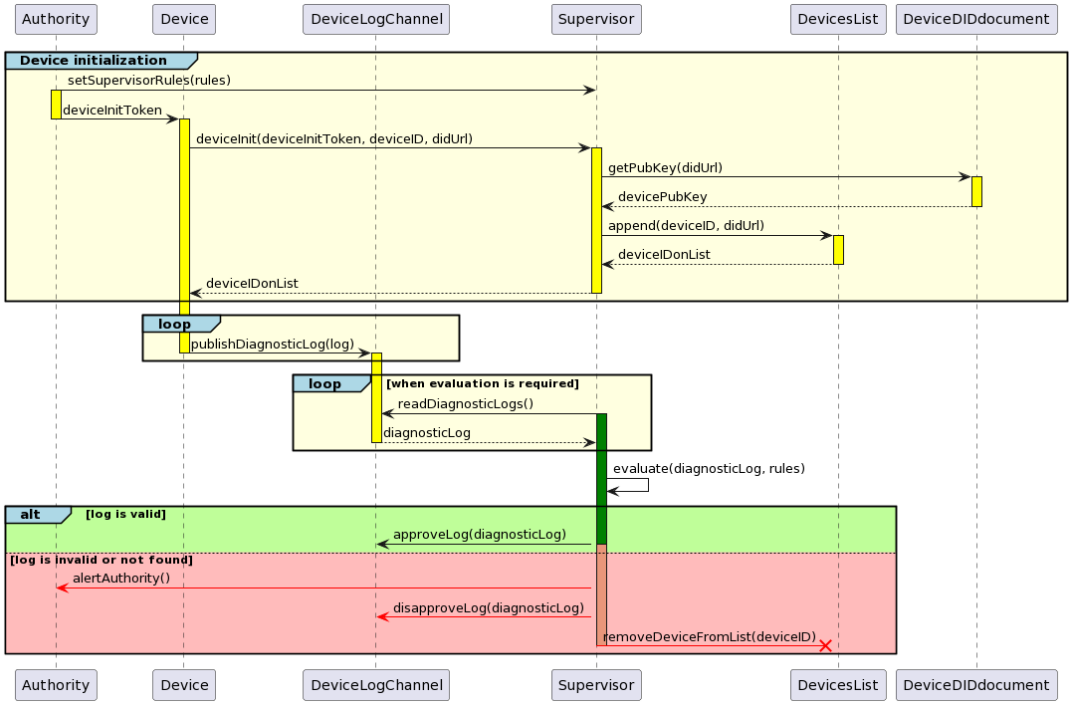


FIGURE 5.3: Supervision sequence diagram

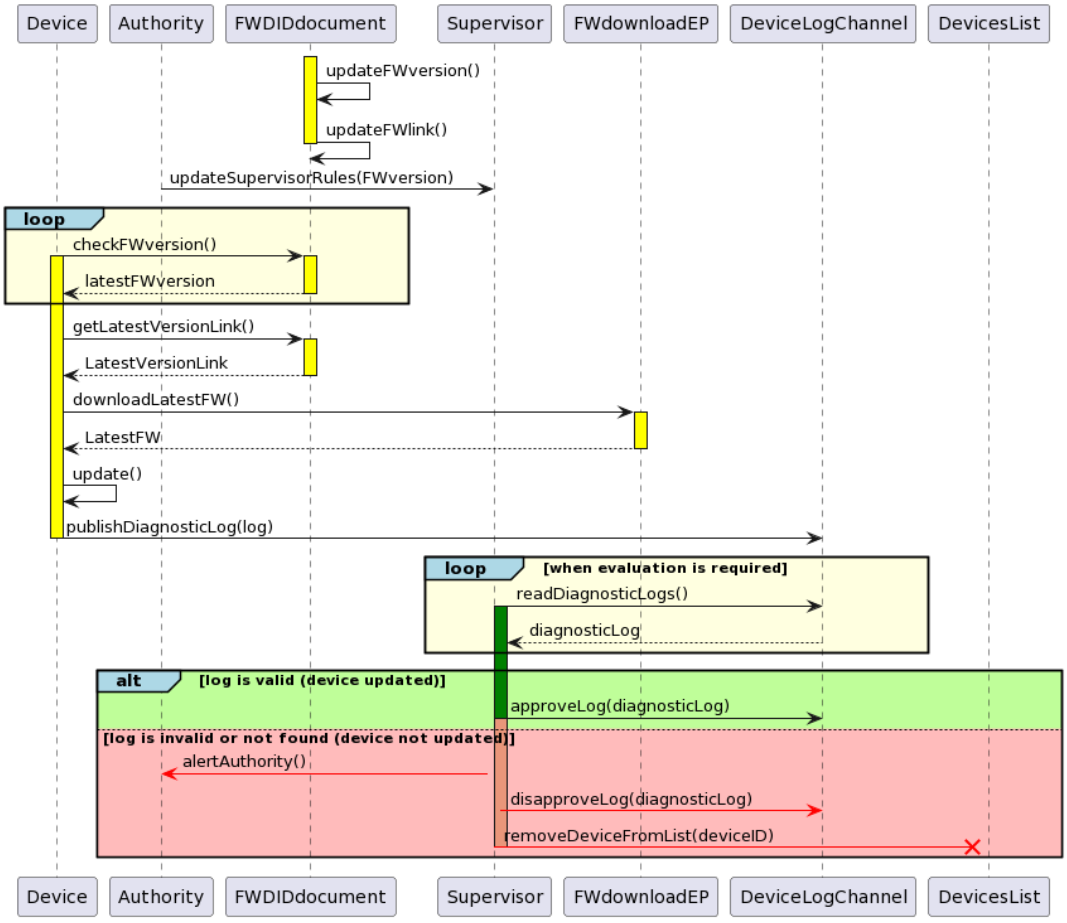


FIGURE 5.4: Update sequence diagram

the tools to implement three of the four components required by the architecture, which were defined in section 5.4.1. The **data communication layer** can be implemented through IOTA Streams, a library written in Rust that allows the creation of communication channels that follow an author/subscriber logic. The **digital signature** scheme used by IOTA is Ed25519, and DLT offers the tools to write and verify these signatures directly through its Client library, available in various languages including Rust and Go. Finally, **digital identity** via DID is made convenient through the use of the IOTA Identity library, written in Rust, which enables the creation and management of `DID-Documents` and the use of Verifiable Credentials. The PoC also includes secure hardware to represent the trusted IoT device. The ARM TrustZone is a technology that partitions the execution environment at the hardware and software level into two sections, one secure and one non-secure. Therefore it was chosen as the enabling technology for a TEE. Hence, hardware equipped with that technology was chosen and configured ad hoc. All operations exploit the IOTA network as shown in the Figure 5.5.

Furthermore, as far as the implementation of the supervision protocol is concerned, even if on the one hand the immaturity of IOTA's smart contract network, Shimmer, compromises its proper use, on the other hand, IOTA's choice to implement the EVM (Ethereum virtual machine) directly in their smart contract chain allowed us to develop a fully compatible set of smart contracts on an Ethereum private network.

The following materials were used for the development of the testbed:

- The IoT device was implemented using Discovery kit B-U585I-IOT02A from STMicroelectronics equipped with STM32U585AI microcontroller certified to PSA level-3 security level, ultra-low-power, with 32-bit ARM Cortex -M33 architecture, cryptographic accelerators and ARM TrustZone; the C++ libraries STM32Cube Expansion and Curve25519 were used for cryptographic operations;
- IOTA as a DAG-DLT together with the two software libraries written in Rust, IOTA Identity (for generating and managing DIDs according to the W3C standard) and IOTA Streams (for creating and managing communication channels on the register);
- STM32CubeIDE and STM32CubeProgrammer are the software used to configure the microcontroller;
- Remix, Ganache and Web3 are the frameworks used to develop the supervision process;
- Matlab R2021b is the software used for data analysis.

A computer equipped with AMD Ryzen 5 3600X 6-Core 3.80 GHz processor and 32 GB of RAM memory was used to simulate the subscriber device and and analyze the data.

In our testbed the B-U585I-IOT02A is the Root of Trust (RoT) of the system, as such it has been set up to behave like one. To ensure this behaviour the following steps have been taken:

1. Configuration of a Trusted Execution Environment (TEE) based on Arm TrustZone with Secure Boot Secure Firmware Update (SBSFU) framework to secure signing, communication, boot, and update;

2. Implementation of cryptographic signing and verification with Ed25519 scheme;

3. Programming of a routine to generate and sign 3 Bytes data;

An IOTA Client acts as an intermediary between the RoT and the Tangle, receiving data from the RoT and forwarding them to the DLT. The Client manages the digital identity, the Stream channel, communicates to the device list its DID URL and shares/reads data on the Tangle. In the developed PoC an author device sets up its identity and stream channel. It provides the channel ID inside the service section of the `DID-Document`. It is added to the device list and starts publishing data. In the meanwhile a subscriber device sets up its identity and is added to the device list. Then the subscriber checks the list to find the author's DID URL, which is resolved into the author's `DID-Document`. The subscriber will send a subscription request to the stream channel appointed in the service field of the author's `DID-Document`. The author then sends the request by checking the subscriber's trustworthiness inside the device list. When the subscription is accepted the subscriber starts receiving all the data published by the author on the ledger. In this PoC the author is directly associated to the HW RoT whereas the subscriber doesn't have an hardware base, it is just a process that simulates the behaviour of another device. The PoC follows the procedure defined in the sequence diagram presented in Figure 5.2 while also implementing a simplified device initialization like the one in Figure 5.3. The authority of the system and the firmware update have not been included, however the hardware has been configured to be fully compatible with a secure firmware update.

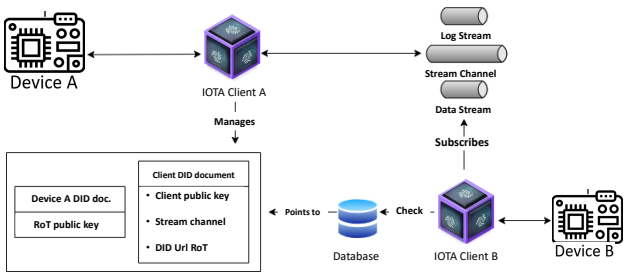


FIGURE 5.5: PoC architecture model

In order to develop the supervision protocol in a completely automated and decentralized way. We implemented an oracles committee written as a Smart Contracts system. Smart Contracts are code containers that encode and mirror real-world contractual agreements in the cybernetic realm. A fundamental premise for Smart Contracts is to represent a binding agreement between two or more parties, where each entity must fulfil its obligations under the agreement. This is ensured by automatic execution of distributed code run and verified by nodes of a Distributed Ledger network.

The actors envisioned in the supervision process of the PoC are:

- **IoT Device**: its role is to publish on the Tangle the data from the sensors connected to it.

- **Device Owner**: it performs the task of subscribing his devices to the supervision system.

- **Oracles Committee**: it guarantees the bidirectional transfer of data from the stream to the Smart Contracts.

We can moreover recognize three phases throughout the supervision process:

- **Device registration**: at this stage the device owner subscribes to the device by calling a particular function. This function passes a string parameter identifying the device's DID URL. After this, the oracular commitee will receive the DID URL and through this will subscribe to the device channel.
- **Correct hash setting**: once the DID URL is acquired, the oracular committee will takes the first log published in the log stream branch, fetches the content hash (assumed correct at the time of the request to the service), and calls a specific function to add the hash.
- **Periodic check of the last log published by the device**: periodically the committee of oracles wakes up by taking the current hash from the last log of the device. This hash is provided to the Smart Contract that then evaluates its correctness and return the result in the form of a boolean value (True if the hash is correct, False if the hash is wrong). This result is then be published in the stream configuration branch.

The necessary steps for the development of the supervision framework are:

1. Smart Contracts development.
2. Development of the simulation of the device operation flow.
3. Development of the simulation of the oracles committee operation flow.
4. Deploy the Smart Contract and test the interactions between the constituent parts of the PoC.

**Simulation** In order to test the scalability of our solution and understand how approval times and confidence rates varies in relation with the Tangle configuration, we developed an event-driven simulator that implements all the interactions between the devices and the DLT. In particular the simulator implements: two message generators, the swarm of nodes and the Tangle.

The simulation objects are:

- the **useful-message generator** that is the actor in charge of posting to the nodes all the useful transactions, therefore all the transactions that have to be tracked in the Tangle.
- The **spam-message generator** that is in charge of posting to the nodes transactions not related to the process of interest and aimed at simulating coexistence with third-party applications and their impact on measured performance.
- The **swarm of nodes**, consumes incoming messages in a first in first out (FIFO) order, posting them as transaction in the DAG structure. Any finite number of nodes can be instantiated.

In the end, to model the **Tangle** we decided to base our simulation on the one presented in [212]. This system elements allow to anchor transactions to each other following the Tangle construction rules and allow us to gather information about the

waiting time required to insert the transactions into the Tangle. The simulation software was developed in the Python programming language using various libraries, including Simpy to develop the simulation entities. The functions required to run the simulation are available on github[1].

All entities in the simulation have tunable parameters. For the message generators these parameters are the ones that compose the equation 5.1. Where $T_{bm}$ represents the time elapsed between the sending of two messages, $m_v$ represents the multiplicative value and $a_v$ the additive factor.

$$T_{bm} = (k \cdot m_v + a_v) \quad k \in \{1, \ldots, 4\} \tag{5.1}$$

These parameters are used to make $T_{bm}$ assume values compatible with those expected in application contexts of interest. As far as the swarm of nodes is concerned, the tunable parameters are: $N_n$ that represent the number of nodes developed, as previously mentioned, and $I_t$, the number of initial transactions to be instantiated in the DAG. As far as the DAG type structure is concerned, we have chosen to leave as input parameters ˇ , which is the transaction rate chosen in set $\{10, 50, 100\}$, and the value of *ff*, the multiplicative factor used in the computation of the random walk chosen in the set of values $\{10^{-1}, 10^{-2}, 10^{-3}\}$.

The flow of operations performed in each simulation can be summarized in three steps.

1. **Tangle initialisation:** transactions are posted on the Tangle in order to start filling the structure, so that it can achieve a more stable behaviour and avoid the effects due to the initial anchoring to the genesis transaction.

2. **Timer start:** once an $I_t$ number of transactions have passed, the simulation timer starts running. At the same time the message generators start posting transactions and the nodes in the swarm start consuming them, when available, publishing transaction in the Tangle-like structure.

3. **Operational status:** the simulation continues by iterating the steps expressed in the previous point until the simulation time runs out.

It was chosen to run the simulation for a total of $5 \cdot 10^8$ timer steps where each simulation step corresponds to a milliseconds. Simulations differ in the amount of spam versus useful messages, and the ratios used in the simulations are {100:0, 50:50, 33:66, 25:75, 20:80, 10:90} where the couples num1:num2 represents indeed the relative amount of useful messages over that of spam messages.

### 5.3.3 Results and Analysis

In order to optimally present the results obtained and their respective analysis, it was decided to stick to the same structure presented before thus dividing them into two separate subsections.

**Testbed Results** The test is performed on the IOTA Mainnet and carried out using two separate Clients, each representing a different IoT device, i.e. an Author in charge of publishing data on the Tangle and a Subscriber in charge of receiving and verifying them. Acquisitions were made on the Clients at 15-minute intervals for 7 days. The collected samples record the timing of micro-operations conducted in the

[1] `https://github.com/DIEM-AnyLAB/TANGLE_SIM`

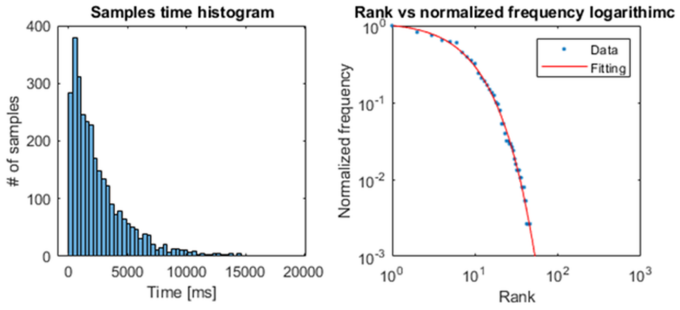


FIGURE 5.6: Histogram and empirical CDF Exponential distribution of the publication time.

TABLE 5.4: Times of the final system.

| Phase | Client | Mean time [s] | Standard deviation [s] | Standard error [s] |
|---|---|---|---|---|
| **Initialization** | **Subscriber** | **14.51** | **10.04** | **0.74** |
| **Communication setup** | **Subscriber** | **3.67** | **2.49** | **0.11** |
| **Communication** | **Subscriber** | **0.77** | **0.09** | **0.01** |
| **Initialization** | **Author** | **15.56** | **10.08** | **0.81** |
| **Communication setup** | **Author** | **5.79** | **3.58** | **0.31** |
| **Communication** | **Author** | **12.78** | **5.55** | **0.45** |

following phases: initialization, communication setup and communication. In addition, the hardware performances concerning the operation of digital signature of the message were collected. Regarding the analysis of times, the mean value, standard deviation, and standard error of the collected times were considered. The samples of the operations involving a publication on the Tangle follow an exponential distribution as visible in Figure 5.6 while the other timings have Gaussian distributions.

Moreover our analysis shown that there are no significant changes in performance over time indeed the times remain similar during all the 7 days of testbed.

Table 5.4 shows the times of partial operations, aggregated in the various phases of communication in the first column. The other columns show the values calculated for each phase to provide an estimate of the performance of the final system. It should be noted that in the test the communication consists of five messages.

Hardware performance was measured for the action of signing or verifying a message, as shown in the first column of Table 5.5.

TABLE 5.5: Singing and verification times under various software/hardware conditions.

| Action | Firmware | SW Library | Time [ms] |
|---|---|---|---|
| **Sign** | **No SBSFU** | **Curve25519** | **1218** |
| **Sign** | **No SBSFU** | **STM32Cube** | **13** |
| **Sign** | **SBSFU** | **Curve25519** | **13500** |
| **Sign** | **SBSFU** | **STM32Cube** | 144 |
| **Verify** | **No SBSFU** | **Curve25519** | **2837** |
| **Verify** | **No SBSFU** | **STM32Cube** | **27** |
| **Verify** | **SBSFU** | **Curve25519** | **31444** |
| **Verify** | **SBSFU** | **STM32Cube** | 299 |

For each action, we evaluated performance for various modes of operation, represented by the second and third columns of the table. The collected times are shown in the last column and can lead to an estimation of the execution time in a final implementation. The estimated data have been highlighted in green.

The exponential distribution of publication-related transactions is attributed to IOTA's transaction validation method, the weighted random walk. This method assigns weights to the transactions to be approved and makes a pseudo-random choice. The probability that a transaction Y, which validates a transaction X, is chosen by the algorithm is exponential and related to the equation 5.2:

$$P_{XY} = \frac{e^{-\alpha(H_X - H_Y)}}{\sum_{z \in Z} e^{-\alpha(H_X - H_z)}} \tag{5.2}$$

Where z ∈ Z which is the set of transactions that directly validate X, Hx, Hy and Hz are the cumulative weights of X, Y and Z, and $\alpha$ is the parameter that governs the importance of the weight. Once $\alpha$ is fixed, given the experimentally demonstrated almost direct proportionality of the CTPS-TPS ratio (Confirmed Transaction per Second), the validation times are stable in a certain range as the number of nodes increases. Therefore, the values collected in the test could be partly representative of those that would be found in a more populated network.

**Cost Analysis** In order to investigate the actual possibility of implementing the proposed architecture on other types of DLT, in addition to DAGs, it was necessary to analyse the implementation and operating costs related to the interaction with the decentralised network. To perform a transaction on classical blockchains like Ethereum, there are fees to pay, these fee are expressed in gas unit. A simple way to estimate execution and transaction costs is offered by the MetaMask portfolio. The formula that is automatically applied by this portfolio is expressed in the equation 5.3.

$$Q = G * (B_f + Tip) \tag{5.3}$$

TABLE 5.6: Functions costs of the Smart Contract with associated frequencies: OTfD (One Time for Device); OTO (One Time Only) and EDC (every device check).

| Function Caller | Function Name | Cost in Ether | Call Frequency |
|---|---|---|---|
| only oracle | addCorrectHash | 0.005679 | OTfD |
| only deployer | addOracle | 0.0008738 | OTO |
| device owner | deviceEnrollment | 0.00131538 | OTfD |
| only oracle | checkStatus | 0.001167 | EDC |
| only oracle | viewDevice | 0 | OTfD |
| only oracle | viewStatus | 0 | EDC |

Where $Q$ represents the cost of the operation, $G$ is the gas units that by defaults is set to 21000, $B_f$ is the base fee and $Tip$ is dependent on the transaction to be carried out and the unit of measurement associated with it is the Gwei ($10^{-9}$ Ether). The execution cost represents the associated costs to interact with the functions of the Smart Contract. The following table 5.6 show the costs in gasand fee in Ether needed to deploy the Smart Contract and those related to the call of the functions that constitute it. In fact not all functions require a gas charge for execution, just writing transactions on the Blockchain requires gas. This means that calling a contract to view data does not involve the payment of gas.

Making a conversion between the costs in Ether reported in euro you can see that a system of this type in a classic Ethereum network is not feasible. This is not mainly due to the high cost of deploying the Smart Contract but rather to those functions that must be called periodically to check the device. The defined architecture does not suffer from such a problem because can be implemented on any DLT-type structure. The high costs, however, especially with respect to recursive function calling, limit the applicability of dRAIN to DLTs that require low costs to perform operations on the ledger. This is the case of IOTA Smart Contract in which you can define a consortium network where the manufacturer of the chain can decide the introduction and the corresponding value of gas and fees. This allows the system to operate periodically keeping costs under control.

**Simulation Results** For a deeper understanding of the results obtained from the simulation, it is necessary to briefly explain how transactions are considered in the Tangle. A transaction $n$ within the Tangle can be in one of three possible states:

- **Tip:** if no other transaction has ever approved it;
- **Approved:** if at least one other transaction has chosen transaction $n$ as the transaction to be anchored to in the approval process;
- **Confirmed:** if the confirmation confidence of the transaction is not below a pre-defined level, normally set to 100 for transaction posted by an unknown author. The transaction confidence can be computed, following these steps, according to [213, 214]:

1. Run the tip selection algorithm $N$ times at a given time instant, $t$;
2. Let $i$ be one of the tips in the tangle at instant $t$, and $n_i$ the number of time in which $i$ has been selected after launching $N$ times the tips selection algorithm;
3. Given a transaction $j$, let $A_j$ be the set of transactions that directly or indirectly approve $j$, the confidence of the transaction $j$ will be:

$$Confidence_j = \sum \frac{n_i}{N} \qquad \forall i \in A_j \tag{5.4}$$

After running the simulation in 216 different configurations, we extracted for each transaction in each simulation the following data;

- **Transaction approved:** the set of transactions approved directly by the current transaction;
- **Read time:** the simulation time in which a node consumed the current transaction from the FIFO broadcast queue;
- **Tangle insertion time:** the time elapsed with the busy node in order to enter the current transaction into the Tangle, corresponding to the time required to perform the PoW and the weighted Random walk;
- **Counter of the lecture:** equals the number of reads made by the nodes up to the current transaction, differing from the transaction number by $m$. Where $m$ is the number of transactions deployed in step 1 of the simulation in the Tangle;
- **Sender:** it is undefined for all the initial $I_t$ transactions. It then assume different values if the sender of the current transaction is the useful message generator or the spam message generator;
- **Node:** it is the integer number representing the node in charge of posting the current transaction in the Tangle;
- **Confidence dictionary:** each time a new transaction is entered, the tip selection algorithm is executed $N$ times. The confidence dictionary contains for each transaction chosen during the execution of the above-mentioned algorithm the number of occurrences.

From this raw data, it was possible to calculate for each transaction the time elapsed from its input into the Tangle until the moment of approval. Furthermore, thanks to the use of the confidence dictionary and the information about the directly approved transactions, it was possible to extract information on the confidence status of each transaction already in the DAG each time a new one was entered.

We further checked whether if as the number of transactions entered into the Tangle like structure increases, the total number of tips and approval times continued to rise or reached a plateau. Our results show that the DAG-type structure achieves stability in a limited number of transactions, and that its approval times are inversely proportional to the rate at which transactions are entered into the system. This argument can be made for both tip and approval times, as can be seen from the cumulative distribution in Figure 5.7.

Furthermore, we can see that using a different number of consumer nodes or varying the *ff* of the Tangle does not produce a noticeable change in the distribution

of approval times. While for the *ff* parameter the aforementioned depends precisely on the properties of the Tangle itself, for the number of nodes this is due to the difference in speed between the publication of a new message in the FIFO queue when compared with the entry time in the Tangle of the single transaction. This is because of the limited Proof of Work time that IOTA requires for the release of a new transaction. These considerations can also be made by analyzing the table 5.7 that reports the average approval times at the end of each simulation. We can also note that the increasing presence of spam messages and thus the consequent greater number of transactions entered into the Tangle allows for faster transaction approval times.

In Figure 5.8 we show the results in terms of the total confidence level, defined as the percentage of transactions with a confidence level of 100%. The graphs report the comparison of the most different Tangle arrangement divided into six group calculated as the average of the results obtained for the simulations that have the same number of spam message generators, respectively: group 1 0% spam message, group 2 50% spam message, group 3 66% spam message, group 4 75% spam message, group 5 80% spam message, group 6 90% spam message. The simulations showed very good results with regard to the number of transactions required to pass 88% of total confidence level. A strong correlation is observed between the Tangle rate $\breve{}$ and the total confidence level, this is shown by the two opposite ends configurations: $\breve{}$ of 10 and *ff* of $10^{-1}$; $\breve{}$ of 100 and *ff* of $10^{-3}$, that reached the 88% of confidence level after the first 1000 transactions and 16000 transactions, respectively. A difference of an order of magnitude between both the number of transitions and the tangle rates is observed. Despite this, the number of transactions required remains small compared to the total number of transactions entered, bout equal to $5 \cdot 10^5$.

Furthermore, we can see in the graphs of approval time versus confidence level that the first reaches a stable value for groups 6 and 5, while the total confidence level continues to rise, albeit slowly. Thus showing a stable and scalable structure capable of improving its performance as the number of incoming messages to the network increases.

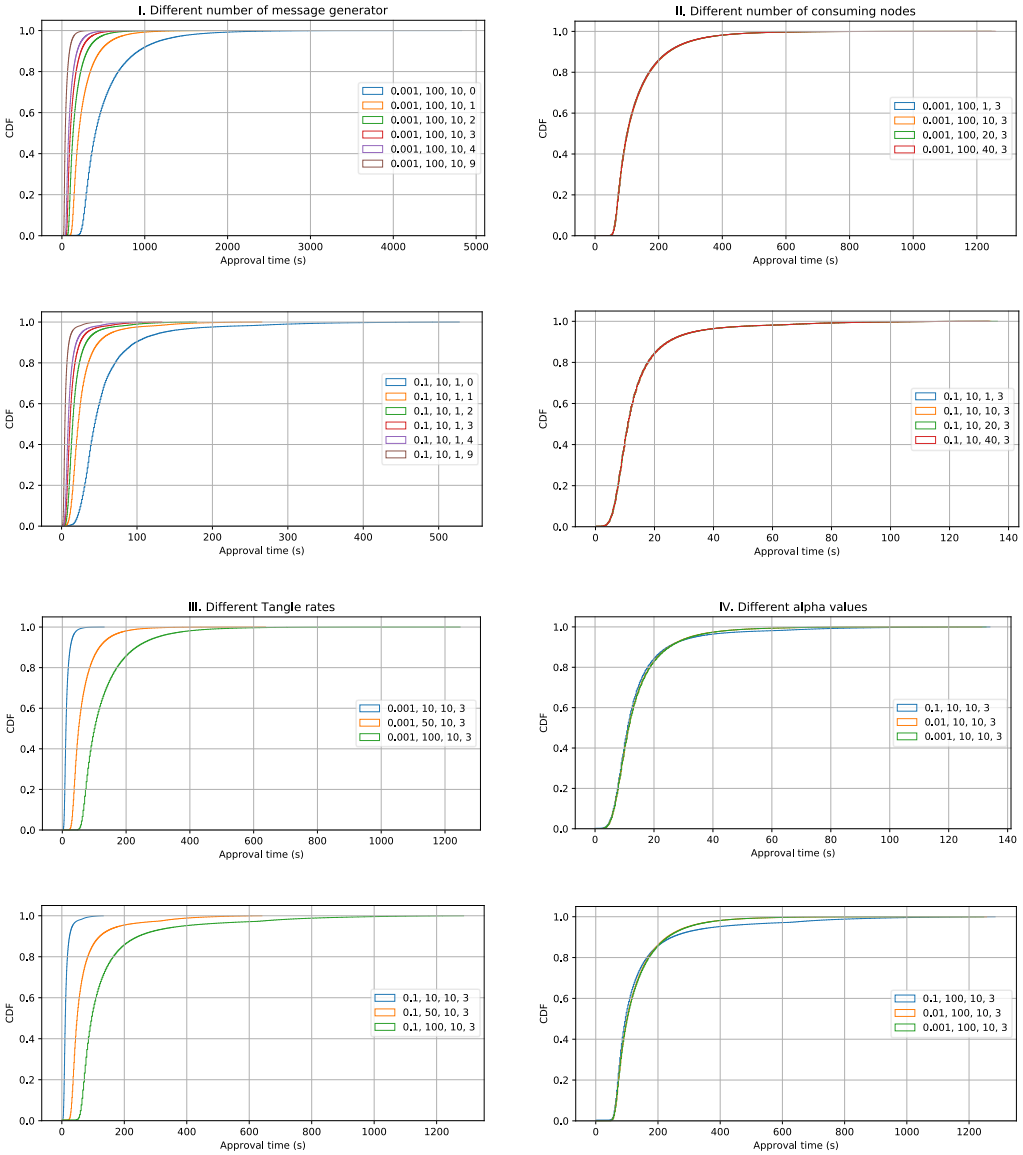


FIGURE 5.7: Cumulative Distribution Function Value (CDF) of the approval times in seconds comparing the most divergent simulation values. The simulation are divided into four sets, to show the variation in the approval times population related to: I, the number of message generators, II, the number of consuming nodes, III, the Tangle rates ˇ, IV, the *ff* values. Each simulation is identified in the legend as a Tuple like object: (*ff*, ˇ, Number of nodes, Number of spam generator.)

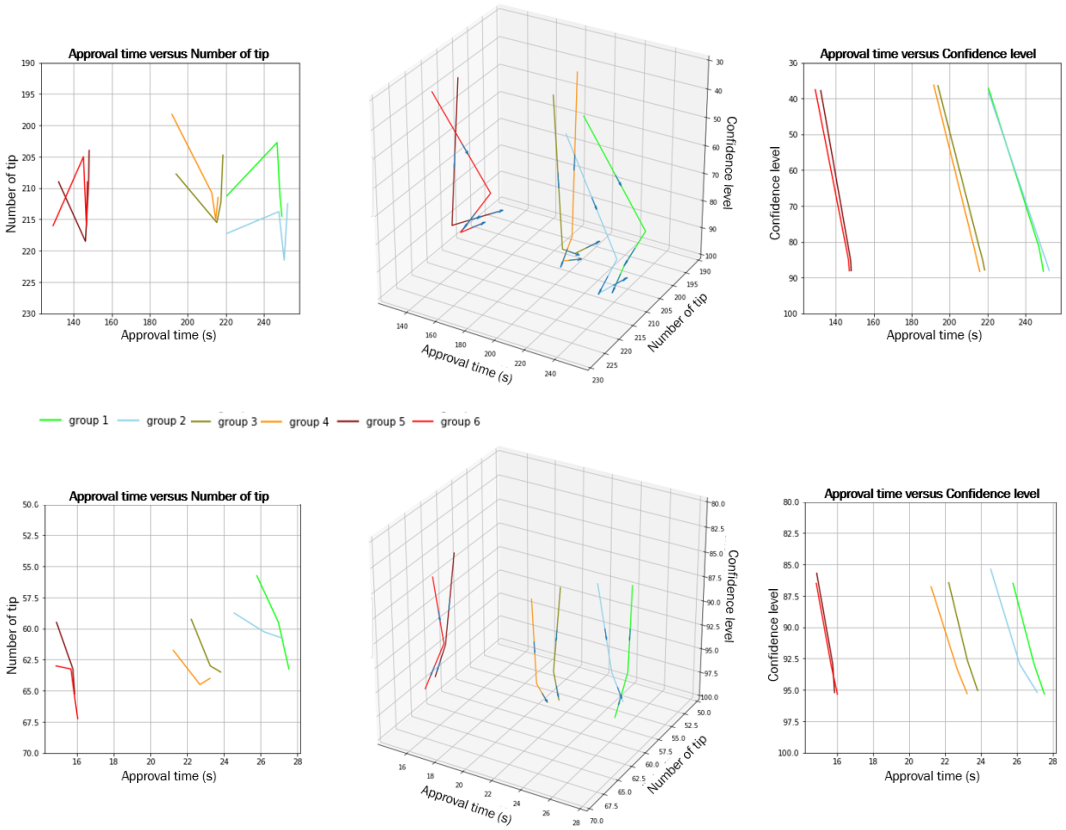


FIGURE 5.8: Comparison of the most different Tangle arrangements, ˘ of 10 and *ff* of $10^{-1}$ for the first row and ˘ of 100 and *ff* of $10^{-3}$ for the second. Three-dimensional graphs are shown with axes:average approval time, number of tips and percentage of transactions with 100% confidence). Each three-dimensional graph is associated with 2 of its projections. The graphs show the trend of the Tangle up to the confidence level above 88% for each group of simulations represented. Each group is calculated as the average of the results obtained for the simulations that have the same number of spam message generators, thus: group 1 = 0% spam message till group 6 = 90% spam messages.

TABLE 5.7: In this table are reported all the mean time of approval expressed in seconds for each of the 216 simulations at the end of them therefore after the Tangle reached stability.

| Number of Nodes | Number of spam generator | $\alpha = 0.1$ | | | $\alpha = 0.01$ | | | $\alpha = 0.001$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | rates 10 | rates 50 | rates 100 | rates 10 | rates 50 | rates 100 | rates 10 | rates 50 | rates 100 |
| 1 | 0 | 58.237 | 285.828 | 574.674 | 57.124 | 262.718 | 522.631 | 57.687 | 266.213 | 523.207 |
| | 1 | 29.135 | 143.153 | 287.055 | 28.582 | 132.158 | 261.845 | 28.714 | 132.598 | 263.707 |
| | 2 | 19.441 | 95.287 | 191.903 | 19.090 | 87.882 | 174.195 | 19.174 | 88.883 | 174.697 |
| | 3 | 14.613 | 71.732 | 143.502 | 14.325 | 66.142 | 131.077 | 14.378 | 66.576 | 131.557 |
| | 4 | 11.652 | 57.307 | 114.867 | 11.487 | 52.941 | 104.555 | 11.483 | 53.168 | 105.328 |
| | 9 | 5.844 | 28.597 | 57.543 | 5.753 | 26.417 | 52.375 | 5.759 | 26.528 | 52.649 |
| 10 | 0 | 58.275 | 286.725 | 572.453 | 57.660 | 264.941 | 524.117 | 57.416 | 267.206 | 526.046 |
| | 1 | 29.119 | 143.555 | 287.097 | 28.735 | 131.831 | 262.201 | 28.693 | 133.273 | 264.029 |
| | 2 | 19.366 | 95.440 | 191.951 | 19.114 | 88.322 | 174.64 | 19.142 | 88.637 | 175.191 |
| | 3 | 14.618 | 71.586 | 143.232 | 14.330 | 66.181 | 131.249 | 14.385 | 66.454 | 131.958 |
| | 4 | 11.688 | 57.169 | 115.131 | 11.484 | 53.000 | 104.768 | 11.498 | 53.214 | 105.508 |
| | 9 | 5.836 | 28.587 | 57.528 | 5.746 | 26.515 | 52.288 | 5.763 | 26.644 | 52.616 |
| 20 | 0 | 58.171 | 286.697 | 573.295 | 57.252 | 263.807 | 523.583 | 57.618 | 266.04 | 524.639 |
| | 1 | 29.197 | 142.918 | 287.801 | 28.71 | 132.112 | 261.78 | 28.724 | 133.094 | 262.490 |
| | 2 | 19.456 | 95.380 | 191.383 | 19.132 | 88.439 | 174.242 | 19.082 | 88.755 | 175.322 |
| | 3 | 14.548 | 71.621 | 143.674 | 14.306 | 66.464 | 130.943 | 14.365 | 66.238 | 131.611 |
| | 4 | 11.646 | 57.261 | 114.683 | 11.466 | 52.896 | 104.599 | 11.516 | 53.255 | 105.039 |
| | 9 | 5.835 | 28.495 | 57.382 | 5.742 | 26.503 | 52.320 | 5.762 | 26.576 | 52.550 |
| 40 | 0 | 58.210 | 286.452 | 574.566 | 57.197 | 264.430 | 522.621 | 57.413 | 265.613 | 526.430 |
| | 1 | 29.200 | 142.396 | 286.981 | 28.599 | 132.055 | 262.302 | 28.712 | 132.983 | 262.419 |
| | 2 | 19.395 | 95.419 | 191.992 | 19.113 | 88.015 | 174.342 | 19.094 | 88.943 | 176.605 |
| | 3 | 14.561 | 71.569 | 143.639 | 14.292 | 66.128 | 131.290 | 14.396 | 66.521 | 131.477 |
| | 4 | 11.652 | 57.169 | 114.860 | 11.467 | 53.010 | 104.896 | 11.489 | 53.189 | 105.390 |
| | 9 | 5.814 | 28.637 | 57.438 | 5.724 | 26.495 | 52.344 | 5.770 | 26.593 | 52.578 |

## 5.4 A Zero-Knowledge Proof Federated Learning on DLT for Healthcare Data

Laws and regulations related to privacy protection expect companies to apply federated training, such as the General Data Protection Regulation of the European Union (GDPR) [215], the California Consumer Privacy Act (CCPA) of the United States [216]. Unfortunately, a significant challenge persists in current FL solutions, as explained in [217, 218]. Specifically, the difficulty lies in publicly verifying all local training results without compromising the privacy of local data. As it is crucial to avoid sharing local data with all participants to safeguard data privacy, a malicious participant instead of genuinely training their model on their local data could generate local parameters (model weights) by supplying carefully crafted random numbers that are designed to push the global model in a direction that benefits them, compromising system reliability. Moreover, in order to be fully incorporated in a Healthcare landscape, an FL algorithm should not only include mechanisms to check the correct computation by participants but should also be able to provide an external *verifier* with all the information necessary for the complete verification process. Furthermore, speaking of sensitive data, it should also incorporate a process for verifying the real identities of participants. In the light of the aforementioned limitations, with the study [219] we introduce a novel architecture that integrates Zero-Knowledge Proofs (ZKP), DLT, and digital identities in alignment with the Decentralized Identifiers (DIDs) standard [220]. This architecture enables participants to self-certify their digital identities on the DLT using the DIDs framework while ensuring the verification of computations through ZKP. The proposed system includes functionalities designed to facilitate participant communication within the FL process, as well as its initialization. To demonstrate the practical applicability of this architecture, the study presents a case study focused on glucose level prediction, serving as a functional proof-of-concept. This evaluation highlights the system's ability to enhance the FL process by improving security, privacy, and auditability.

### 5.4.1 Description of general architecture

In the presented architecture, all critical operations and their outcomes will be recorded on the blockchain, allowing all participants to audit the FL process. First, the structure of the network will be examined. The elements required for this approach to work will also be defined; furthermore, these elements will be identified within the associated necessary requirements. Next, the roles and characteristics of the main entities of the network that interact in the system will be defined. Finally, the functional aspects will be analyzed, describing how to perform certain operations.

**Network entities** The main entities envisioned by the proposed framework are:

- **Peer**: physical devices that possess a digital identity distributed over the DLT. They operate by collecting personal data, executing the FL round, and communicating through the specific data communication layer defined outside the DLT, the proof of their work and the weights and biases computed to the aggregator. Each device has access to a communication channel through which it can communicate with the authority. These devices must possess cryptographic and computational capabilities to manage the authentication processes related to their digital identity and execute the federated round;

- **Aggregator**: a centralized entity in charge of proposing the FL process, through a request to the system manager. It could be implemented as a single server or as a non-static committee of devices. It is responsible for setting the rules, subscribing the peers of the FL process, and evaluating the operational state of the subscribed peer's devices based on rules defined at the outset. It possesses a digital identity distributed over the DLT and has its own secure communication channels. The Aggregator is also in charge of triggering the start and the end of each federated round and of the federated process by updating the information present in the corresponding smart contract;

- **System Manager**: an abstract entity that administers the system by managing its operating rules in a fully automated and auditable way. It has the role of deploying the set of smart contracts needed to implement the FL process required by the aggregator. Talking about authority may seem contradictory in a decentralized system as it seems to expose an obvious single point of failure in the system. However, in this case, the authority is not an active component in the processes but is only responsible for setting the rules, stored in a smart contract, without participating in the operations, except for the initial verification phase of the aggregator's request. It is not an always-online component, it cannot be reached by others, and it is controllable only by those who possess its private cryptographic keys, i.e., the owner of the system or the person acting on their behalf. It interacts with the system by entering instructions directly on the DLT ledger and transferring the ownership of the smart contracts needed for a single FL process to the aggregator that is proposing it. In general, it does not coincide with the owner of the system authority since the network services may not necessarily be offered by the same institution that is acting as the aggregator. It holds a DID on the DLT with which it certifies its identity unambiguously, offering services to update components in a verifiable and authentic manner;

- **Distributed ledger**: it is the DLT chosen to implement all verification processes. It can be either public or private; if managed by the system manager, the primary criterion for choosing the appropriate DLT is its capability to implement smart contracts and its Turing completeness, since the correct functioning of the architecture requires the implementation of Turing complete smart contract;

- **Communication Channels**: they are the channels outside the DLT in which the devices post information, following the defined protocol, and they are identified through the use of the `ChannelID`. They represent the basic structures that constitute the communication layer. In order for the device to gain access to the channel, the aggregator has to verify that the device is in the devices list and then gives the `ChannelID` to the device that wants to log.

In a real-life application scenario, the system manager assumes the role of the verifying entity, responsible for authenticating the identities of participants in the FL process. As such, it serves as a recognized certification authority by both the participants and the governing bodies of the respective state or region of operation. Complementing this, peers represent the rightful owners of the data or entities authorized to process the data on their behalf. Meanwhile, the aggregator holds a higher position compared to the peers and is entrusted with the consent of each peer to participate in the FL process. This information is added to the aggregator's

DID document. To illustrate this, consider a pharmaceutical company interested in training an AI model. The company can obtain the consent of its customers who are using a specific device to participate in FL. Once consent has been obtained, the company can request the system manager to initiate an FL process. After the system manager verifies the permissions and identities of both the pharmaceutical company and its clients, the FL process begins. From that point on, the pharmaceutical company assumes the role of an aggregator, while its clients act as peers, following the schemes explained in Section 5.4.1.

**IoT Federated architecture with DLTs** In the field of IoT, a system comprises three primary components: edge, fog, and cloud. By adopting a DLT approach, the cloud can serve as the host for the DLT itself, thus incorporating all its properties such as reliability, complete traceability, immutability, and auditability. The fog can encompass interchangeable, distributed and highly redundant DLT nodes, which offer a decentralized access point to the network. In practice, there is minimal distinction between using one node or another to access the DLT network, with the only variation being the reliability of each individual node. Although owning a reliable node resolves the trust issue, it also introduces a single point of failure in the system. Therefore, it would be wise for the aggregator to have a set of trusted nodes for all peers and periodically rotate the access point. The peers' devices together represent the edge of the system. They are the source of the data that is forwarded through the secure channels to the aggregator and to the nodes, and subsequently to the DLT.

Thanks to the implementation of a ZKP algorithm, even if a device is not executing the scripts correctly, either due to a malfunctioning or to an attack on the federated system, the aggregator is able to recognize its behavior and thus exclude it from the participants in the round or directly from the participants in the system. Once the correct computation of scripts by peers has been ensured, the network must provide services to enable their operations. There are four main components that are required for the system to function properly:

- *Data communication layer*: to enable traceable and customizable communication between peers and the aggregator and between the aggregator and the system manager;
- *Digital identity provider*: it facilitates authentication between the network entities, enhances traceability and makes clear the identity of a peer or of an aggregator on the network, preventing any identification bias; in the proposed architecture, this role is explicated by the system manager;
- *Digital signature*: it ensures ownership and integrity of the published data; it is also mandatory for achieving digital identity;
- *Distributed list of devices*: enabling tracking and classification of devices roles in the network, it acts like an access point to the network, allowing devices to know who they should connect with; it also binds in an unambiguous way every peer or aggregator ID to its digital identity ID.

Digital identity, in DLTs, could be easily provided by the W3C's (World Wide Web Consortium) DIDs standar. DIDs could be used to establish and verify the identities of the participants within the FL process. This verification could help ensure that the aggregators or peers have the necessary authority or permissions to contribute to the FL model. To draw a parallel with the HTTPS protocol's certified authority

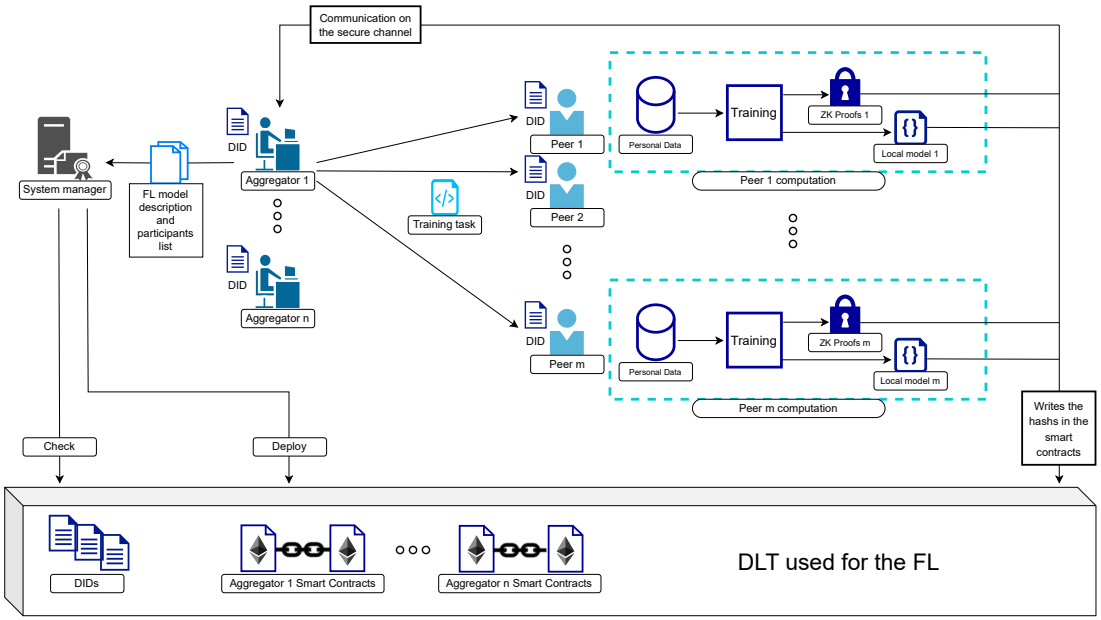


FIGURE 5.9: Architectural scheme of the proposed FL process

process, it is important to understand that the HTTPS protocol relies on digital certificates issued by trusted Certificate Authorities (CAs) to verify the authenticity of websites. These certificates help establish trust and encryption between the server and the client. In a similar vein, within an FL context, a trusted identity framework that includes DIDs could be used to establish the authority and authenticity of aggregators or peers.

The specifics of how the verification process would operate depend on the implementation and design choices made by the system manager. The data communication layer should be specifically designed at a higher level of abstraction than the transactions' ledger, for instance, external to the DLT. Meanwhile, the DLT, even if not directly involved in managing data traffic, serves as a critical verification point for the information exchanged, thereby ensuring their verifiability and clear attribution.

**Functional description** In this section, we provide a functional overview of the architecture, depicted in Figure 5.9, elucidating the system operations and the interactions among the different network entities. Subsequent operational schemes are presented as potential alternatives; in real-world implementations, they may be defined differently while remaining functional. Thus, the aim of this subsection is to showcase the practicability of the operations and abstractly delineate the interaction algorithms. We will examine the FL process initialization as well as communication functions between peers and the aggregator.

**FL process initialization** The initialization of the FL process starts with the communication between the system manager and a certified entity that acts as an aggregator for the federated process. The process is depicted in figure 5.10, and follows these steps:

- **Communication set up:** In this phase, the entity that wants to act as an aggregator informs the system manager of its intention by providing the information in its DID;

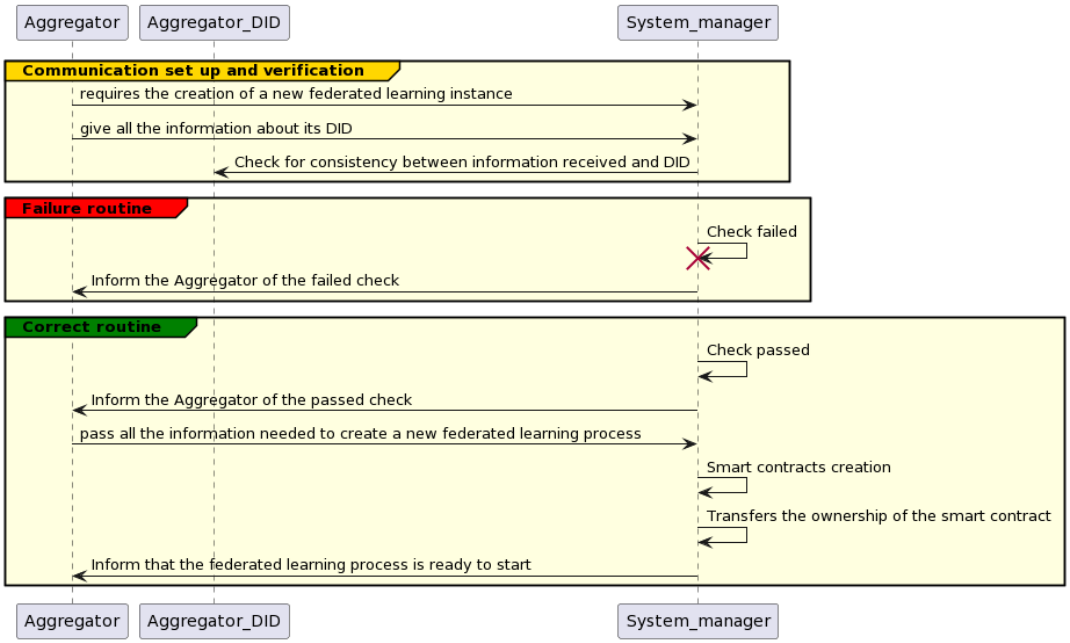


FIGURE 5.10: FL process initialization sequence diagram.

- **Verification:** The system manager checks that the information in the DID is correct and that the applicant meets all the requirements to apply for the initialization of an FL process, e.g., whether he/she is a certified physician, a healthcare facility, or a certified pharmaceutical/healthcare company. If the conditions are met, the system manager requests the aggregator to provide it with all the necessary information for the initialization of the FL process;
- **Formal communication:** The aggregator provides the system manager with information: concerning the AI model to be trained, the addresses of all peers that will be able to participate in the FL process, the minimum number of peers that will be required to pass a round of FL, the maximum duration of the round itself, and the maximum number of rounds that can be reached;
- **Contract creation:** The system manager is in charge of generating all the smart contracts necessary to verify the actions to be carried out during the FL process and assigns the peer roles to the addresses indicated by the aggregator;
- **Transfer ownership:** The system manager transfers the ownership of all the smart contracts created to the verified aggregator. After this action, the aggregator will be the only centralized authority of the system.

**Communication between a peer and the aggregator** The communication between the peers and the aggregator, depicted in figure 5.11, follows these steps:

- **Setup phase:** the aggregator and the peer reach an agreement on the AI model parameters, including the number of hidden units, size of inputs and outputs, and so on. This can be done by writing into a smart contract the hash of the complete summary of the AI model to be trained. This ensures a shared understanding of the model architecture and its specifications.

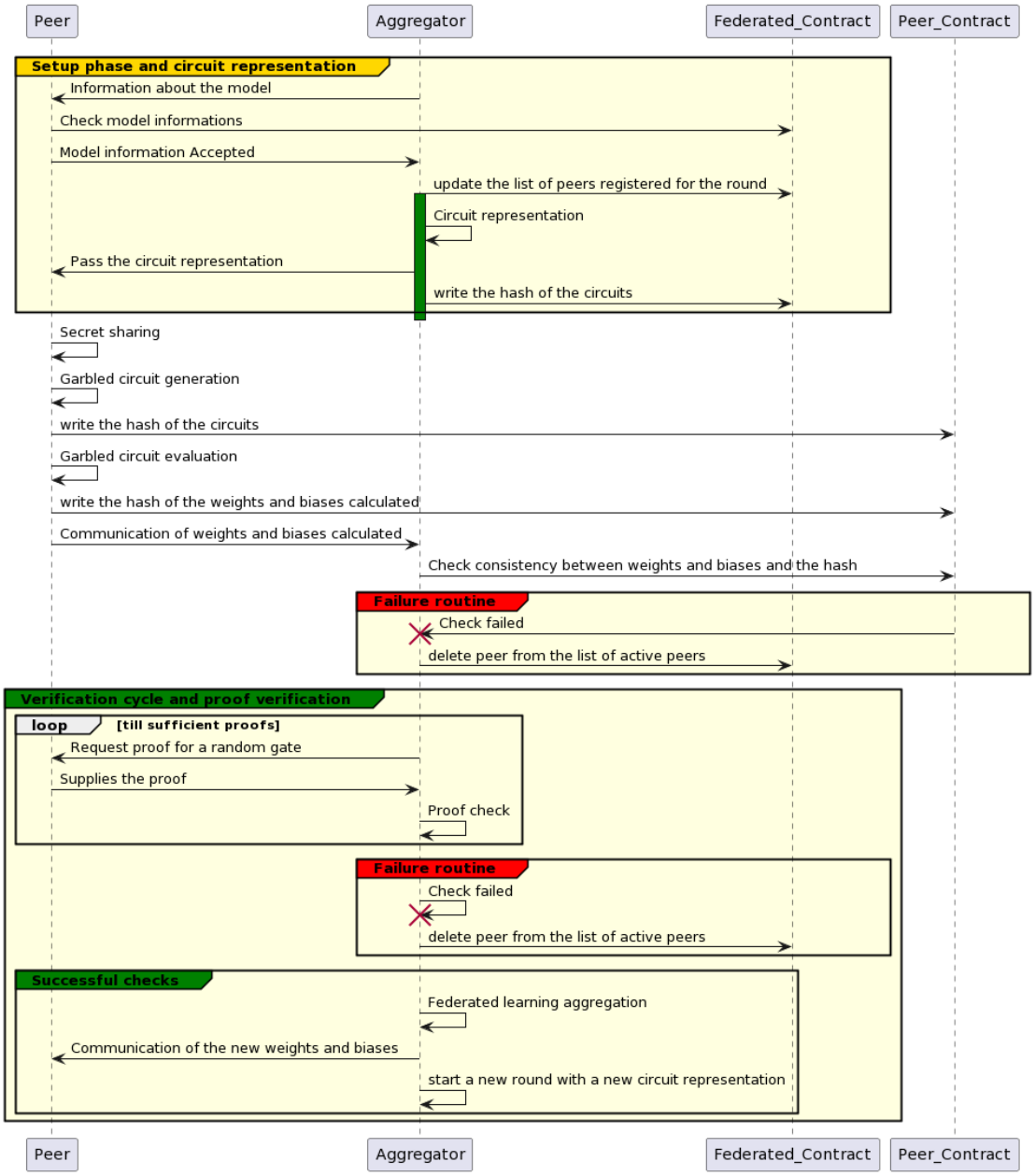


FIGURE 5.11: Communication between a peer and the aggregator sequence diagram

- **Circuit representation**: The AI computation is transformed into an arithmetic circuit, representing each AI operation such as matrix multiplications, element-wise operations, and activation functions as arithmetic operations. An arithmetic circuit is a mathematical abstraction used to represent and solve problems by breaking them down into simpler, binary operations (e.g., AND, OR, NOT gates). Therefore the circuit representation transforms the problem statement into a structured format, where inputs are processed through a series of gates to produce an output. This format is essential for applying cryptographic techniques of the garbled circuits, as it standardizes the problem into a form that can be securely manipulated and evaluated. The circuit should faithfully capture the computations performed by the AI model.

- **Secret sharing:** The peers divide their input data into shares using a secret sharing scheme, such as Shamir's secret sharing [221, 222]. This process partitions the private inputs into multiple shares, ensuring that no subset of shares discloses any information about the original data. The shares are allocated across the peer's private input, preventing the aggregator from reconstructing the original data from any subset of shares.

- **Garbled circuit generation:** For each gate in the arithmetic circuit, the peer generates garbled circuits. Garbled circuits conceal gate operations and their inputs, allowing the peer to compute results without revealing actual values. Construction of garbled circuits employs cryptographic techniques like oblivious transfer. The garbled circuit generation is a cornerstone of secure multiparty computation and ZKPs, as it allows for the execution of computations in a manner that preserves the privacy of the inputs and ensures that no additional information is leaked beyond the correctness of the computation [223, 224].

- **Proof generation and communication:** The peer constructs a proof by evaluating the garbled circuits using their secret shares as inputs. This computation is carried out without divulging the private inputs or intermediate values to the aggregator. Additionally, the peer generates cryptographic commitments for each gate output, serving as evidence that the computation was conducted accurately. Moreover, the peer sends through its secure channel the weights and biases calculated for the federated round to the aggregator and writes the corresponding hash within its smart contract.

- **Verification cycle:** The aggregator challenges the peer by randomly selecting gates within the arithmetic circuit and requesting proofs for those gates. Subsequently, the peer provides the requested proofs and commitments for verification. The aggregator assesses the correctness of the proofs by employing the provided commitments and verifies the consistency of the peer responses with the expected results. To bolster confidence in the peer's computation, the peer can repeat the challenge phase multiple times. By selecting different gates for each challenge, the peer ensures a consistent and accurate computation of the AI model by the peer, all while keeping the private inputs undisclosed.

- **Proof verification:** The aggregator examines the correctness of the proofs and commitments provided by the peer. If all the proofs successfully pass the verification process, the aggregator can ascertain that the peer accurately computed the AI process without obtaining knowledge of the peer's inputs. The

proof verification stage establishes the integrity and privacy properties of the iZKP.

It must also be taken into account that the described steps are necessary if one chooses to use an iZKP algorithm; in the case of choosing to use a NIZK algorithm or a SNARK algorithm, the **verification cycle** may become a single **verification step**.

### 5.4.2 Case study

In this section, we analyze how the case study of our architecture has been implemented, starting from the description of the dataset and the preprocessing adopted, and then defining the models implemented and the smart contracts developed. Finally, we describe the experimental setup of our case study with the aim of showing a performance comparison, a robustness test, and a cost analysis of the implementation on the DLT of the proposed FL architecture. For this case study, we chose to analyze the medical case of blood glucose levels prediction in people with Type-1 diabetes, using a public dataset to allow easier reproducibility. This case study is motivated by the critical need for personalized and specific machine learning models that can adapt to the diverse and unique datasets found in different healthcare settings [225]. Unlike pre-trained models developed on generalized datasets, FL allows for the creation of highly tailored models that learn from a variety of distributed data sources without compromising patient privacy [226]. This approach not only enhances model performance by accommodating the specific characteristics of patient data in different healthcare environments but also ensures that sensitive information is protected. The capability of FL to provide bespoke solutions that can be more accurately applied to individual patient care and specific medical conditions underscores its importance and potential in transforming healthcare outcomes through more precise and personalized interventions.

**Dataset and preprocessing** The dataset used for this work is the Ohio T1DM, this dataset was chosen, as explained in the 5.2 section, because of its wide adoption in the literature.

**Interruptions elimination** The disconnections occurred during the data recording period concerning both the CGM sensor and the fitness band. In general, this leads to complications when training a time-series model. Moreover, previous studies [227] have shown that using an input sequence of 6 timestamps (25 minutes) produces optimal results. Therefore, to minimize complications, we considered only timestamps where the CGM values were available for a minimum of 12 consecutive timestamps (55 minutes) because shorter sequences do not provide a reliable ground truth to evaluate the effectiveness of the prediction.

The consequences of these choices led to training, validation, and test sets in which each sample is, at least, a list consisting of 6 CGM values (equivalent to 25 minutes) which correspond to a single ground truth value that, in temporal terms, is the CGM value 30 minutes later than the last value in the list. To give a clearer idea an explaining formula is given 5.5: given a timestamp $x_i$ from the original dataset, a sample $X_i$ of the training set or test set and the corresponding ground truth value $y_i$ are related as follows

$$\begin{aligned} X_i &= [x_i, x_{i+1}, \ldots, x_{i+5}] & y_i &= x_{i+11} \\ X_{i+1} &= [x_{i+1}, x_{i+2}, \ldots, x_{i+6}] & y_{i+1} &= x_{i+12} \end{aligned} \tag{5.5}$$

After that, every sample was normalized based on the highest CGM value of the training set.

**Features extraction** The dataset has 19 features that, in addition to CGM values, include self-reported features (e.g., illness, stressors, meals, etc.) and features given by other sensors (e.g., basis heart rate, steps count, basis sleep, etc.) collected with a 5-minute sampling. Drawing on commonly employed features, as depicted in [228], we opted for CGM, operative carbohydrates (CHO) from meals, and insulin on board (IOB).

Starting from the operative CHO, it has to be taken into account that Kraeger *et al*. [229] demonstrated that blood glucose levels begin to rise 15 minutes after consuming a normal meal. It reaches its peak after 60 minutes and then gradually declines, stabilizing after 3 hours from the peak. To capture this process, CHO absorption was transformed into operative CHO, starting from the raw data relative to the carbs present in the dataset. Those data are characterized by the instant the meal was eaten (which in the following explanation will coincide with instant 0) and the carbohydrate value $c$, corresponding to the meal. The operative carbs $C_{op}(t)$ at the time $t$ can be calculated with the following equation:

$$C_{op}(t) = \begin{cases} 0 & , 0 \leq t \leq 15\ [min] \\ c \cdot \frac{t-15}{45} & , 15 \leq t \leq 60\ [min] \\ c \cdot \left[1 - \left(\frac{t-60}{180}\right)\right] & , 60 \leq t \leq 240\ [min] \end{cases} \tag{5.6}$$

Therefore, given $t_{CGM}$ the time of the sample of CGM, the corresponding operative carb was calculated using equation 5.6 by considering the meals preceding the sample.

For what concerns IOB, it has to be considered that a short period is needed for insulin to start affecting blood glucose levels,followed by a usual peak after 1 hour and a gradual decrease over 6 hours. Usually, the percentage of active insulin after a bolus insulin injection can be modeled using an exponential decay curve [230]. The bolus curve decays to 0 after 6 hours, but is close to 0 after about 4 hours and, for practical reasons, has been approximated as linear in this work following equation 5.7. The dataset offers 3 features to build the IoB:

- Basal: the rate at which basal insulin is continuously infused. The basal rate begins at the specified timestamp $ts$, and it continues until another basal rate is set;
- Temp basal: a temporary basal insulin rate that supersedes the patient's normal basal rate.
- Bolus: the insulin delivered to the patient. Two kinds of bolus can be distinguished: *normal*, which starts at the value indicated by the bolus and was modeled as it decays linearly to 0 after 4 hours, and *square*, which stretch out the insulin dose over the period between $ts_{begin}$ and $ts_{end}$, was modeled as $n$ equally spaced normal bolus injections over a $ts_{end}$-$ts_{begin}$ time interval and whose values are $\frac{1}{n}$ of the original square bolus injection value, where $n$ is $(ts_{end} - ts_{begin})/5$. Therefore, given the injection time of the bolus $t_0$ and the amount of insulin $b$, the expected active insulin in the human body at time $t$ is

$$bolus(t) = b \cdot \left[1 - \left(\frac{t - t_0}{a}\right)\right] \tag{5.7}$$

Where $a = 240\,min$ considering a 4-hour time decay.

Therefore, we extracted the value of IOB at the time $t$ using the equation 5.8:

$$IOB(t) = bolus(t) + basal(t) \tag{5.8}$$

where $bolus(t)$ is given by the equation 5.7 and $basal(t)$ is the one reported in the dataset unless a particular temporary basal is given. For those times when a temporary basal is defined, the value of $basal(t)$ is replaced by it. As the CGM values, both the IOB and operative carbs values were normalized in both the training, validation, and test set by the maximum value of the training set.

**Models** In the field of BGLP, the Long Short-Term Memory (LSTM) Model has led to impressive results over recent years, and it is now viewed as the state-of-the-art model in this research field [231, 232]. Therefore, we also adopted this neural-network model. The LSTM models, reported in table 5.8 considered in this work consist of an input layer that is a sequence-input layer, which takes as input an $m \times n$ matrix of features, where $m$ is the number of features and $n = 6$ is the number of recent timestamps (i.e., the latest 25 minutes) to be input. The sequence-input layer output is fed to an LSTM layer composed of $n_h$ hidden units followed by two fully connected layers with a Rectified Linear Unit (ReLU) activation function, composed, respectively, of $\frac{1}{2}n_h$ hidden units and $\frac{1}{4}n_h$ hidden units. The output layer is a dense layer with a ReLU activation function composed of one unit that returns the predicted CGM value. Since hyperparameter tuning was not of relevance to the presented work, we adopted a set of parameters already optimized in our previous work [233], therefore $n_h$ is equal to 32.

TABLE 5.8: LSTM Model Architecture

| **Layer Type** | **Output Dimension** | **Activation** |
|---|---|---|
| Sequence-Input Layer | $m \times n$ | None |
| LSTM Layer | $n_h$ hidden units | ReLU |
| Fully Connected Layer 1 | $\frac{1}{2}n_h$ hidden units | ReLU |
| Fully Connected Layer 2 | $\frac{1}{4}n_h$ hidden units | ReLU |
| Output Layer | 1 value | ReLU |

We pursued both a univariate and three multivariate approaches. In the univariate approach, the model has CGM as the only input feature (**F1**), thus $m = 1$. The multivariate approaches use the following combinations of features:

- **F2**: CGM + operative carbs
- **F3**: CGM + IOB
- **F4**: CGM + operative carbs + IOB

Therefore a total of 4 cases for the input dataset were considered in this study.

All the models implemented are trained by minimizing the mean squared error of the validation set.

**Smart contracts** To manage the verification logic both within the FL rounds and by an external *verifier* we implemented two Non-Fungible Tokens (NFTs) written in Smart Contracts, namely the
`Fed-Aggregator-NFT` and the `Fed-Peer-NFT`. It has to be considered that to ensure that the deployed NFTs align with the latest standards, the following measures were taken. Firstly, ERC165 was implemented to enable standard interface detection. This allows for seamless integration with other contracts and systems. Moreover, the Ownable function set from OpenZeppelin was implemented to facilitate the transfer of smart contract ownership, needed in the first phase after its deployment. Thanks to the two types of NFT created, it is possible to manage all the proposed FL steps. Specifically, the `Fed-Aggregator-NFT` is composed of three transaction functions and nine call functions. The list below summarizes and explains what is the role of each function.

- `modelhash`**:** is a public function, so it can be queried by everybody, and it returns the hash of the model;
- `InfoFed`**:** is a public function, and it returns all the information given by the aggregator to the system authority in .csv format;
- `aggregator`**:** is a public function, and it returns the address of the aggregator in charge of the current FL process.
- `current-round`**:** is a public function, and it returns the current round number as a `uint256`.
- `Federated-status`**:** is a public function, and it returns the status of the federated process; 0 if it is not ended and 1 otherwise.
- `Round-n-complessive-hashs`**:** is a public function, it takes as input the number of the round that someone wants to check, and it returns all the information about the required round.
- `mint`**:** is a function that can be used only by the aggregator when it coincides with the owner of the contract and when the `Federated-status` is 0. This function takes as input the hash of the model after the last round of FL and all the necessary information about the round (like the number of participants etc.) in string format, and as a result generates and assigns the NFT of the round executed to the aggregator.
- `Federated-end`**:** is a function that can be used only by the aggregator when it coincides with the owner of the contract. This function changes the `Federated-status` to 1.
- `transferOwnership`**:** is a function inherited from the Ownable contract of OpenZeppelin and gives the possibility to the current owner to transfer the ownership of the smart contract. In the proposed architecture is only used once by the system authority after the development of the smart contract.
- `CANNOT-TRANSFER-TO-ZERO-ADDRESS`**:** is a function inherited from the Ownable contract of OpenZeppelin and blocks the possibility to transfer a token to a zero address, preventing the transfer of a token to not proprietary accounts.
- `NOT-CURRENT-OWNER`**:** is a function inherited from the Ownable contract of OpenZeppelin and gives as output "018001" if an only-owner function is called by someone other than the owner himself.

- `supportsinterfaces`: Returns true if this contract implements the interface defined by `interfaceId`.

Furthermore, for what concern the `Fed-Peer-NFT` it is composed of four transaction functions and ten call functions. The list below summarizes and explains what is the role of each function.

- `Peer-status`: is a public function, and it returns the status of the peer; 0 if it is active and 1 otherwise.
- `peer`: is a public function, and it returns the address of the peer that can use this smart contract.
- `Federated-process-address`: is a public function, and it returns the address of the smart contract of the `Fed-Aggregator-NFT` for which the peer is working.
- `last-participated-round`: is a public function, and it returns the last participated round by the `peer` number as a `uint256`.
- `mint`: is a function that can be used only by the peer when it coincides with the owner of the contract and if the `Peer-status` is 0. This function takes as input the hash of the model parameters computed and the corresponding round of FL in string format, and as a result, generates and assigns the NFT of the round executed to the peer.
- `StopPeer`: is a function that can be used only by the aggregator. This function changes the `Peer-status` to 1.
- `RestartPeer`: is a function that can be used only by the aggregator. This function changes the `Peer-status` to 0.
- `modelhash`, `InfoFed`, `aggregator`, `transferOwnership`, `CANNOT-TRANSFER -TO-ZERO-ADDRESS`, `NOT-CURRNT -OWNER`, `supportsinterfaces`: are the same functions defined in the former list.

It has to be noted that all the public functions have no active role in federated training but can be called upon by an external auditor to check the status of the system or by each participant in the process in order to synchronize with it and check its status. This does not apply to the `mint` functions in the two contracts, which are the only ones necessary for the advancement of the federated round. Additionally, the `Federated-end` function is useful for interrupting the learning process and notifying participants, while the `StopPeer` and `RestartPeer` functions are necessary for expelling or readmitting a peer in the learning process.

**Experimental setup** In order to test the functionality of the designed architecture, we chose to perform tests in order to answer four questions:

- **Q1:** Are the performance reached by models trained with the proposed architecture consistent with the performance reached using classical training approaches?
- **Q2:** How does the architecture perform varying the number of participants in the round?

- **Q3:** What are the actual implementation costs of the proposed architecture on a public DLT?
- **Q4:** What are the overheads in terms of computational cost and time required due to the introduction of a ZKP algorithm into the framework?

**Performance test** To answer **Q1** we decided to perform two tests. In the first test, we trained our models on the different sets of features with three approaches: using our FL architecture, using a classical but not privacy-preserving approach (e.g. training the models as if the data were all in one server), and using a classical privacy-preserving approach (e.g., the same model topology is trained with just single patients' data separately). Finally, we compared the results obtained in terms of Root Mean squared Error (RMSE). In the second test, we conducted a Leave one patient out (Lopo) test. In this test, we trained models on our federated architecture without including one patient in the training set. Then, we used the excluded patient only for testing. We compared the values obtained from this test with the values obtained from the same model trained only on the excluded patient. The goal was to verify the advantages of using the federated model, even if the excluded patient was not among the FL participants, compared to a training task done alone knowing only the network topology.

**Robustness test** To answer **Q2** we trained the models in our FL architecture eliminating, according to a uniform random selection different for each round, a set of peers ranging from 0 to 9 out of 12, therefore excluding up to 75% of peers repeating the experiments 10 times for each percentage of excluded peers. In these conditions, we evaluated the variation in the number of rounds required to train the model and the performance changes.

**Cost analysis** It has to be taken into account that to perform a transaction on classical blockchains like Ethereum, there are fees to pay, these fees are expressed in `gas`[2] unit. In order to answer **Q3** we have estimated the execution and transaction costs using Remix, official ethereum.org IDE. On Remix, both implementation costs and individual transaction costs can be calculated, in gas, according to the rules of the selected Ethereum Virtual Machine (EVM). We chose to run these tests on Shanghai, which is the latest EVM update released. Therefore, we implemented the two contracts described in 5.4.2 and analyzed their implementation costs and the usage costs for each function.

**Overheads analysis** In order to answer the **Q4** we chose to implement a SNARK-type ZKP algorithm, and specifically the *Groth16* algorithm introduced by Jens Groth in [235]. It represents a pairing-based scheme that relies on pairing-friendly elliptic curves, derived in the trusted setup, and the hardness of the discrete logarithm problem over them. The usage of an auxiliary elliptic curve points from the trusted setup is to prevent forged proofs. The proofs generated in this scheme are small and therefore fast to verify. In fact, the proofs $\pi$ consist only of three elliptic curve elements $\pi = ([\pi_1], [\pi_2], [\pi_3])$.

[2]Gas [234] is a unit of measurement that represents the computational effort required to execute a transaction or a smart contract on the Ethereum network. The block gas limit ensures that the network is not overwhelmed with too many transactions at once, and it is determined by the Ethereum validators and can be adjusted over time based on the network's usage and capacity. The current block gas limit can be viewed on any Ethereum block explorer or node client.

| Approach/Patients | 540 | 544 | 552 | 559 | 563 | 567 | 570 | 575 | 584 | 588 | 591 | 596 | Mean | Standard Deviation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CPP F1 | 22.06 | 17.40 | 17.24 | 19.58 | 19.26 | 20.71 | 17.65 | 24.86 | 21.79 | 19.83 | 21.44 | 18.24 | 20.00 | 2.19 |
| CNPP F1 | 21.86 | 17.40 | 16.51 | 18.99 | 18.57 | 21.53 | 16.21 | 22.37 | 21.94 | 18.63 | 21.28 | 17.42 | 19.39 | 2.19 |
| FL F1 | 22.02 | 17.24 | 16.66 | 19.53 | 18.44 | 21.47 | 16.99 | 23.16 | 21.63 | 18.44 | 21.34 | 17.28 | 19.52 | 2.20 |
| CPP F2 | 21.79 | 16.43 | 19.77 | 19.61 | 18.96 | 21.13 | 16.87 | 22.91 | 22.15 | 19.85 | 21.25 | 17.51 | 19.85 | 2.02 |
| CNPP F2 | 21.81 | 16.82 | 16.68 | 18.95 | 18.56 | 20.87 | 17.84 | 22.33 | 22.12 | 18.64 | 20.54 | 16.71 | 19.32 | 2.05 |
| FL F2 | 21.95 | 16.56 | 16.54 | 18.90 | 18.30 | 21.39 | 16.48 | 22.29 | 21.89 | 18.09 | 20.73 | 16.46 | 19.13 | 2.28 |
| CPP F3 | 21.25 | 17.16 | 16.72 | 19.41 | 24.88 | 20.57 | 17.31 | 23.36 | 21.44 | 18.10 | 22.75 | 17.85 | 20.07 | 2.60 |
| CNPP F3 | 21.65 | 17.24 | 16.33 | 18.93 | 18.25 | 20.98 | 15.88 | 21.82 | 21.69 | 17.74 | 20.68 | 17.11 | 19.02 | 2.13 |
| FL F3 | 21.98 | 17.59 | 16.54 | 19.55 | 18.38 | 21.24 | 16.51 | 23.04 | 21.38 | 18.14 | 21.02 | 17.17 | 19.38 | 2.18 |
| CPP F4 | 21.75 | 16.64 | 18.93 | 18.93 | 19.53 | 21.05 | 16.27 | 23.81 | 21.08 | 18.86 | 20.91 | 16.91 | 19.56 | 2.18 |
| CNPP F4 | 21.24 | 16.48 | 16.43 | 18.17 | 18.80 | 20.62 | 15.53 | 22.26 | 21.91 | 17.53 | 21.07 | 17.08 | 18.93 | 2.28 |
| FL F4 | 21.67 | 17.05 | 16.47 | 18.37 | 18.19 | 20.79 | 15.97 | 22.04 | 21.69 | 18.29 | 20.70 | 16.47 | 18.98 | 2.18 |

TABLE 5.9: Results of the performance tests as RMSE $[mg/dL]$ for each patient, and mean +/- standard deviation for each approach; Centralized Privacy Preserving (CPP), Centralized Non-Privacy Preserving (CNPP), Federated Learning (FL). The alphanumeric code F1 to F4 represents the subset of features used as reported in subsection 5.4.2.

### 5.4.3 Results and Analysis

In this section, we present the results of the tests carried out for our case study. Following the experimental setup described in subsection 5.4.2, we present the results in three separate subsections.

**Performance test** Table 5.9 and Table 5.10 show the results of the first test carried out to answer **Q1** expressed in subsection 5.4.2.

Table 5.9 presents the outcomes of the performance test in terms of RMSE. The rows indicate the approach employed, specifying the model type using codes described in the Performance Test section of 5.4.2, as well as the subset of features used, described in section 5.4.2. The columns correspond to individual patients for whom the predictions were made, while the last two columns represent the overall mean value across all patients and the associated standard deviation. The presented table clearly illustrates the satisfactory performance achieved by FL. Moreover, it has to be noted that each client in this study has its own dataset with different characteristics in terms of size and data distribution. As it can be observed in Table 5.9, although the centralized non privacy preserving with two features has a slightly higher mean RMSE (19.32 $mg/dL$) compared to the FL approach (19.13 $mg/dL$), its lower standard deviation (2.05 $mg/dL$ versus 2.28 $mg/dL$) suggests greater consistency in performance across different datasets. This indicates that, despite a slightly higher mean, the centralized non privacy preserving approach may offer more reliability and less variability in performance. Furthermore, to delve deeper into the obtained results, a bootstrap test was performed, yielding a p-value of 0.408. This p-value implies that we cannot reject the null hypothesis, indicating that the regressions performances are comparable, amidst normal statistical variability. Notably, FL showcases an appreciable improvement when compared to the centralized privacy-preserving approach, while experiencing a slight drop in performance compared to the centralized non-privacy-preserving approach. Furthermore, it is essential to consider the

| Approach/Patients | 540 | 544 | 552 | 559 | 563 | 567 | 570 | 575 | 584 | 588 | 591 | 596 | Mean | Standard Deviation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CPP F1 | 22.06 | 17.40 | 17.24 | 19.58 | 19.26 | 20.71 | 17.65 | 24.86 | 21.79 | 19.83 | 21.44 | 18.24 | 20.00 | 2.19 |
| FL F1 | 22.45 | 17.39 | 16.34 | 19.49 | 18.24 | 21.51 | 17.45 | 22.49 | 21.60 | 18.48 | 20.97 | 17.41 | 19.48 | 2.12 |
| CPP F2 | 21.79 | 16.43 | 19.77 | 19.61 | 18.96 | 21.13 | 16.87 | 22.91 | 22.15 | 19.85 | 21.25 | 17.51 | 19.85 | 2.02 |
| FL F2 | 22.34 | 17.07 | 16.77 | 19.08 | 18.51 | 21.79 | 18.51 | 22.59 | 22.08 | 18.30 | 21.07 | 16.59 | 19.56 | 2.19 |
| CPP F3 | 21.25 | 17.16 | 16.72 | 19.41 | 24.88 | 20.57 | 17.31 | 23.36 | 21.44 | 18.10 | 22.75 | 17.85 | 20.07 | 2.60 |
| FL F3 | 21.65 | 18.14 | 16.22 | 19.39 | 18.64 | 21.19 | 16.94 | 22.13 | 21.55 | 18.29 | 20.94 | 16.94 | 19.34 | 2.01 |
| CPP F4 | 21.75 | 16.64 | 18.93 | 18.93 | 19.53 | 21.05 | 16.27 | 23.81 | 21.08 | 18.86 | 20.91 | 16.91 | 19.56 | 2.18 |
| FL F4 | 21.99 | 18.17 | 16.83 | 19.04 | 18.41 | 21.46 | 16.65 | 22.04 | 21.85 | 18.69 | 20.51 | 16.36 | 19.33 | 2.07 |

TABLE 5.10: Results of the performance tests as RMSE ($[mg/dl]$) in the Lopo test for each patient, and mean +/- standard deviation for each approach; Centralized Privacy Preserving (CPP) and Federated Learning (FL). The alphanumeric code F1 to F4 represents the subset of features used as reported in subsection 5.4.2.

trade-off between the modest performance drop and the enhanced privacy and security offered by FL in contrast to the centralized non-privacy-preserving approach. This becomes particularly crucial when dealing with sensitive medical data. Hence, it is clear that the FL approach represents a substantial improvement in learning processes, as it successfully balances performance, privacy, and security considerations.

Table 5.10 shows the results of the Lopo test. The FL approach demonstrates a lower RMSE on all patients than the centralized privacy-preserving approach. This result highlights that the FL approach possesses superior generalization capabilities, demonstrated by better performance on previously unobserved data. These observations indicate that the FL approach is able to adapt more effectively to new scenarios or data that were not present in the training set. This suggests a greater ability to learn and extrapolate intrinsic features of the data, allowing the FL model to provide more accurate and more solid predictions on new instances. Therefore, the FL approach shows considerable potential in addressing prediction problems on new patients or scenarios, while ensuring the privacy of sensitive data.

**Robustness test** In Figure 5.12 the results of the robustness test are presented: through the three plots in the figure, it is possible to see the relationships between the number of participants in the distributed architecture, the performance achieved (in terms of RMSE) and the number of rounds required to reach convergence. In particular, the x-axis represents the number of participants, the y-axis shows the performance measures in terms of RMSE, and the z-axis reflects the number of rounds needed to reach convergence. On the side of the same figure, can be found the 2D projections of the central plot that compare, respectively, the number of rounds required to converge versus the number of participants considered and the RMSE achieved versus the number of participants considered.

The results show a decreasing trend in the number of rounds required for convergence as the number of participants decrease. In particular, there is a significant drop, by approximately 80%, in the number of rounds required during the transition from 12/12 participants to 11/12 participants. This decrease in required rounds reduces the computational load required to train the model with FL. Excluding a subset of nodes introduces a form of regularization, which prevents overfitting to the data of any specific node or subset of nodes [236]. This can lead to quicker convergence towards a solution. Furthermore, the 80% reduction indicates that there is

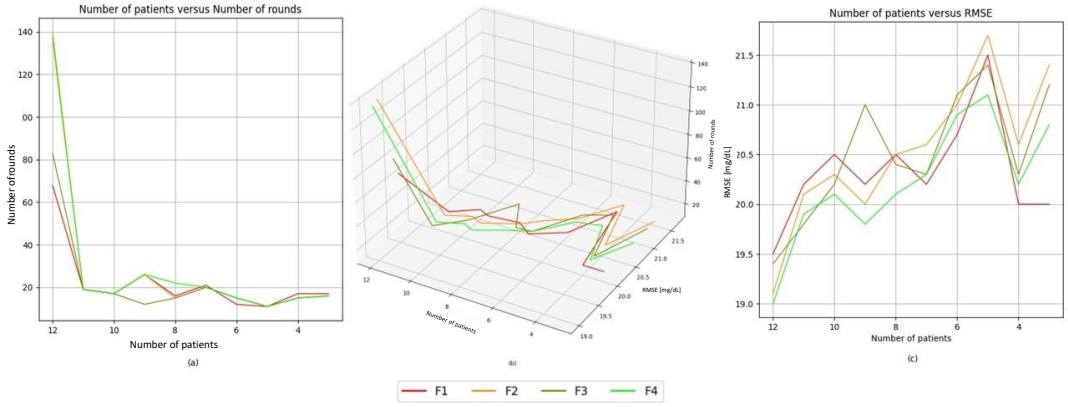


FIGURE 5.12: The graph (b) shows the relationship with the FL approach between the number of participants in the distributed architecture, the performance metric (RMSE) and the number of rounds required to reach convergence. In the graph, the x axis represents the number of participants, the y axis indicates the performance metric (RMSE), and the z axis represents the number of rounds required to achieve convergence. Plot (a) and plot (c) represent the 2D projections of graph (b) that compare, respectively, the number of rounds required to converge versus the number of patients considered and the RMSE [mg/dL] achieved versus the number of patients considered.

similarity among the data across different nodes. Therefore, by focusing on a subset of nodes, the training can zero in on more informative updates, thereby speeding up the learning process. This highlights the important role played by each participant in the distributed architecture and the effect that even a single random exclusion can have on the entire training process. This observation may suggest the appropriateness of a strategic selection or weighting of participants based on their specific computational power or data size. In addition, it shows how even a slight reduction in the number of participants can lead to considerable benefits in terms of system efficiency and response time.

For what concern the RMSE, as can be observed in 5.12, there is a slight drop in the metric as the number of excluded peers increases. The primary focus of this figure is to demonstrate the central tendency of data collected from various clients, rather than assessing the variability or uncertainty associated with these data. Moreover, the peak in RMSE at 5/12 participants, preceded by lower RMSE values at 6/12 participants and followed by a drop at 4/12 participants, may indicate that performance optimization does not follow a linear trend in relation to the number of participants, but may have turning points or 'peaks' that significantly influence the results.

These results, however, are related to the specific dataset that we are using, so in a real application scenario the aggregator if aware of particular training trends, related to the data or the task being handled can vary ad hoc the total number of participants required to advance a round allowing to manage the trade off between the drop in computational demand and the drop in model quality.

In summary, the experimental results highlight the robustness of the distributed architecture to participant exclusions, demonstrating the architecture's adaptability

to changes in data distribution and peer participation, allowing for efficient management of variations and ensuring greater operational flexibility.

**Cost analysis** Table 5.11 shows the costs in gas needed to deploy the Smart Contract and those related to the call of the functions that constitute it. In fact, not all functions require a gas charge for execution, if executed by a personal address and not by a smart contract, just the transactions that change the records on the blockchain require gas. This means that calling a contract to view data does not involve the payment of gas.

As depicted in table 5.11, it is evident that all status verification functions, when called from an account address rather than from another contract, incur zero costs. This enables unrestricted verification of information stored on the blockchain and allows external auditors to validate the accuracy of the FL process without any financial burden. On the contrary, mint functions do have associated costs, albeit limited, which increase with the number of rounds required to conclude the FL process. Nevertheless, the existence of these costs, when combined with the primary expense of deploying the contracts, encourages cautious usage of these functions. This not only mitigates the risk of spam, but also facilitates the estimation of the effective economic value of the trained model. Consequently, such a quantification justifies its potential sale and reinforces responsible behavior in utilizing the FL system. However, it should be noted that total training costs scale linearly with an increase in the number of peers and can reach considerably high overall costs. Therefore, in the event that an FL process with a considerable number of participating peers needs to be conducted, the most sustainable solution to be adopted may involve the choice of a blockchain with a low fee cost such as Tezos, Algorand, etc., or the adoption of a private or consortium blockchain network.

**Overheads analysis** Data reported in Table 5.12 are derived from a series of 10 FL simulations, each comprising 50 rounds. All experiments were conducted on the same computer, equipped with an Intel Core i9-11900K processor, 64GB of DDR4 RAM, and an NVIDIA GeForce RTX 3070 GPU, ensuring consistent and robust computational performance. The tabulated values represent the means and standard deviations of the computational times, measured in milliseconds. Our investigation delineates the comparative performance metrics between a standard FL configuration (FL-Vanilla) and an augmented variant employing SNARK implementation (FL-SNARK). Notably, the FL-Vanilla configuration exhibited an average computational latency of 21600 milliseconds, with a standard deviation of 2344 milliseconds. In stark contrast, the FL-SNARK configuration demonstrated a significant elevation in computational time, registering an average of 127108.3 milliseconds with a standard deviation of 13955.8 milliseconds for the Fit (F) phases, alongside an overhead (F-O) of 105508.3 milliseconds with a standard deviation of 11611.8 attributable to the SNARK computations. Despite the marked increase in computational overhead, it is important to highlight that such overheads are solely imposed on the peer side, with no implications for the aggregator. This design decision is underpinned by the computational efficiency of SNARK verifications using *Groth16*, which can be expediently executed within a matter of tens of milliseconds. Consequently, while the nodes are subjected to more rigorous computational demands, this mechanism serves to promote accurate and honest participation without detracting from the scalability attributes of the envisioned architecture.

TABLE 5.11: Functions costs of the Smart Contracts with associated frequencies: OTfR (One Time for Round); OTO (One Time Only) ; EC (Every Check) ; OfEC (Only for External Check) and OiN (Only if Needed). The f.c. after the cost in gas stand for "for contract" because those function are free if called by a normal account.

| Function Caller | Function Name | Cost in gas | Call Frequency |
|---|---|---|---|
| System manager | deploy `Fed-Aggregator-NFT` | 1546159 | OTO |
| Anyone | `modelhash` | 7755 f.c. | EC |
| Anyone | `InfoFed` | 10009 f.c. | EC |
| Anyone | `aggregator` | 2582 f.c. | EC |
| Anyone | `current-round` | 2521 f.c. | EC |
| Anyone | `Federated-status` | 2455 f.c. | EC |
| Anyone | `Round-n-complessive-hashs` | 12860 f.c | EC |
| Aggregator | `mint` Agg_NFT | 211386 | OTfR |
| Aggregator | `Federated-end` | 45535 | OTO |
| Owner | `transferOwnership` | 33371 | OTO |
| Anyone | `CANNOT-TRANSFER-TO-ZERO-ADDRESS` | 831 f.c. | OfEC |
| Anyone | `NOT-CURRENT-OWNER` | 808 f.c. | OfEC |
| Anyone | `supportsinterfaces` | 630 f.c. | OfEC |
| System manager | deploy `Fed-Peer-NFT` | 1615224 | OTO |
| Anyone | `Peer-status` | 2521 f.c. | EC |
| Anyone | `peer` | 2583 f.c. | EC |
| Anyone | `Federated-process-address` | 2538 f.c. | EC |
| Anyone | `last-participated-round` | 2477 f.c. | EC |
| Peer | `mint` Peer_NFT | 188457 | OTfR |
| Aggregator | `StopPeer` | 45557 | OiN |
| Aggregator | `RestartPeer` | 23634 | OiN |

| | **F** | **std** | **F-O** | **std** |
|---|---|---|---|---|
| **FL-Vanilla** | 21600 | 2344.0 | - | - |
| **FL-SNARK** | 127108.3 | 13955.8 | 105508.3 | 11611.8 |

TABLE 5.12: Results of the analysis of overheads between an FL Vanilla (**FL-Vanilla**) and FL with SNARK implementation (**FL-SNARK**). The table shows the mean values with their standard deviations (**std**) divided by Fit (**F**) phases and their associated overheads (**F-O**). All reported values are in milliseconds.

## 5.5 Federated Online Extreme Learning Machine for Blood Glucose Level Forecasting in Type 1 Diabetes

This study proposes a solution to the previously mentioned problems concerning machine learning and deep learning in the context of diabetes by introducing a system for Online Learning (OL) of specialized models within a federation dedicated to a privacy-secure learning paradigm. The framework employs a triple NN approach, where each submodel specializes in a particular glycemic condition. It leverages a decentralized and distributed paradigm known as FL, recognized for its robust data privacy management capabilities. FL enables the sharing of knowledge extracted from private data across a large network of federated devices without the need to exchange the raw data itself. Additionally, the use of Extreme Learning Machine (ELM) [237] models in the FL environment ensures excellent generalization abilities with minimal computational overhead. A variant of ELM, Online Sequential Extreme Learning Machine (OS-ELM) [238], has been further enhanced to create the Robust Online Sequential Extreme Learning Machine (ROS-ELM), particularly suitable for time series forecasting. As a result, online model training can be performed with minimal computational resources, while maintaining the privacy of sensitive data. All these components converge into a single methodology, referenced here as "FedROS-ELM," illustrated in Figure 5.13.

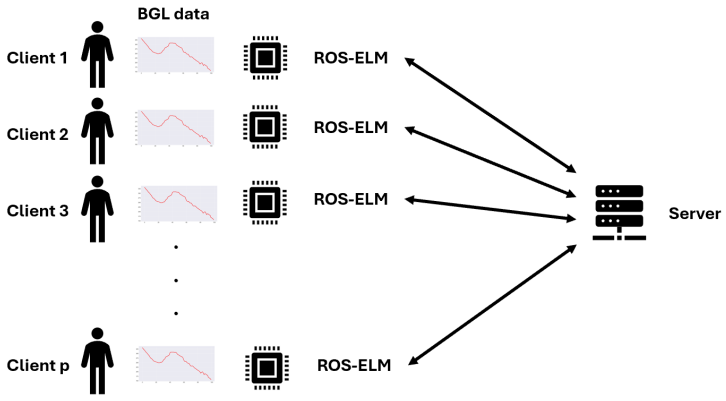


FIGURE 5.13: Representation of the FedROS-ELM framework in which each *client* represents a device connected to the individual patient for monitoring the glycemic level and training a local ROS-ELM model on it. The knowledge extracted from each local model is sent to the *server*, a centralized computing unit used for aggregating the knowledge shared within the federation.

### 5.5.1 Comparison with the literature

The comparison with the models was made in terms of tradeoff between prediction performance and computational complexity required for training. For this reason, we provisionally excluded from the selection described in the paragraph *"Approaches for BGL Prediction with Ohio T1DM Dataset"* of section 5.2 excessively complex models which, although performing well in terms of prediction error, belong to a category of prediction models that are not suitable for implementation in edge devices. These models include the Transformer and other large Deep Neural Networks architectures, which perform very well on time series forecasting tasks [239], [240],

but are too complex for our purposes. Also relatively simple models, such as linear models, have been excluded from further consideration: while they require minimal computational effort, they typically exhibit lower levels of predictive accuracy than those found in the present study. Six deep learning approaches were selected from the literature, referred to as "comparison models", each of which was originally applied to Ohio T1DM Dataset. A summary overview of the approaches used can be observed in Table 5.13. The first selected work [241], based on ensemble learning, includes three different approaches (1), (2), (3) that differ in the criteria by which the predictions of three separate models (Linear Model, Vanilla-LSTM: VLSTM and Bidirectional-LSTM: BiLSTM) are aggregated: Stacking (1), Multi-variate (2) and Subsequences (3). The combined output of the three models is given as input to a Meta-Learner, which, depending on the approach, looks like a Linear Model (1), a Multivariate-LSTM (2) and a Conv-LSTM Encoder-Decoder (3). In the case of [242] we also find an ensemble approach, but this time it is the integration of the predictions made by MLPs and LSTM networks at different levels. Again, the work gives us more than one prediction model, in fact we find either an MLP network or an LSTM network as Meta-Learners. In [32] is shown the application of a BiLSTM network to three different feature sets (uni-variate and two multi-variate), of which only the uni-variate was considered for consistency. Also in [243] we find the application of Recurrent Neural Network (RNN) to the glucose level regression task, in particular LSTM networks. In all the papers presented the prediction error values are provided directly by the authors of the works, this is not the case for the computational complexity associated with training the models. The latter must be estimated from the architecture. To this end, reference may be made to [244], in which a consistent method is provided for calculating the Floating Point Operations (FLOPs), performed by the algorithm during training. However, this aspect will be described in more detail later.

| Model | Approach | Based on | Computation | Dataset |
|---|---|---|---|---|
| Proposed | Triple NN | FedROS-ELM | Decentralized | Ohio T1DM |
| [241](1) | Ensamble | Linear+RNN | Centralized | Ohio T1DM |
| [241](2) | Ensamble | Linear+RNN | Centralized | Ohio T1DM |
| [241](3) | Ensamble | Linear+RNN | Centralized | Ohio T1DM |
| [242] | Ensamble | MLP+RNN | Centralized | Ohio T1DM |
| [32] | Single NN | RNN | Centralized | Ohio T1DM+Private |
| [243] | Single NN | RNN | Centralized | Ohio T1DM |

TABLE 5.13: Summary of the proposed approach and the comparison models. The term Triple NN refers to the simultaneous use of three sub-models specializing in three different situations (euglycemia, hypoglycemia, and hyperglycemia).

### 5.5.2 Materials

The proposed predictive system is structured in a federated fashion, comprising a *server* and several *clients*. Each *client*, which represents the physical device used to monitor the individual patient, is equipped with three ROS-ELM regressor models

that are trained on distinct subsets of the dataset. At the conclusion of each training iteration, each *client* transmits knowledge to the *server*, where it is aggregated with knowledge from all other *clients*. The information extracted from the data and aggregated is used to generate a global model containing knowledge from the entire federation, this model will then be responsible for making future predictions.

**Pre-processing** For this work we used the Ohio T1DM dataset 5.2, it was necessary to verify that the values were all temporally spaced by 5 minutes and that they were all positive real numbers ($\mathbb{R}^+$). For each sample with missing values or not spaced 5 minutes apart from the previous or next sample (with a tolerance of 10 seconds), it is labeled as "compromised". All compromised samples plus the (6 + PH) samples following the compromised were eliminated. PH is the prediction horizon, it represents how far ahead in time you want to perform the prediction and is expressed in number of samples. It was decided that a linear interpolation of the missing data would not be performed in order to avoid introducing a bias in the inference process. This would ensure that the model would not over-perform on the interpolated samples. Furthermore, no higher order interpolation was conducted as the majority of the missing or unusable data were isolated. The introduction of such a solution would have necessitated the local increase in sample density in the temporal neighbourhood of the sample to be interpolated. This, although effective in many contexts, would have deviated from the real-world application context in which sampling occurs at a fixed frequency.

The single training example has been generated as follows. A number of observation samples was chosen to be 6 (25-minute observation), i.e. the size of the input vector. This choice was motivated by two factors: firstly, this is the observation time span that maximizes the performance of our approach; secondly, a literature review shows that 6 observation samples is one of the most popular choices. Subsequently, the target sample for prediction must be selected, i.e. the one the model must learn to predict from the input. As previously stated, the time interval separating the last sample of the input vector from the prediction target is PH and this value was also chosen to be 6 samples for a prediction horizon of 30 minutes. Figure 5.14 provides a schematic representation of the generation of the single training sample. The entire dataset is transformed into training examples by treating the aforementioned process as a moving window. The step or resolution at which this window moves is variable and it was called SWstep.

### 5.5.3 Proposed Methodology

**Curriculum for expert model** With the aim of generate specialized models for different glycemic conditions, we first needed to define these conditions and find a way to differentiate them. This was achieved by defining a threshold, called $Hypo$, which separates euglycemic conditions from hypoglycemia ($Hypo = 100$ mg/dL) and a second threshold, called $Hyper$, which separates euglycemic conditions from hyperglycemia ($Hyper = 180$ mg/dL). Subsequently, $\hat{M}$, the mean value in mg/dL over the input vector, for each example was calculated. This value was then assigned to each example to identify its relevance to the areas of hypoglycemia, euglycemia and hyperglycemia. This operation permitted the training of three distinct models on three different *curriculum*: the first was designed for hypoglycemic conditions, or those that may lead to hypoglycemia, the second was designed for hyperglycemic conditions or those that may lead to hyperglycemia. Finally the third *curriculum* was designed for euglycemyc conditions. The calculation of $\hat{M}$ is precisely intended to

manage transition conditions between the areas bounded by *Hypo* and *Hyper*. Although the dataset has been divided into the sections just seen, the order in which the global model is trained and thus with which the different local models view the time series has not been altered. For each training example, the glycemic condition indicated by $\hat{M}$ of the input vector is checked, on the basis of which the most appropriate regressor to be trained is chosen.

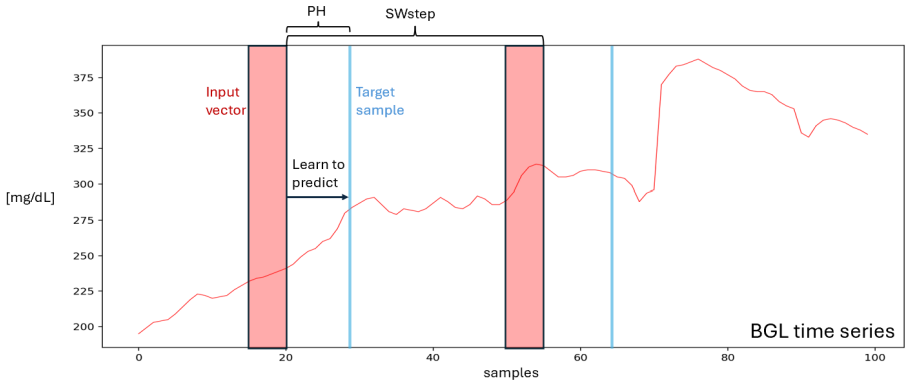


FIGURE 5.14: This image illustrates the process of generating training samples for the model. The red portion highlights the input vector formed by concatenating six consecutive samples, while the blue line identifies the target sample, which the model is tasked with predicting. 'PH' denotes the prediction horizon, expressed as the number of samples, and 'SWstep' refers to the step size by which the time window, used to generate training samples, shifts along the BGL series, also measured in sample units.

**Extreme Learning Machine and ROS-ELM** The idea behind ELM training underlies each of the local models in the proposed framework. This architecture is based on a pseudolinear model, which learns in a non-iterative manner: it simultaneously look at all the training examples given to it, on which it solves a linear system to extract knowledge. ELM is essentially a Neural Network *single hidden layer*, where the first layer performs a mapping of the input in a random feature space and has neurons that after being initialized, assigning to their weights $\mathbf{w}$ and biases $\mathbf{b}$ values sampled from a generally arbitrary distribution, are no longer updated. The second layer, on the other hand, performs predictions. An illustrative diagram of the network can be found in Fig. 5.15.
The random mapping performed by the first layer can be formalized through the $\boldsymbol{H}$ matrix, which takes the following form:

$$\boldsymbol{H}(\mathbf{X}, \mathbf{w}, \mathbf{b}) = \begin{pmatrix} g(X_1 w_1 + b_1) & \cdots & g(X_1 w_{\hat{N}} + b_{\hat{N}}) \\ \vdots & \ddots & \vdots \\ g(X_N w_1 + b_1) & \cdots & g(X_N w_{\hat{N}} + b_{\hat{N}}) \end{pmatrix} \tag{5.9}$$

The variable $\mathbf{X}$, which represents the training batch, is itself a matrix of dimensions $(N, m)$, where $N$ represents the number of examples and m represents the number of samples within the single example (in the present case, $m$ is the size of the input vector and is therefore equal to 6), $g$ represent a generic activation function. $\hat{N}$, on the other hand, represents the number of neurons in the hidden layer, $\mathbf{w}$ and $\mathbf{b}$

are respectively a matrix of dimension $(m, \hat{N})$ and a vector of dimension $(\hat{N})$. The output of the network can then be expressed as:

$$\boldsymbol{HB} = \boldsymbol{Out} \tag{5.10}$$

Where $\boldsymbol{B}$ represents the transfer function of the hidden layer and has dimensions $(\hat{N}, N_{out})$, where $N_{out}$ is precisely the number of output neurons. By providing the network with targets $\boldsymbol{T}$ (having the same size as $\boldsymbol{Out}$), it is possible to calculate $\boldsymbol{B}$ by solving a simple linear problem:

$$\boldsymbol{B} = \boldsymbol{H^{-1}T} \tag{5.11}$$

The calculated $\boldsymbol{B}$ will then be used to make predictions. The knowledge extracted from the data will therefore be contained within the values of $\boldsymbol{B}$. Such a learning process is limited: it has no memory of the past, since at each iteration $\boldsymbol{B}$ is recalculated and the information extracted during the previous iteration is lost. To overcome these limitation, it is possible to use a regularized variant of ELM specialized for online learning: ROS-ELM. In essence, our objective is to solve the following optimization problem:

$$\begin{gathered} Minimize : \frac{1}{2}\|\boldsymbol{B}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|^2 \\ subject\ to : \boldsymbol{H}(\mathbf{X})\boldsymbol{B} = \boldsymbol{T} - \boldsymbol{e} \end{gathered} \tag{5.12}$$

Where $\boldsymbol{e}$ represents the vector of errors committed by the model for each training sample ($\boldsymbol{e} = \boldsymbol{T} - \boldsymbol{Out}$), $C$ is a parameter that assigns a weight to the regularisation $L_2$ for the elements of $\boldsymbol{B}$.
The problem can be solved by deriving a closed-form:

$$\boldsymbol{B} = \left(\frac{1}{C}\boldsymbol{I} + \boldsymbol{H}^T\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{T} \tag{5.13}$$

Assuming we increase the number of examples contained in $\mathbf{X}$ ($\mathbf{X_0}$ to $\mathbf{X_{0\cup 1}} : \mathbf{X_0} \cup \mathbf{X_1}$), the number of rows of $\boldsymbol{H}$ ($\boldsymbol{H_0}$ to $\mathbf{H_{0\cup 1}} : \boldsymbol{H_0} \cup \boldsymbol{H_1}$) will also increase. The overall $\boldsymbol{H}$ matrix takes the following form:

$$\boldsymbol{H_{0\cup 1}} = \begin{pmatrix} \boldsymbol{H_0} \\ \boldsymbol{H_1} \end{pmatrix} \tag{5.14}$$

it is then easy to verify that $\boldsymbol{B_1}$ (i.e. the $\boldsymbol{B}$ relative to $\mathbf{X_{0\cup 1}}$) can be calculated from $\boldsymbol{B_0}$ (i.e. the $\boldsymbol{B}$ relative to $\mathbf{X_0}$):

$$\boldsymbol{B_1} = \boldsymbol{B_0} + \boldsymbol{K_1}\boldsymbol{H_1}^T(\boldsymbol{T_1} - \boldsymbol{H_1}\boldsymbol{B_0}) \tag{5.15}$$

$$\boldsymbol{K_1} = \boldsymbol{K_0} + \boldsymbol{H_1}^T\boldsymbol{H_1} \tag{5.16}$$

where $\boldsymbol{K_1} = \left(\frac{\boldsymbol{I}}{C} + \begin{pmatrix} \boldsymbol{H_0} \\ \boldsymbol{H_1} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{H_0} \\ \boldsymbol{H_1} \end{pmatrix}\right)^{-1}$. It is then possible to generalize these update laws with respect to the k-th iteration:

$$\boldsymbol{B_{k+1}} = \boldsymbol{B_k} + \boldsymbol{K_{k+1}}\boldsymbol{H_{k+1}}^T(\boldsymbol{T_{k+1}} - \boldsymbol{H_{k+1}}\boldsymbol{B_k}) \tag{5.17}$$

$$K_{k+1} = K_k + H_{k+1}{}^T H_{k+1} \tag{5.18}$$

The update process represented by equations 5.17 and 5.18 is employed within the ROS-ELM models utilized in the proposed system.

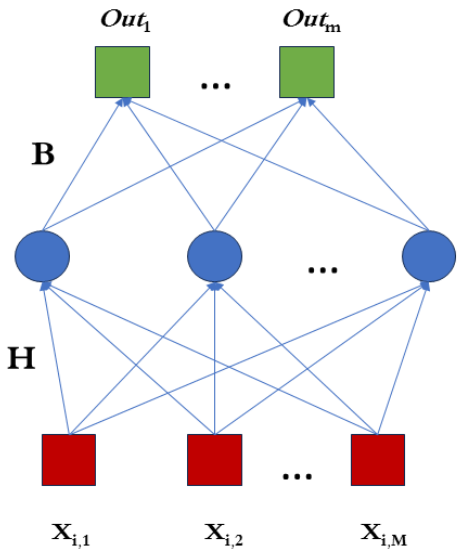


FIGURE 5.15: Graphical representation of an ELM as a single hidden layer neural network, processing the i-th example. The input matrix, represented by $\mathbf{X}$, is mapped by the random function $\boldsymbol{H}$ to produce a latent vector, highlighted in blue. The $\boldsymbol{Out}$ vector is then predicted by the function B.

**Federated Approach** As previously stated, a decentralized and distributed training approach was implemented, enabled by the federated environment that is comprised of two actors: a *server*, which is responsible for managing the entire environment, and a series of *clients* (or nodes), which execute the *server*'s instructions in a federation that is characterized by a Master-Slave(s) approach. In the context of our case study, each node represents a single device used to monitor the individual patient. Consequently, there are 12 *clients*. Each node maintains a local and private dataset that is continuously updated, as it is always measuring new samples of BGL. The intrinsic robustness and security of FL from a privacy perspective is derived from the fact that each node trains a learning model on its own dataset, without sharing it with any other participant in the federation. Subsequently, each *clients* performs its own training process, which is arbitrarily defined by the *server* but consistent across all *clients*. The parameters, computed by the *clients*, which contain the knowledge extracted from the data, are then sent to the *server*. The method by which the parameters from the various *clients* are aggregated is arbitrary and depends on the specific problem. For instance, simple averaging may be employed. The aggregation produces a set of parameters that is compatible with the learning model chosen by the *server* beforehand and it is used to produce a trained global model, which is then sent back to the *clients* so that they can update it on the new data. This is an iterative process that lasts until the end of the available data or up to a set maximum number of iterations (commonly called rounds).

**Initialization** The process of initialising the federated environment involves a number of fundamental steps necessary for proper peer-to-peer efficent communication:

- **Setup**: *clients* wishing to participate in the federation must notify the *server* of their intention.
- **Verification**: In this phase, the *server* verifies that the information sent by the *clients* is correct.
- **Defining the environment**: The *server* informs the participating *clients* of important information: the chosen learning model and its hyperparameters, the maximum number of rounds, the number of iterations per round, the characteristics of the object containing the learned parameters and its size.

**Rounds** : Each round follows the same procedure. All the *clients* in the federation perform one or more training iterations (depending on what was specified during initialisation) and store the parameters resulting from the training in memory, to send it to the *server* at the end of the current round. The training phase ends when the *server* receives positive feedback, concerning the execution of the training calculation, from all the *clients* participating in the federation. The *server* informs all the *clients* to send their parameters. There is then a phase of aggregation of these parameters *server*-side. It results in an object that is compatible with the established learning model, so that global parameters can be obtained, which are sent back to all *clients* and determines the end of the current round.

### 5.5.4 Experimental setup

In this section we present the choices made to adapt the system to the BGL forecasting task and the aspects investigated through a series of experiments.
Regarding the process of constructing the training examples, the step (SWstep) at which the observation window, that generates the input vector, moves over the BGL series, was considered as a hyperparameter to be investigated. We started with a window moving with a SWstep of 1 sample per example (i.e. the j-th example is found lagged by 1 sample from the (j-1)-th example), up to a SWstep of 3 samples per example (i.e. the j-th example is found lagged by 3 samples from the (j-1)-th example). Larger SWsteps have been neglected in order to avoid updating the model too infrequently, which would have been clearly contrary to the objectives of this study, one of which is to propose a predictive system capable of adapting very quickly to the variability of the data.
Focusing on the hyperparameters of ROS-ELM, due to the relative simplicity of the model and its extremely short training times, we decided to tune everything that could be tuned, in particular we find the size of the training batch ($N$, interpreted as the number of examples that the network sees simultaneously and on which it solves the linear problem), the number of neurons in the hidden layer ($\hat{N}$), constrained, as shown in [245], by the size of the batch. We also have the value of the $L_2$ regularisation parameter ($C$) and, finally, the activation function used in the hidden layer ($g$).
In the federated environment, only the aggregation method aspect was investigated, in particular the mathematical function required to aggregate the parameters sended from the *clients* to the *server*. These functions were tested in terms of their robustness against the presence of outlayers artificially inserted into the dataset through dedicated experiments.

For the hyperparameters, a grid search was performed and the ranges of all the investigated values are reported in Table 5.14.

| **Hyperparameter** | **Range** | **Step** |
|---|---|---|
| SWstep | [1, 3] samples | 1 sample |
| Batch Size | [10, 200] | 10 |
| Hidden Neurons | [50, 100] | 10 |
| C | [0.001, 0.1] | 0.005 |
| $g$ | [ReLU, Sigmoid, Tanh] | // |

TABLE 5.14: Ranges of the tuned hyperparameters.

The federated aggregation methods investigated are:

- **The Geometric Mean**: it is the simplest and computationally least expensive aggregation method selected, with a complexity that can be approximated as $\boldsymbol{O(\hat{N}p)}$ where $\boldsymbol{\hat{N}}$ represents the size of the matrix $\boldsymbol{B}$, which, in a regression task, is reduced to a vector of dimensions $\boldsymbol{(\hat{N}, 1)}$, while $\boldsymbol{p}$ represents the number of vectors over which to compute the mean (i.e. the number of participants in the federation, 12 in our case);

- **The Weighted Geometric Mean**: this type of aggregation is characterized by the multiplication of each element of the vector $\boldsymbol{B}$ by a weight ***score*** that quantifies the the inverse of the error (i.e. Root Mean Square Error: RMSE) and thus the accuracy that the single *clients* presents on a local test set containing samples related to the same subject on which the *clients* itself has been trained. The complexity of this method is still comparable to that of the simple Geometric Mean, which can again be approximated by $\boldsymbol{O(\hat{N}p)}$;

- **The Geometric Median (*GM*)**: it is a particularly robust aggregation method in the presence of outliers (vectors $\boldsymbol{B}$ geometrically very distant from all others), it is extremely effective in reducing the deterioration of the global model's performance (consequent to aggregation) in cases of malfunctions or corruption of one or more *clients*. This method has been implemented through an iterative algorithm, whose convergence is guaranteed in a finite number of iterations, through a tolerance set a priori on the distance in the space $\mathbb{R}^{\hat{N}}$ and equal to $10^{-6}$. The complexity of this method grows compared to the previous ones and can be approximated by $\boldsymbol{O(I\hat{N}p)}$ where $\boldsymbol{I}$ represents the number of iterations required for convergence:

$$\underset{\boldsymbol{GM}}{\arg\min} \sum_{i=1}^{p} (\|\boldsymbol{B}^{i,2} - \boldsymbol{GM}\|^2) \tag{5.19}$$

  In the preceding expression, $\boldsymbol{B}^i$ denotes the $\boldsymbol{B}$ vector sent to the *server* by the i-th *client*.

The RMSE is utilized not only within the second aggregation method but also as a performance metric to assess the performance of the global model trained on the test set. However, predictive accuracy does not take into account the clinical risk associated with the predictions made by the model. Therefore, relying solely

on the RMSE would be a superficial approach. For this purpose, a metric widely accepted in diabetes research has been used: the Clarke Error Grid or CEG. The CEG represents a graphical tool for assessing the clinical applicability of blood glucose predictive or measuring instruments. In particular, the CEG is presented as a two-dimensional grid that relates predictions (or measurements) to a reference glycemic value, which is considered reliable. The 2D plan is divided into five zones (A, B, C, D, E) characterized by increasing levels of risk associated with the clinical measures that can be taken consequently to the predictions (or measurements) of the instrument being assessed. The risk levels range from no risk to a risk of serious complications, which may result in death.

**FLOP estimation** It is not straightforward to ascertain the complexity of a training algorithm when the implementation process is unknown. One might consider training time as a metric of complexity, but this is highly dependent on the machine used to perform the calculations. Furthermore, it is not always possible to trace the hardware specifications of the machines used to train the predictive models in the literature. We were assisted by [244], which presents a method for estimating the number of floating-point operations (FLOPs) required for the training process of the most common deep learning architectures. The number of FLOPs represents the number of mathematical operations carried out within an algorithm and allows us to estimate its computational complexity, as opposed to processing times, which are highly dependent on hardware specifications. In the aforementioned article, it is evident that there is a clear distinction between architectures trained with backpropagation and those trained without. For each architecture under investigation, the number FLOPs associated with the processing of either a single training example, a training batch or an entire epoch can be calculated. The training-related complexity is defined as the total processing complexity, including the updating of the architecture parameters. In particular, the latter can be approximated as three times the FLOPs required for processing alone (i.e. without updating the parameters). This allowed us to estimate the complexity involved in the training of the models selected in the literature and highlighted above as comparison models.

**Inference modes** We decided to evaluate performance, and then extract RMSE and CEG, in two different test configurations. The purpose is precisely to show the behavior of the framework in two different application contexts:

- **Test on test set**: In the first case, we simply use the test data provided directly by the publishers of the dataset. In each federated round, after aggregating the parameters from the different *clients*, we have a test phase on the entire test set.
- **Online test**: In the second case, the approach changes. We wanted to simulate a real application context; in fact, in each round, the model should make predictions on data immediately following the data on which it has just learned (a classic online learning condition). To do this, in each round the global model is tested on the training batch immediately following the one it has just seen.

In both scenarios, the performance obtained is the average of the performance obtained by the global model on the data for the 12 different subjects.

### 5.5.5 Results and discussion

First, before presenting the results, you can appreciate the outcome of the grid search in Table 5.15. The hyperparameters shown in that table are those used to obtain all the results.

| Hyperparameter | Value |
|---|---|
| SWstep | 1 samples |
| Batch Size | 100 |
| Hidden Neurons | 100 |
| C | 0.01 |
| $g$ | ReLU |

TABLE 5.15: Optimal hyperparameters obtained via grid search.

**Single Regressor** The trained global model proved to achieve particularly low prediction error values in the test phase, confirming the initial hypotheses about its excellent generalisation capabilities. This is true both for the online test conditions and for the entire test set, as shown in Figure 5.16. In the same figure, the rapid convergence of learning to a quasi-stationary value can also be observed. In particular, the global model achieves an average minimum RMSE (averaged over the 12 *clients*) of 14.73 for the test on test set and 18.48 for the online test during the FL process. The RMSE value converges very quickly after 20 rounds in both test cases, suggesting a model pre-tuning window of about one week (consistent with the sampling times used in the acquisition of this dataset) before reliable performance is achieved. It can be seen that there is a substantial difference between the mean error committed by the global model in consecutive rounds, evidenced by high variability in the trend of RMSE. The percentage absolute error committed in relation to the target BGL value is depicted in Figure 5.17, to highlight the difficulties encountered by the model in making predictions under different glycemic conditions. It can be observed that the predictive accuracy is particularly high in the euglycemic zone, but it deteriorates in the hypoglycemic and hyperglycemic zones but this is justified by the imbalance in the dataset between the different glycemic conditions.
With regard to the clinical applicability of the method, the analysis carried out by the CEG showed a good degree of clinical reliability. In fact, as can be seen in Figure 5.18 a percentage of predictions greater than 97.88% can be found in zones A and B (characterized by low risk) for all subjects, a percentage of predictions of about 0.02% in zone C and about 2.20% in zone D. As can be seen from the graph of the CEG, the predictions are distributed on or around the diagonal.

**Triple Regressor** In the case of the triple regressor model, performance improves significantly. A minimum RMSE averaged over the 12 subjects of 13.73 in the case of the on test set and 12.03 in the case of the online test is achieved, as can be appreciated in Figure 5.19. There is a clear advantage in terms of RMSE over the single regressor model in both test cases, also both the variability of the mean RMSE and the mean standard deviation over all rounds are reduced. This is due to the greater accuracy with which predictions are made mainly in the hypoglycemic zone


FIGURE 5.16: Representation of the average error committed by the global model in testing phase for both testing methods. The solid lines indicate the RMSE value averaged over the data of the 12 subjects, while the opaque area indicates the standard deviation obtained under the same conditions.

but also in hyperglycemic one, as the triple regressor model provides specialized sub-models for different glycemic conditions. Figure 5.20 is intresting because it indicates that employing three regressors significantly enhances the performance achieved in hypoglycemia compared to the single regressor, while the performance on hyperglycemia remains essentially unaltered. The reason for the fluctuations in this graph (for BGL equals to 80 mg/dL and 100 mg/dL) is due to the method used to select the expert sub-model for the current forecast. As mentioned above, this method is implemented using a hard approach, which inevitably makes the transitions discontinuous. However, it should be noted that these oscillations are limited to a range in which the percentage of error is under 6%. The utilization of the three regressors does not affect the rate of convergence during training. This is reasonable to assume, given that no novel information is introduced into the dataset in comparison to the single regressor case, and that the three regressors do not share any internal information. It is quite obvious that the introduction of specialized models for different glycemic conditions reduces the level of risk associated with predictions. This is evident from the CEG in Figure 5.21. In particular, we have a 98.75% prediction rate for the trained global model in zones A and B, about 0.08% in zone C and about 1.16% in zone D. In general, from Figure 5.21, it is possible to visually observe a decrease in the concentration of points in zone D between 0 and 70 mg/dL of the reference value, indicating a reduction in risk compared to the single regressor case; there is also a greater adherence of the points to the plane bisector.
It is important to note that the introduction of the three regressors does not significantly increase the complexity of the algorithm. In fact, the additional FLOPs

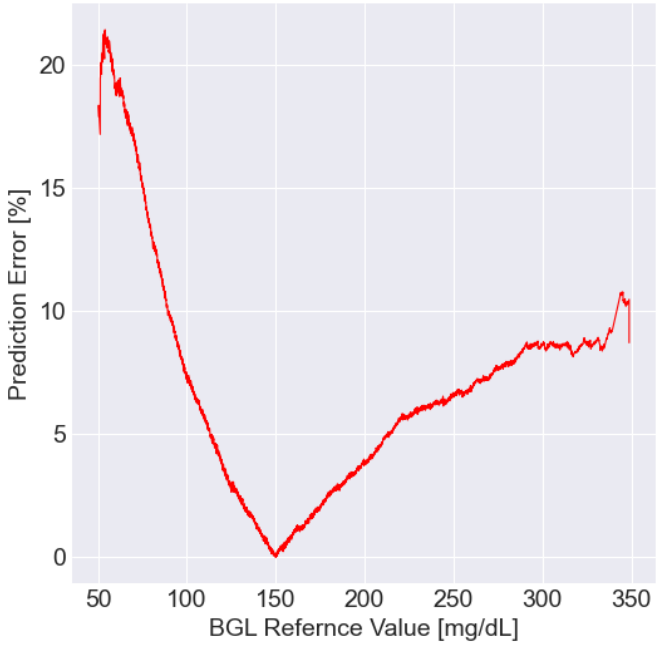


FIGURE 5.17: Percentage error committed by the global model. The graph shows a high level of predictive accuracy within the euglycemic zone (100–180 mg/dL), with a marked decrease in accuracy for BGL values below 100 mg/dL, as well as in the hyperglycemic zone (BGL $>$ 180 mg/dL).

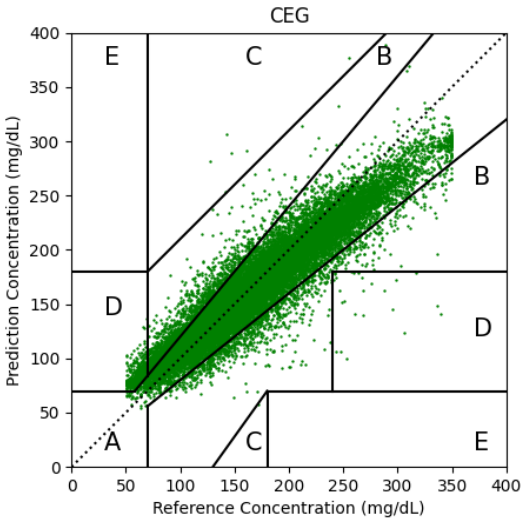


FIGURE 5.18: Distribution of predictions generated by the global model on the test set of data from 12 subjects, within the five zones of the CEG.

compared to the single regressor case can be considered negligible. This is true because, during training, for each training sample, the competence model is selected

from the input vector for the network, as previously explained. This mutually exclusive model selection system presents a computational complexity that can be neglected. The additional complexity that must be considered pertains to the training of the model under conditions that are dubious in terms of glycemic control. These conditions are those that are situated near the thresholds that separate euglycemia from hypoglycemia and euglycemia from hyperglycemia. In such instances, all the sub-models in question are trained simultaneously in order to facilitate a seamless transition between sub-models in terms of RMSE and percentage error.

**Federated aggregation** The outcome of the robustness experiments demonstrated the previously established properties of the geometric median, which was identified as the optimal aggregation method among those investigated. It is noteworthy that the advantages of employing the geometric median are only discernible under specific circumstances, such as *clients*-side malfunctions or *server-clients* communication issues. In such instances, the geometric median is observed to result in a reduction in the extent of performance deterioration in comparison to the other methods that were investigated.

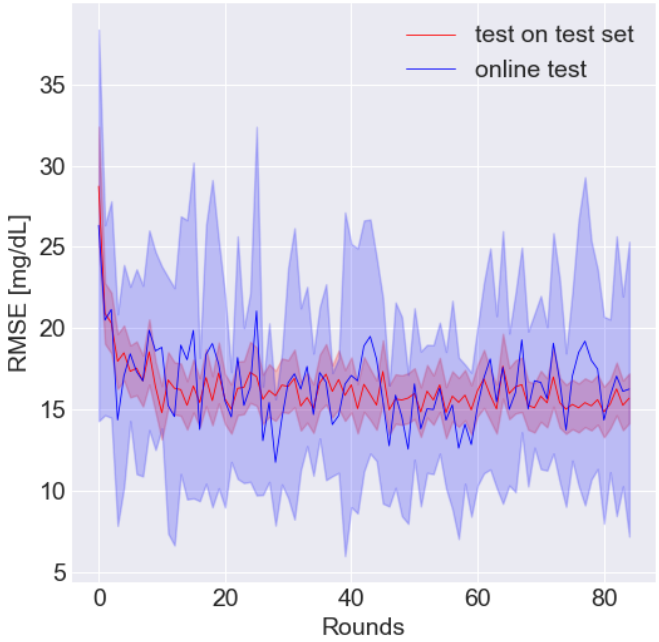


FIGURE 5.19: Representation of the average error committed by the triple regressor in both tests. The RMSE was plotted in relation to the federated rounds, obtained by performing inference by both the test on test set and online test. The solid lines indicate the RMSE average value, while the opaque areas indicate the standard deviation obtained under the same conditions.

**Comparison with literature** Throughout the experimental phase, the proposed system has shown remarkable capabilities that demonstrate its validity and superiority over the comparison models. In particular, the triple regressor FedROS-ELM has been shown to obtain overall lower RMSE values than comparison models.

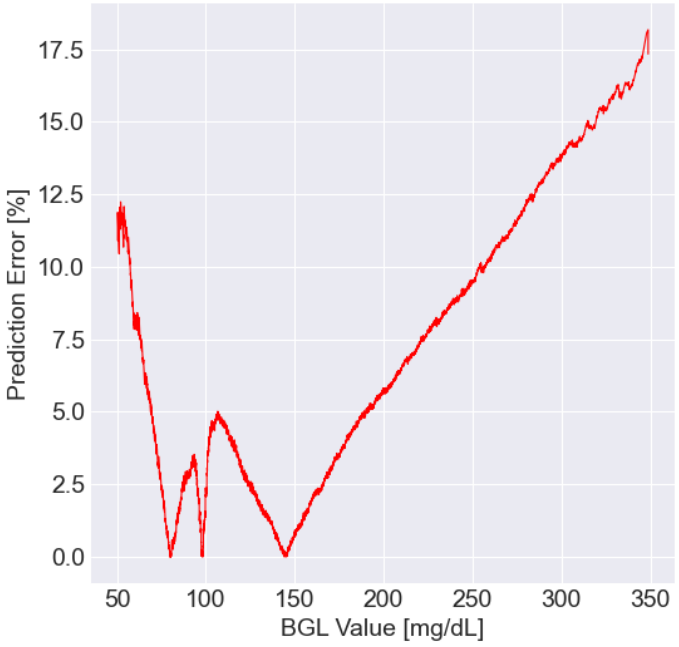


FIGURE 5.20: Representation of the percentage error committed by the global triple regressor model.

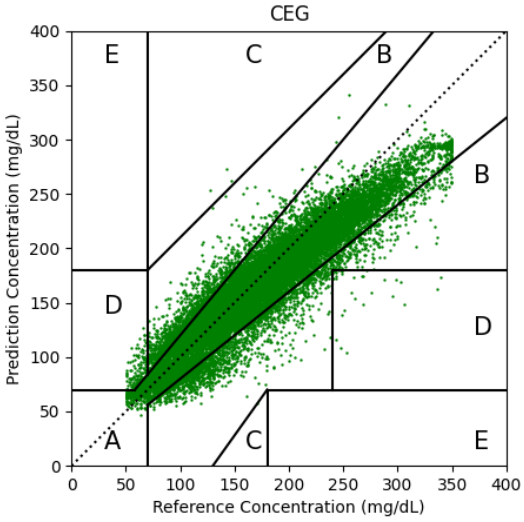


FIGURE 5.21: Distribution of predictions performed by the global triple regressor model within the five zones of the CEG.

However, it is important to note that the performance of [32] is affected by an analysis that includes data from both the Ohio T1DM dataset and a private dataset, a circumstance that could compromise the rigour of the direct comparison.

The trade-off between performance and complexity of the proposed model is particularly favourable. This is demonstrated by an average RMSE of 15.03 mg/dL, achieved with a notably low computational cost of approximately $10^3$ FLOPs. This

represents a 10.59% reduction in RMSE compared to the best-performing competitor model [32], which, in contrast, demands three orders of magnitude more FLOPs. Importantly, the use of linear interpolation for missing samples in [32] may have simplified the prediction process, directly affecting the performance comparison. Table 5.16 summarizes the comparison.

| Model | **RMSE** | FLOPs |
|---|---|---|
| Proposed | **15.03** | $\sim$ **$10^3$** |
| [241](1) | 19.63 | $\sim 10^7$ |
| [241](2) | 19.64 | $\sim 10^8$ |
| [241](3) | 19.62 | $\sim 10^8$ |
| [242] | 19.97 | $\sim 10^6$ |
| [32] | 16.81 | $\sim 10^6$ |
| [243] | 18.87 | $\sim 10^6$ |

TABLE 5.16: Comparison between the proposed approach and models selected from the literature in terms of RMSE and number FLOPs.

## 5.6 Graph driven Federated For Healthcare 4.0

As previously stated FL provides a decentralized framework for machine learning, enabling multiple participants to collaboratively refine a global model without directly exchanging data [16]. To address challenges such as client heterogeneity and communication inefficiencies, federated aggregation of diverse client data can be optimized through strategic approaches like partitioning clients into cohorts. This method has been shown to improve outcomes and accelerate convergence rates in FL setups, allowing for more effective handling of heterogeneity, streamlined communication protocols, and efficient model updates [246, 247].

Within this context, we present a Graph-aided FL (GaFL) specifically designed for predicting blood glucose levels (BGL). We propose a mathematical framework that models patient similarities using a graph-based approach, providing a structured and quantifiable representation of these relationships. This framework is complemented by a triple-regressor system, where each regressor is a Robust Online Sequential Extreme Learning Machine (ROS-ELM) specialized in a specific glycemic condition. Our approach is inspired by traditional clinical practices, where physicians draw on insights from previous patient cases to guide their decision-making without requiring explicit data sharing.

Our main contributions are:

- The implementation of a novel framework GaFL applied to glycemic level prediction, able to model similarities between patients in a structured and quantifiable way.
- Extensive testing of our flexible and adaptable system on two different datasets, demonstrating its effectiveness in leveraging Online Learning (OL) for glycemic level management.

### 5.6.1 System description

In this section, we present the architectural overview of the proposed system, emphasizing its dual structure that combines a robust delivery framework with an integrated machine learning component. First, we describe the primary actors *Peer*, *Aggregator*, and *Certified Authority*, outlining their roles in ensuring transparency and efficiency. Second, we examine the operational procedures, focusing on the *Enrollment* and *Operational* phases, which maintain secure, decentralized coordination. Third, we introduce the method used to generate the patient graph and its relevance. Finally, we present the machine learning model, detailing its contribution to optimizing the FL process and improving predictive accuracy.

**Actors** The system involves three key actors: the *Peer*, the *Aggregator*, and the *Certified Authority*. The *Peer* holds the patient's digital identity or acts on their behalf, serving as the primary source of personal health data and consenting to data sharing or updates. Without transferring raw data, the *Peer* trains models locally on personal information. The *Aggregator* collects the model weights and biases from the *Peers* and performs the aggregation phase, following the directives of the *Certified Authority*. The *Certified Authority* manages and verifies the digital identities of all entities, ensuring that operations comply with healthcare regulations and privacy standards. It oversees rules in the federated system, verifies the credentials of *Peers* and *Aggregators*, and maintains system integrity and security.

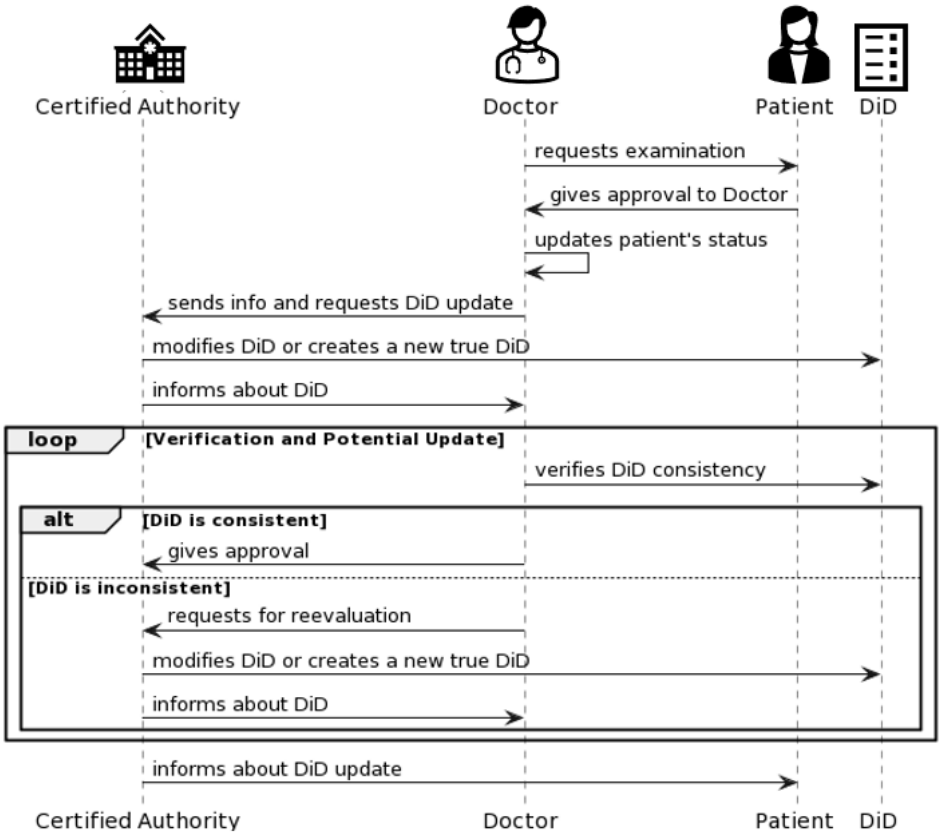


FIGURE 5.22: Visual representation of the Subscription phase

**Enrollment Phase** The enrollment phase initiates with a doctor requesting an examination from a patient, which could stem from either routine health monitoring or specific health concerns raised by either the patient or the physician. Upon the patient's approval, which is mandatory under privacy regulations and ethical standards, the doctor proceeds to update the patient's health status.

Following the patient's approval, the doctor submits the updated health information to a *Certified Authority*, as depicted in Figure 5.22. This submission includes a request for an update to the patient's DiD to reflect their latest health status accurately. The *Certified Authority*, upon receiving this information, either modifies the existing DiD or creates a new one that truly represents the current health information of the patient, ensuring that the DiD is an accurate reflection of the patient's current health condition.

Once the DiD is updated or created, the *Certified Authority* informs the doctor of the changes. A verification loop is then initiated to ensure the consistency and accuracy of the updated DiD. If the DiD is consistent with the provided information and meets all regulatory standards, the *Certified Authority* approves the update. However, if inconsistencies are detected, the doctor may request a reevaluation, leading potentially to further adjustments to the DiD by the *Certified Authority*.

After achieving a consistent and accurate DiD, the *Certified Authority* finalizes the update. Both the doctor and the patient are then notified about the updated DiD, ensuring both are informed of the changes made to the patient's DiD.

**Operational phase** The operational phase, depicted in Figure 5.23, represents the learning phase of our GaFL framework. This phase involves an iterative process

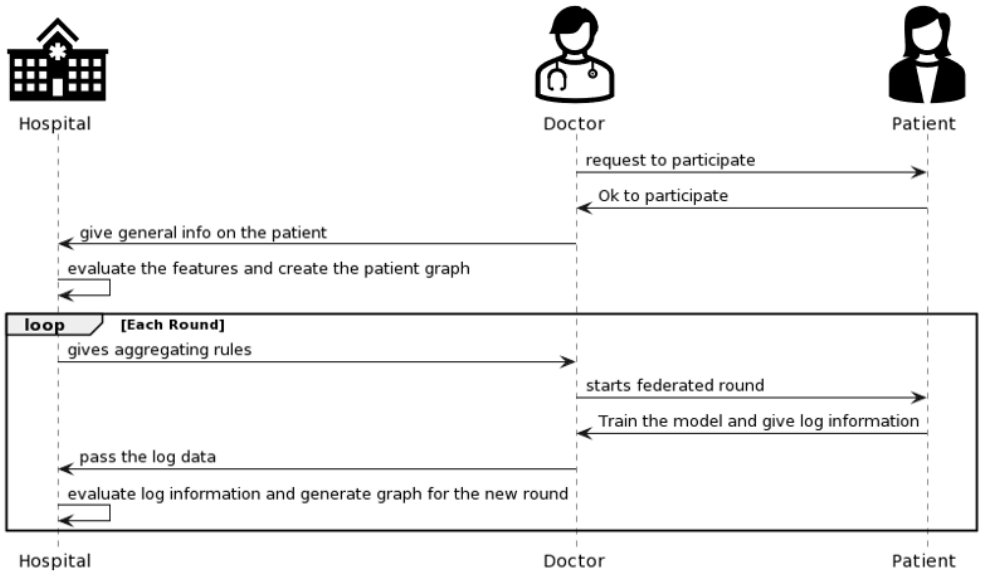


FIGURE 5.23: Visual representation of the operational phase

that enhances the management and application of patient data through a collaborative model involving *Certified Authority*, *aggregators*, and *peers*. The process begins with a *aggregator*'s request to participate in the FL system. Upon obtaining consent from the *peers*, the *aggregator* provides the *Certified Authority* with general information about the FL process, i.e., establishing the subset of feature pesent in the DiD of the *peer* usefull for the learning process. This initial exchange is critical as it sets the parameters needed for the generation of the patients graph. The generation of the patient graph, as explained in subsection5.6.1, can be replaced by another technique for identifying clusters.

Once the patient graph is established, the *Certified Authority* communicates the rules for data aggregation to the *aggregator*. These rules dictate how *peers* models should be averaged, ensuring consistency and accuracy across different nodes in the federated system. The *aggregator* then trigger the start of the FL round, where they apply these rules to train a model on the aggregated data. Throughout this process, the model's performance and any insights gained are logged and communicated back to the *Certified Authority*. The *Certified Authority* plays a continuous role in evaluating the information received from the logs. This evaluation is essential not only for refining the patient graph and adapting the learning model for subsequent rounds but also to enable the audit of the learning process. Through successive rounds of FL, the model evolves to become more attuned to the nuances of patient care, ultimately leading to optimized health outcomes.

**Patient graph construction** At this stage, we refine the FL by incorporating a patient similarity metric into the process. This entails constructing a graph-based framework to cluster patients into cohesive cohorts, thereby improving both computational efficiency and personalization. The following paragraphs outline the rationale, the calculation of similarity scores for different feature types, and the integration of these components into a unified similarity measure.

To tackle the challenges associated with FL, our methodology incorporates a clustering mechanism within the patient graph. This mechanism forms narrowly

defined cohorts of federated peers, which reduces the computational overhead by limiting the scope of data interaction and synchronization to more homogeneous groups of data holders. This strategy enhances the efficiency and scalability of the learning process. Moreover, the patient graph not only supports efficient FL but also ensures that the insights derived are deeply personalized and contextually relevant, capturing the inherent patterns and similarities within the data.

To encapsulate the nuanced decision-making process of physicians, who often intuitively weigh patient similarities based on their professional experiences and observations, building on the formulation presented in [248], we have modified it to accommodate not only categorical information but also discrete features. This enhancement enables the model to handle a broader spectrum of data types, thereby increasing its applicability and flexibility in various analytical contexts.

In order to assign a pairwise similarity score between all patients, we distinguish the features as either categorical or discrete. First, we transform the discrete features into a multi-hot vector for each instance, resulting in a categorical feature matrix $D_{\mu_c} \in \mathbb{R}^{N \times m_c}$ where $m_c$ is the number of unique categorical features and $N$ is the number of patients. The categorical similarity score $Mc_{ij}$ between nodes $i$ and $j$ is defined as:

$$Mc_{ij} = a_1 \sum_{\mu_c=1}^{m} \left( D_{i\mu_c} D_{j\mu_c} \left( d_{\mu_c}^{-1} + c_1 \right) \right) - \sum_{\mu_c=1}^{m} \left( D_{i\mu_c} + D_{j\mu_c} \right) \tag{5.20}$$

where $d_{\mu_c}$ is the occurrence of a categorical feature $\mu_c$, and $a_1$ and $c_1$ are tunable constants. The first term positively rewards shared features. Note that the $d_{\mu_c}^{-1}$ term incorporates the idea that two patients sharing a rare feature is more significant than a common one. The second term penalizes the total number of features to prevent patients with many features becoming 'hubs' of high connectivity, attracting imprecise matches with several non-shared features.

Subsequently with respect to discrete features, first these are normalized feature by feature among all patients, resulting in a discrete feature matrix $D_{\mu_d} \in \mathbb{R}^{N \times m_d}$ where $m_d$ is the number of discrete features and $N$ is the number of patients. The discrete similarity score $Md_{ij}$ between nodes $i$ and $j$ is defined as:

$$\begin{aligned} Md_{ij} = a_2 \sum_{\mu_d=1}^{m} \left( \left( 1 - \frac{1}{1 + e^{-\frac{1}{\left| D_{i\mu_d} - D_{j\mu_d} \right|}}} \right) \left( d_{\mu_d}^{-1} + c_2 \right) \right) + \\ - 2 \sum_{\mu_d=1}^{m} \left( 2 - \left( \frac{1}{1 + e^{-\frac{1}{\left|\left| D_{i\mu_d} - D_{\mu_d \text{mean}} \right|\right|}}} + \frac{1}{1 + e^{-\frac{1}{\left|\left| D_{j\mu_d} - D_{\mu_d \text{mean}} \right|\right|}}} \right) \right) \end{aligned} \tag{5.21}$$

where $d_{\mu_d}$ is the occurrence of a discrete feature $\mu_d$, and $a_2$ and $c_2$ are tunable constants. The first term positively rewards shared features. Note that the $d_{\mu_d}^{-1}$ term incorporates the idea that two patients sharing a rare feature is more significant than a common one. As for the categorical similarity score the second term prevent patients with features values close to the mean becoming 'hubs' of high connectivity.

Once the discrete and categorical similarity score of two nodes have been calculated, their similarity score will be given by:

$$M_{ij} = Mc_{ij} + Md_{ij} \tag{5.22}$$

This score is specifically designed to mirror a clinician's heuristic approach to evaluating patient similarity, integrating both clinical relevance and the rarity of shared conditions to construct a mathematically robust, yet intuitively aligned, framework for patient comparison in Healthcare environments.

**Models description** In this study, we employed an ELM-based predictive model, the Robust Online Sequential Extreme Learning Machine (ROS-ELM) [249], which is notable for its low computational cost. The decision to utilize ROS-ELM was driven by the necessity to address the elevated complexity of the proposed frameworks, which entail the training of multiple models within a single federated round. The single predictor is composed of three ROS-ELM regressors, each of which is trained on a different glycemic level. Two thresholds are employed to assign each sample in the dataset to a competency model: one that divides euglycemia from hypoglycemia (100 mg/dL) and one that divides euglycemia from hyperglycemia (180 mg/dL).

Although the dataset has been divided into the sections previously described, the sequence, in which the global model is trained and how the different local models interpret the time series, remains unchanged. For each training instance, the glycemic condition in the input vector is verified, and based on this, the most suitable regressor for training is selected. Specifically, the glycemic condition is identified by averaging over the input vector samples, and the resulting value is then compared with the pre-established thresholds.

### 5.6.2 Case study

In order to clarify the advantages and disadvantages of the proposed methodology, two clinical datasets were selected for analysis: the Ohio T1DM Dataset [197] and the Campus Bio-Medico Dataset.

**Experimental setup** In order to provide a federated clustering framework based on the information contributed by a similarity metric, such as the one previously explained, three different pipelines were developed with the intention of offering a solution that would be applicable in a variety of scenarios. In this section whenever we refer to a model we refer to the triple regressor model i.e. a multiexpert model. **Pipeline 0: GM:** represents the fundamental reference approach for subsequent pipelines. It is composed of the straightforward aggregation of parameters extracted from each client within the federation, employing the geometric median as the aggregation rule. **Pipeline 1: Clustering-GM:** is based on the key idea of dividing the set of subjects participating in the federation into two or more clusters maximizing the inter-subject similarity of each cluster. The number and the composition of clusters is determined through the maximization of the clustering silhouette score. The clustering algorithm used is that of spectral clustering performed on the adjacency matrix of the patient similarity graph. Specifically, the algorithm computes the Laplacian of the adjacency matrix, extracts its leading eigenvectors, and applies k-means and Dbscan clustering to partition the data based on these spectral features. Subsequently, the learning process adheres to the standard procedures of a federated environment, with the sole exception that federated aggregation is conducted exclusively among clients belonging to the same cluster, and the global model thus computed is returned to the same clients. Ultimately, the aggregation method selected for this pipeline is the geometric median. **Pipeline 2: Clustering-Similarity-Avg:** a modification to the methodology of Pipeline 1. The key difference

is the addition of an aggregation rule that leverages similarity information. Specifically, within each cluster, the knowledge from each client is aggregated using a weighted average, where each client's weight is calculated as the sum of its similarity scores with all other clients in the cluster, divided by the total sum of all similarity scores within the cluster. Let $C_k$ be the $k$-th cluster. $N_k$ be the number of clients in cluster $C_k$. Let $s_{ij}$ represent the similarity score between client $i$ and client $j$. $w_i$ denote the weight assigned to client $i$ in the aggregation. The weight $w_i$ for each client $i$ in cluster $C_k$ is computed as:

$$w_i = \frac{\sum_{j \in C_k} s_{ij}}{\sum_{i \in C_k} \sum_{j \in C_k} s_{ij}}$$

where the numerator represents the sum of similarity scores between client $i$ and all other clients in the cluster, and the denominator represents the total sum of all similarity scores within the cluster $C_k$. Using these weights, the knowledge from each client in the cluster is aggregated via a weighted average. **Pipeline 3: Similarity-Avg:** is the most computationally complex, as it necessitates that each individual client train multiple models, with a maximum of the number of clients present within the entire federation. In the assumption that $C$ represents the number of clients engaged in the federation and let $i$ be an index such that $i = \{1, 2, 3, \ldots, C\}$, the process is conducted in accordance with the following steps:

- *First Round*: The client trains a local model on individual patient data, after which the knowledge from each client is aggregated to generate $C$ models. Each of these models is generated by aggregating the knowledge from $i - th$ client with the knowledge from all the others according to a weighted average, with each weight representing the similarity of $i - th$ subject with all the others. At the conclusion of this procedure, all $C$ models are passed to each client, thus ensuring that each client possesses $C$ models.

- *Other Rounds*: Given that each client now has access to $C$ models, it trains each one on its own data. The server will receive $C^2$ models for aggregation according to the following criterion: the $i - th$ model from $i - th$ client is aggregated with the $i - th$ model from all other clients through a weighted average rule, wherein the weight represents the similarity between $i - th$ subject and all others.

The objective of the experiments was to highlight three distinct metrics. The following metrics were employed: root mean square error (RMSE) versus rounds, percent prediction error (i.e., prediction error as a percentage of the target glycemic value), and Clarke Error Grid (CEG), a metric commonly utilized in the context of glycemic value monitoring or prediction tools. For all pipelines, these tree metrics are extracted for each global model generated, with the exception of pipeline 3, where each client has a personalized model.

### 5.6.3 Results and Analysis

The experimental phase played a crucial role in highlighting the distinctive characteristics of each pipeline across different training stages. Specifically, the analysis focused on: accuracy and consistency, evaluated through metrics such as RMSE and standard deviation, clinical applicability, assessed using the Clarke Error Grid , a metric designed to evaluate the predictive method's alignment with clinical relevance, model reliability in relation to the patient's glycemic status, examined through

the Error Percentage metric. It is notable that the Ohio T1DM Dataset, with respect to Campus Bio-Medical Dataset, offers the opportunity for more in-depth training due to the availability of a greater quantity of data.

**Comparative Analysis of RMSE and Standard Deviation Across Pipelines** To identify areas of competitive advantage relative to the state of the art, we conducted an analysis of pipelines in terms of accuracy (RMSE) and consistency (standard deviation). These metrics pertain to the behavior of the global model, which infers on the data from the two datasets. RMSE and standard deviation are calculated across different subjects for each federated round and separately for each dataset or cluster (in case a pipeline includes a split into clusters).

**Ohio T1DM Dataset** As detailed in **Table** 5.17 and illustrated in **Figure** 5.24, the performance of the proposed pipelines highlights key trends in accuracy and stability across different clients. **Pipeline 0**, serving as the baseline, exhibited moderate variability but left room for significant improvement in both accuracy and robustness. With **Pipeline 1**, substantial enhancements were observed, particularly in terms of increased consistency and reduced variability, suggesting the effectiveness of the modifications introduced. **Pipeline 2** further refined these improvements, achieving greater accuracy and robustness compared to its predecessors, making it the most reliable among the pipelines. In contrast, **Pipeline 3**, while showing improved variability relative to the baseline, did not surpass the accuracy and consistency of **Pipelines 1** and **2**. Overall, the results reveal a clear trend of progressive enhancement from **Pipeline 0** to **Pipeline 2**, highlighting the crucial role played by the similarity metric in assigning appropriate weights to each subject within the dataset, during aggregation. The observed trade-offs in **Pipeline 3** underscore the need to balance accuracy and variability based on specific application requirements.

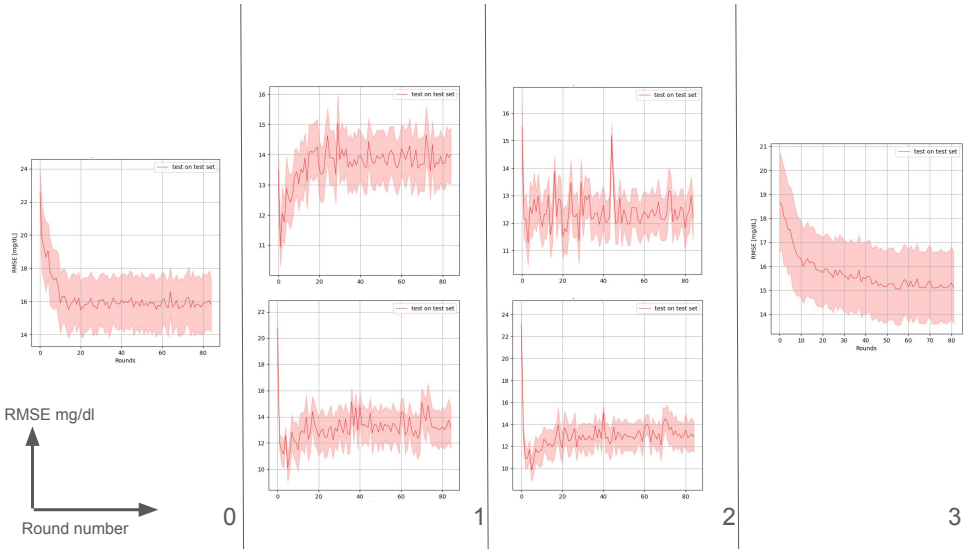


FIGURE 5.24: The data presented in the figure are calculated using the OHIO dataset. The numbers from 0 to 3 identify the pipelines depicted in each plot. Each axis represents the measures defined in the legend located at the bottom left, such as RMSE (in mg/dL) and Round number.

**Campus Bio-Medico Dataset** The analysis conducted on this dataset aligns with the trends observed in the Ohio T1DM Dataset, confirming the consistency of the pipeline behaviors. However, in this case, the overall accuracy appears consistently higher, reflecting a better adaptation of the models to this dataset, as shown in **Table** 5.17 and **Figure** 5.25. At the same time, the variability between clients is slightly greater, likely due to increased heterogeneity within the data. These results further emphasize the robustness of the proposed approach while highlighting the influence of dataset-specific characteristics on performance.

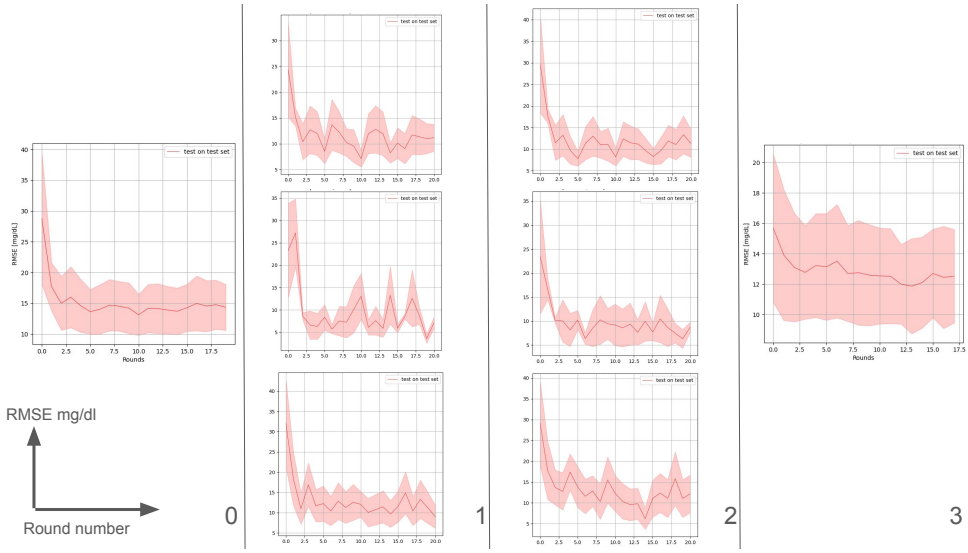


FIGURE 5.25: The data presented in the figure are calculated using the Campus Biomedico dataset. The numbers from 0 to 3 identify the pipelines depicted in each plot. Each axis represents the measures defined in the legend located at the bottom left, such as RMSE (in mg/dL) and Round number.

To summarize, the graphs in **Figure** 5.24 and **Figure** 5.25 provide a visual summary of the RMSE values for both datasets, highlighting the differences in predictive performance across the pipelines. Overall, **Pipeline 2** demonstrates the best balance between accuracy (as indicated by RMSE) and consistency (as indicated by standard deviation), making it the most effective approach for the datasets under study. **Pipeline 1** also performs well, especially with clustering, but Pipeline 2 offers the highest robustness and lowest error, suggesting that its advanced methodologies contribute significantly to improved model performance.

**Comparative Analysis of Clarke Error Grid Across Pipelines** The Clarke Error Grid provides an assessment of the clinical accuracy of glucose predictions by categorizing predictions into different zones (A, B, C, D, and E) based on the risk associated with erroneous predictions. Zones A and B represent clinically acceptable predictions, while zones C, D, and E represent increasingly severe errors.

A summary of the distribution of predictions across the Clarke Error Grid zones for each pipeline and dataset is provided in **Table** 5.18 allowing for a straightforward comparison of clinical accuracy.

**Ohio T1DM Dataset** **Pipeline 0**, serving as the baseline, displayed a moderate concentration of predictions in Zone A, indicative of clinically accurate outcomes,

TABLE 5.17: RMSE and Standard Deviation for Each Pipeline and Dataset

| Pipeline | Dataset | RMSE | Standard Deviation |
|---|---|---|---|
| Pipeline 0 | Ohio T1DM | 14.39 | 1.16 |
| Pipeline 0 | Campus Bio-Medico | 13.18 | 4.36 |
| Pipeline 1 | Ohio T1DM | 11.62 / 10.64 | 1.42 / 0.92 |
| Pipeline 1 | Campus Bio-Medico | 10,48 / 5.64 / 5.59 | 3.18 / 3.10 / 4.56 |
| Pipeline 2 | Ohio T1DM | 11.02 / 9.52 | 0.71 / 1.26 |
| Pipeline 2 | Campus Bio-Medico | 7.57 / 3.76 / 7.24 | 3.70 / 3.47 / 4.78 |
| Pipeline 3 | Ohio T1DM | 14.61 | 1.55 |
| Pipeline 3 | Campus Bio-Medico | 10.36 | 3.85 |

but a notable portion fell into Zone B, with small percentages spread across the remaining zones. This distribution reflects the initial limitations in the model's ability to minimize clinically less desirable predictions. With **Pipeline 1**, there was a marked improvement in accuracy, particularly evident in the increased proportion of predictions in Zone A and the corresponding reduction in Zone B. This trend was consistent across both clusters, indicating the positive impact of the modifications introduced in this pipeline on clinical reliability. **Pipeline 2** further built on these improvements, achieving the highest proportion of predictions in Zone A and reducing errors in other zones to minimal levels. This performance underscores the pipeline's effectiveness in maintaining clinically safe predictions while minimizing less desirable outcomes in Zones B, C, D, and E. Finally, **Pipeline 3** demonstrated a stable distribution similar to Pipeline 2, with predictions concentrated primarily in Zone A and a small percentage in Zone B. However, a slightly higher proportion of predictions fell into Zones D and E compared to Pipeline 2, indicating a minor trade-off in clinical accuracy.

**Campus Bio-Medico Dataset** **Pipeline 0** demonstrates a higher proportion of predictions in Zone A compared to the baseline of the Ohio dataset, indicating better initial clinical reliability. This trend of improved reliability persists across all pipelines, with **Pipeline 1** and **Pipeline 2** achieving excellent distributions in Zone A and minimal percentages in Zone B, reinforcing the effectiveness of the proposed approach. However, **Pipeline 3**, while maintaining good clinical reliability, shows a slight decline in performance compared to Pipelines 1 and 2, similar to the behavior observed in the Ohio dataset.

The graphs in **Figure** 5.27 and **Figure** 5.26 provide a visual summary of the Clarke Error Grid distribution for both datasets, illustrating the clinical accuracy of the different pipelines.

Overall, **Pipeline 2** demonstrates the best clinical safety (as indicated by Clarke Error Grid distribution), making it the most effective approach for the datasets under study. **Pipeline 1** also performs well, especially with clustering, but Pipeline 2 offers the highest robustness and lowest clinical risk.

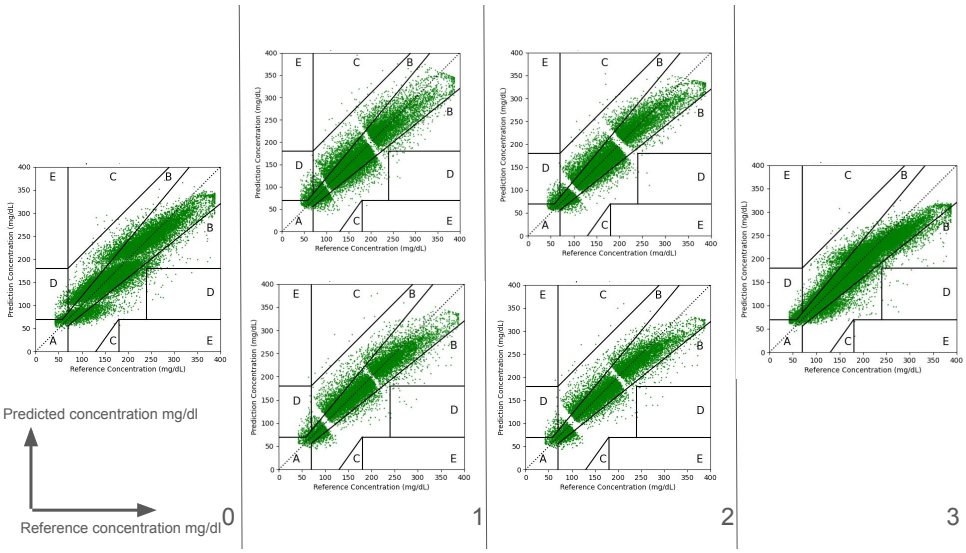


FIGURE 5.26: The CLARK error grids presented in the figure are calculated using the OHIO dataset. The numbers from 0 to 3 identify the pipelines depicted in each plot. Each axis represents the measures defined in the legend located at the bottom left, such as predicted concentration (in mg/dL) and reference concentration.

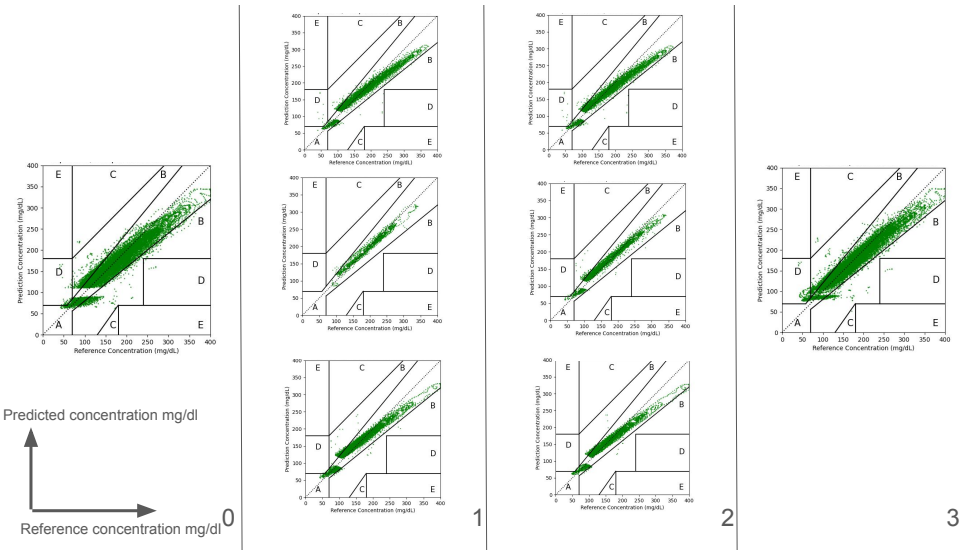


FIGURE 5.27: The CLARK error grids presented in the figure are calculated using the Campus Biomedico dataset. The numbers from 0 to 3 identify the pipelines depicted in each plot. Each axis represents the measures defined in the legend located at the bottom left, such as predicted concentration (in mg/dL) and reference concentration.

**Comparative Analysis of Percentage of Error Across Pipelines** In this section, we compare the performance of the different pipelines based on the average percentage error in the predictions across different glycemic ranges. Specifically, glycemic ranges are categorized as Hypoglycemia (A) for values below 100 mg/dL, Euglycemia (B) for values between 100 and 200 mg/dL, and Hyperglycemia (C) for values above 200 mg/dL. The percentage error is a useful metric for estimating the accuracy of the predictive model in relation to target glucose levels.

A summary of the percentage error for each glycemic range, pipeline, and dataset

TABLE 5.18: Clarke Error Grid Distribution for Each Pipeline and Dataset

| Pipeline | Dataset | Zone A (%) | Zone B (%) | Zone C (%) | Zone D (%) | Zone E (%) |
|---|---|---|---|---|---|---|
| Pipeline 0 | Ohio T1DM | 85.23 | 13.67 | 0.03 | 1.04 | 0.01 |
| Pipeline 0 | Campus Bio-Medico | 92.02 | 7.53 | 0.04 | 0.40 | 0.00 |
| Pipeline 1 | Ohio T1DM (Cluster 1) | 80.45 | 17.59 | 0.11 | 1.82 | 0.01 |
| Pipeline 1 | Ohio T1DM (Cluster 2) | 85.08 | 13.52 | 0.10 | 1.28 | 0.01 |
| Pipeline 1 | Campus Bio-Medico (Cluster 1) | 96.76 | 2.98 | 0.03 | 0.20 | 0.02 |
| Pipeline 1 | Campus Bio-Medico (Cluster 2) | 98.67 | 1.33 | 0.00 | 0.00 | 0.00 |
| Pipeline 1 | Campus Bio-Medico (Cluster 3) | 96.00 | 3.83 | 0.00 | 0.17 | 0.00 |
| Pipeline 2 | Ohio T1DM (Cluster 1) | 84.59 | 13.70 | 0.07 | 1.64 | 0.00 |
| Pipeline 2 | Ohio T1DM (Cluster 2) | 86.31 | 12.34 | 0.08 | 1.24 | 0.01 |
| Pipeline 2 | Campus Bio-Medico (Cluster 1) | 97.86 | 1.95 | 0.00 | 0.19 | 0.00 |
| Pipeline 2 | Campus Bio-Medico (Cluster 2) | 96.52 | 3.27 | 0.00 | 2.01 | 0.00 |
| Pipeline 2 | Campus Bio-Medico (Cluster 3) | 97.11 | 2.73 | 0.00 | 0.15 | 0.00 |
| Pipeline 3 | Ohio T1DM | 85.29 | 11.95 | 0.02 | 2.73 | 0.01 |
| Pipeline 3 | Campus Bio-Medico | 92.01 | 5.76 | 0.03 | 2.19 | 0.00 |

is provided in **Table** 5.19 offering a clear comparison of predictive accuracy for different glycemic conditions. Furthermore, the images in **Figures** 5.28 and 5.29 show how the error percentage behaves on the OHIO and Campus Biomedical datasets, respectively, in relation to the value of BGL to be predicted.

**Ohio T1DM Dataset** Across all pipelines, the glycemic range significantly influenced the percentage error, with higher errors consistently observed in the hypoglycemic range and reduced errors in the euglycemic and hyperglycemic range. **Pipeline 0**, serving as the baseline, exhibited higher error rates, particularly in Hypoglycemia, highlighting the challenge of accurately predicting lower glucose levels. Both **Pipeline 1** and **Pipeline 2** showed marked improvements in all glycemic states, with Pipeline 1 demonstrating superior accuracy in the hyperglycemic range and over other pipelines in all the glycemic ranges. However, **Pipeline 3** displayed inconsistent performance, with the highest error rates in Hypoglycemia and notable variability across ranges. Specific values confirm these observations: for **Pipeline 0**, errors peaked in Hypoglycemia at **6.74**%, **Pipeline 1** reduced errors across clusters, particularly in Euglycemia, achieving as low as **2.67**%, **Pipeline 2** improved Hyperglycemia predictions, with errors as low as **3.74**% and **Pipeline 3** had the least consistent performance, with a Hypoglycemia error of **9.41**%.

**Campus Bio-Medico Dataset** Similar trends were observed in the Campus Bio-Medico Dataset, with errors generally higher in the hypoglycemic and hyperglycemic ranges. Compared to the Ohio dataset, errors in Hypoglycemia were consistently larger, reflecting the increased complexity of this dataset. **Pipeline 0** displayed the highest errors across these critical ranges, whereas **Pipeline 1** and **Pipeline 2** delivered notable improvements, particularly in Hyperglycemia. Nevertheless, **Pipeline 3** again showed inconsistent performance, with the highest error in Hypoglycemia among all pipelines. Supporting values include: **Pipeline 0** exhibited **7.93**% error in Hypoglycemia and **7.30**% in Hyperglycemia, **Pipeline 1** improved significantly,

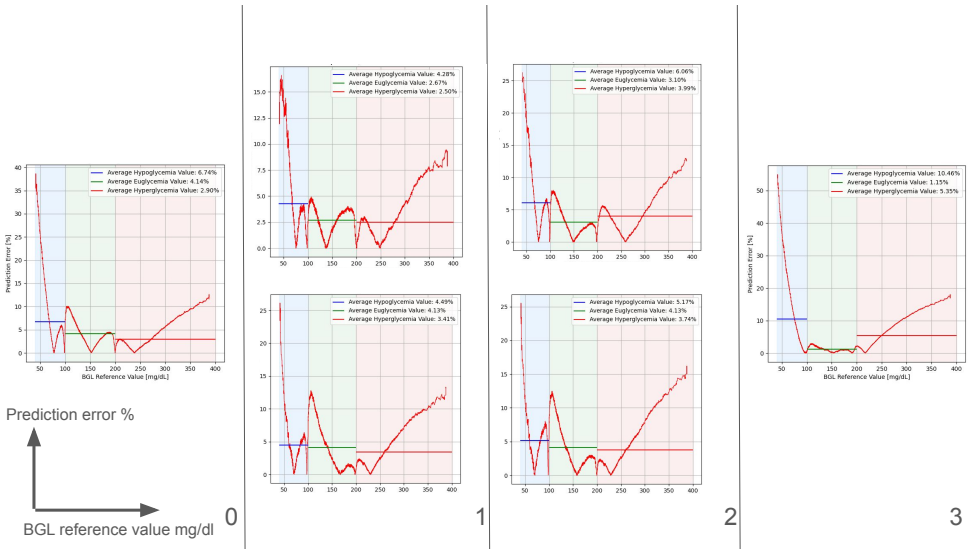


FIGURE 5.28: The data shown in the figure are derived from calculations based on the OHIO dataset. The numbers ranging from 0 to 3 denote the corresponding pipelines represented in the plots. Each graph displays axes that are consistent with the definitions provided in the legend at the bottom left. Specifically, the x-axis represents BGL reference values (in mg/dL), while the y-axis shows the prediction error percentage

particularly in Cluster 2, with errors as low as **1.61**% in Hyperglycemia, **Pipeline 2** maintained lower error rates in Hyperglycemia, achieving **2.71**% in Cluster 2 and **Pipeline 3** showed the highest Hypoglycemia error at **12.39**%.

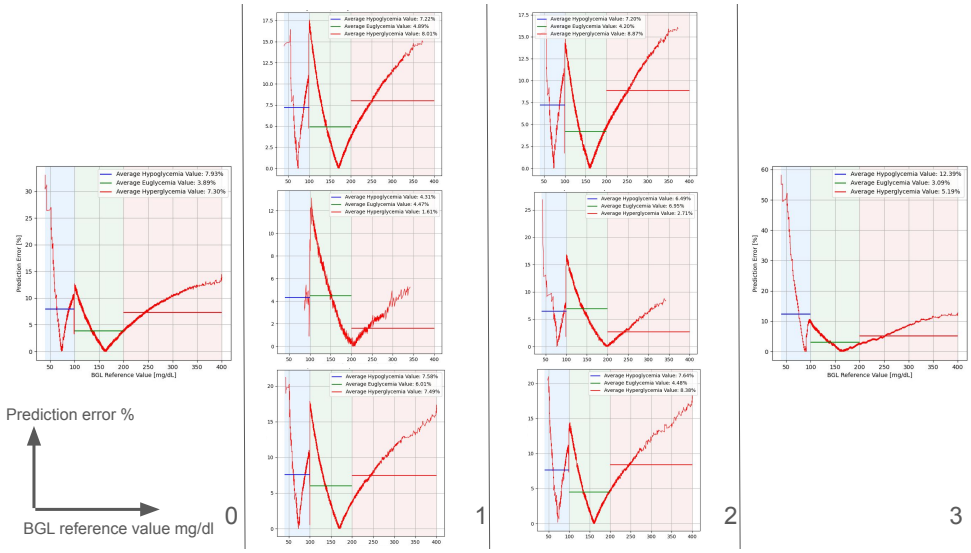


FIGURE 5.29: The data shown in the figure are derived from calculations based on the Campus Biomedico dataset. The numbers ranging from 0 to 3 denote the corresponding pipelines represented in the plots. Each graph displays axes that are consistent with the definitions provided in the legend at the bottom left. Specifically, the x-axis represents BGL reference values (in mg/dL), while the y-axis shows the prediction error percentage

The comparison of percentage errors across pipelines indicates that **Pipeline 2**

generally provides lower errors, particularly in the hyperglycemic range, compared to other pipelines. **Pipeline 1** also shows a good performance, while **Pipeline 0** and **Pipeline 3** show higher errors, especially in the hypoglycemic range.

TABLE 5.19: Percentage Error for Each Glycemic Range, Pipeline, and Dataset

| Pipeline | Dataset | Hypoglycemia (%) | Euglycemia (%) | Hyperglycemia (%) |
|---|---|---|---|---|
| Pipeline 0 | Ohio T1DM | 6.74 | 4.14 | 2.90 |
| Pipeline 0 | Campus Bio-Medico | 7.93 | 3.89 | 7.30 |
| Pipeline 1 | Ohio T1DM (Cluster 1) | 4.28 | 2.67 | 2.50 |
| Pipeline 1 | Ohio T1DM (Cluster 2) | 4.49 | 4.13 | 3.41 |
| Pipeline 1 | Campus Bio-Medico (Cluster 1) | 7.22 | 4.89 | 8.01 |
| Pipeline 1 | Campus Bio-Medico (Cluster 2) | 4.31 | 4.47 | 1.61 |
| Pipeline 1 | Campus Bio-Medico (Cluster 3) | 7.58 | 6.01 | 7.49 |
| Pipeline 2 | Ohio T1DM (Cluster 1) | 6.06 | 3.10 | 3.99 |
| Pipeline 2 | Ohio T1DM (Cluster 2) | 5.17 | 4.13 | 3.74 |
| Pipeline 2 | Campus Bio-Medico (Cluster 1) | 7.20 | 4.20 | 8.87 |
| Pipeline 2 | Campus Bio-Medico (Cluster 2) | 6.49 | 6.95 | 2.71 |
| Pipeline 2 | Campus Bio-Medico (Cluster 3) | 7.64 | 4.48 | 8.38 |
| Pipeline 3 | Ohio T1DM | 9.41 | 0.93 | 5.77 |
| Pipeline 3 | Campus Bio-Medico | 12.39 | 3.09 | 5.19 |

The comparison of percentage errors across pipelines indicates that **Pipeline 2** generally provides lower errors, particularly in the hyperglycemic range, compared to other pipelines. **Pipeline 1** also shows a good performance, while **Pipeline 0** and **Pipeline 3** show higher errors, especially in the hypoglycemic range.

Overall, **Pipeline 2** demonstrates the best clinical safety (as indicated by lower percentage errors in hyperglycemic ranges), making it the most effective approach for the datasets under study. Moreover Pipeline 2 offers the highest robustness and lowest clinical risk, suggesting that its advanced methodologies contribute significantly to improved model performance.

**Comparison with literature** Table 5.20 shows a comparison with the approaches, described in the paragraph *"Approaches for BGL Prediction with Ohio T1DM Dataset"* of section 5.2, in terms of the RMSE achieved after an inference process on the Ohio T1DM test set. [198] did not provide RMSE on the entire test set, so it was excluded from the comparison.

The results obtained by our proposed approach (Pipelines 0-3) show that all pipelines outperform the models selected for comparison, with the exception of the transformer-based model. Notably, Pipelines 1 and 2 surpass the transformer, achieving RMSE values of **11.13** and **10.27**, respectively, demonstrating a significant improvement over the transformer's RMSE of **13.94** [48]. Pipelines 0 (**14.39**) and 3 (**14.61**) also exhibit competitive results, although they fall slightly short of the transformer model. These outcomes emphasize the overall strength of our proposed approach, particularly Pipelines 1 and 2, in advancing predictive accuracy.

The performance of the models reveals distinct clusters in the RMSE results, highlighting the impact of architectural choices. Three main clusters can be identified: the most accurate predictions, with RMSE values around 10–11, represent the pipelines 1 and 2, followed by a middle cluster with RMSE values around 13–14, and finally, a less accurate cluster with RMSE values around 16–19. This separation underscores that simpler or tailored architectures tend to perform better, while added complexity in some cases fails to translate into improved results. For example, simpler RNN architectures consistently underperformed, with RMSE values of **16.81** and **18.87** reported in [32] and [45], respectively. Derivatives or more complex adaptations of RNNs showed no significant advantage, often falling into the least accurate cluster. In contrast, the Transformer model from [48], achieving an RMSE of **13.94**, fits into the middle cluster but still did not surpass the results of Pipeline 2 and Pipeline 1, which achieved RMSE values in the most accurate cluster. These observations highlight the importance of both simplicity and domain-specific optimizations in achieving superior performance.

In conclusion, the results highlight the superior performance of our proposed approach which consistently outperformed other methods, confirming the efficiency and effectiveness of the proposed methodology.

| Model | **RMSE** |
|---|---|
| Pipeline 0 | 14.39 |
| Pipeline 1 | 11.13 |
| Pipeline 2 | **10.27** |
| Pipeline 3 | 14.61 |
| GLuADFL [199] | 19.66 |
| FedGlu [198] | - |
| Linnear+RNN [47](1) | 19.63 |
| Linnear+RNN [47](2) | 19.64 |
| Linnear+RNN [47](3) | 19.62 |
| MLP+RNN [46] | 19.97 |
| RNN [32] | 16.81 |
| RNN [45] | 18.87 |
| Transformer [48] | 13.94 |

TABLE 5.20: Comparison between the proposed approach and models selected from the literature in terms of RMSE reached.

# Chapter 6

# Conclusion

This doctoral thesis has presented a comprehensive exploration of innovative methodologies for developing secure, efficient, and interpretable AI-IoT systems tailored for healthcare applications. By addressing critical challenges in computational efficiency, explainability, and data privacy, the research lays a solid foundation for next-generation solutions capable of advancing Healthcare 5.0. Below, we summarize the key contributions and their implications for the field.

The four contributions presented in the Chapter 3 demonstrate the breadth of applications and the critical importance of developing efficient Neural Network (NN) solutions on edge devices. From an industrial perspective, the ability to obtain *immediate* computational information regarding NNs greatly streamlines production timelines and fosters the rapid testing of multiple algorithms. Optimization techniques such as pruning and quantization have been shown to yield diverse benefits: pruning can significantly reduce Multiply–Accumulate operations (MACs) and inference time, whereas quantization particularly helps conserve memory resources and facilitates the simultaneous deployment of several NNs on the same hardware 3.3.On the hardware acceleration front, the adoption of new VPU technologies 3.5 exemplifies a transformative leap in performance: classification times were reduced by up to 84.5% using the NCS and approximately 90% with the NCS2, which aligns with [250] and underscores the viability of state-of-the-art edge solutions. Notably, these devices retain modest cost and dimensions, making them particularly suitable for IoT scenarios wherein size, portability, and energy efficiency are paramount. Beyond computational efficiency, these works highlight cross-disciplinary impacts, as evidenced by advanced frameworks employed in diverse tasks: from real-time microplastic detection in fluids 3.6 (aimed at fulfilling the 17 Sustainable Development Goals) to blood glucose prediction in pediatric Type 1 Diabetes patients 3.4. In the latter domain, the CNN and LSTM models tested on various devices demonstrated numerical accuracy comparable to adult patients, representing, to the best of our knowledge, the first edge-computing system specifically targeting glucose concentration forecasting for children. Meanwhile, for microplastic detection, the proposed NeuralCasting approach and its native C code compilation have proven more efficient than mainstream solutions such as ONNX Runtime, offering improvements in latency, memory footprint, energy consumption, and portability.

In the Chapter 4, we have explored diverse approaches across multiple domains, demonstrating advancements in both methodology and application. The feature-based knowledge distillation approach outlined in section 4.5 demonstrated its capability to enhance the explainability and efficiency of AI models for pulmonary disease classification. By effectively transferring knowledge from complex models to lightweight architectures, this methodology underscored the importance of optimizing models for limited data scenarios and paved the way for further research into feature alignment techniques and their generalizability across datasets. The

implementation of a glycemic predictor on edge devices, section 4.4, emphasized the trade-off between model simplicity and performance. The results showed that optimized architectures could achieve comparable accuracy to more complex models while ensuring practicality and scalability, moreover, the explainability analysis made through LIME enhances the interpretability of the model for the user. These findings form the basis for developing real-life decision support systems, offering both numeric and clinical reliability for glycemic management. Furthermore, the Tandem model presented significant improvements in segmentation tasks, particularly for small and medium-sized objects, as highlighted in section 4.3. The ability to deliver precise confidence maps and robust performance across datasets reinforces the model's potential to enhance medical imaging analysis and aid clinical decision-making processes.

In Chapter 5, we presented four different works, each addressing specific challenges in secure and efficient data management across Federated Learning (FL), Internet of Things (IoT), and biomedical scenarios. First, as reported in Section 5.4 the proposed network architecture integrates Zero-Knowledge Proofs (ZKPs) and Distributed Ledger Technology (DLT) into the FL paradigm to ensure security, verifiability, and reliability of exchanged data. The overheads analysis shown that the incorporation of SNARKs into the FL paradigm, albeit introducing significant computational and temporal overheads, strategically capitalizes on the resources at the peer level to safeguard integrity and veracity within the learning process. Crucially, it avoids overwhelming the aggregator, thereby maintaining operational scalability. Cost considerations, driven by the DLT fee policy, highlight the importance of carefully selecting or customizing the underlying ledger technology, especially when many peers participate in the FL process. Second, the dRAIN architecture described in Section 5.3 combines a hardware root of trust with DLT for secure, reliable, and auditable data management. The proof-of-concept showed acceptable cryptographic overhead, ensuring robust security guarantees. However, the approval times intrinsic to Tangle-based transactions introduce variability and potentially slow confirmations. This trade-off underscores the wider theme of balancing strong security properties with the performance requirements of real-world IoT applications. Nonetheless, the system's adaptability to different devices and networks demonstrates promising potential for broader deployments. In the biomedical domain, the triple regressor FedROS-ELM solution presented in the section 5.5 was employed for Blood Glucose Level (BGL) forecasting. By leveraging a federated environment, this method achieves high predictive accuracy with efficient learning, making it suitable for edge devices commonly used in clinical practice. The utilization of multiple sub-regressors specialized for distinct glycemic states preserves the model's generalization capabilities while maintaining rapid training speeds. Clinically reliable predictions, validated through CEG analysis, confirm the viability of this approach in real-world healthcare contexts, where both computational efficiency and patient data privacy are crucial. Lastly, the novel Graph-aided Federated Learning (GaFL) framework summarized in Section 5.6 emphasizes patient similarity modeling for personalized glycemic level prediction. By employing a triple-regressor ROS-ELM approach adapted to various glycemic conditions, the framework demonstrates lower RMSE and improved accuracy on the Ohio T1DM and Campus Bio-Medico datasets. The graph-based clustering not only enhances predictive performance but also protects patient data via federated training. This strategy further highlights the importance of decentralized, privacy-preserving methodologies in modern healthcare solutions.

The findings and methodologies presented in this thesis pave the way for transformative advancements in Healthcare 5.0. By demonstrating the practical feasibility of secure, interpretable, and efficient AI-IoT systems, this research encourages the broader adoption of intelligent technologies across various medical and industrial domains. Future work could explore:

- Expanding the scalability of federated learning frameworks to support more diverse and heterogeneous IoT networks.
- Enhancing interpretability techniques to encompass multi-modal data and more complex decision-making processes.

In conclusion, this thesis underscores the transformative potential of AI-IoT integration for healthcare. By addressing the interplay between computational efficiency, interpretability, and privacy, the work contributes to the development of trustworthy, sustainable, and impactful solutions that align with the goals of modern medicine and technology.

# Bibliography

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenet," 2017, arXiv:1704.04861.

[2] Y. Brima, M. Atemkeng, S. Tankio Djiokap, J. Ebiele, and F. Tchakounté, "Transfer learning for the detection and diagnosis of types of pneumonia including pneumonia induced by covid-19 from chest x-ray images," *Diagnostics*, vol. 11, no. 8, p. 1480, 2021.

[3] P. Afferni, M. Merone, and P. Soda, "Hospital 4.0 and its innovation in methodologies and technologies," in *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, 2018, pp. 333–338.

[4] A. Al Kuwaiti, K. Nazer, A. Al-Reedy, S. Al-Shehri, A. Al-Muhanna, A. V. Subbarayalu, D. Al Muhanna, and F. A. Al-Muhanna, "A review of the role of artificial intelligence in healthcare," *Journal of personalized medicine*, vol. 13, no. 6, p. 951, 2023.

[5] P. P. Ray, D. Dash, and D. De, "Edge computing for internet of things: A survey, e-healthcare case study and future direction," *Journal of Network and Computer Applications*, vol. 140, pp. 1–22, 2019.

[6] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[7] B. Sun, S. Bayes, A. M. Abotaleb, and M. Hassan, "The case for tinyml in healthcare: Cnns for real-time on-edge blood pressure estimation," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023, pp. 629–638.

[8] Y. He and L. Xiao, "Structured pruning for deep convolutional neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2023.

[9] L. Wei, Z. Ma, C. Yang, and Q. Yao, "Advances in the neural network quantization: A comprehensive review," *Applied Sciences*, vol. 14, no. 17, p. 7445, 2024.

[10] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626.

[11] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2921–2929.

[12] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[13] A. Maccaro, K. Stokes, L. Statham, L. He, A. Williams, L. Pecchia, and D. Piaggio, "Clearing the fog: a scoping literature review on the ethical issues surrounding artificial intelligence-based medical devices," *Journal of Personalized Medicine*, vol. 14, no. 5, p. 443, 2024.

[14] S.-R. Oh, Y.-D. Seo, E. Lee, and Y.-G. Kim, "A comprehensive survey on security and privacy for electronic health data," *International Journal of Environmental Research and Public Health*, vol. 18, no. 18, p. 9668, 2021.

[15] W. N. Price and I. G. Cohen, "Privacy in the age of medical big data," *Nature medicine*, vol. 25, no. 1, pp. 37–43, 2019.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[17] R. S. Antunes, C. André da Costa, A. Küderle, I. A. Yari, and B. Eskofier, "Federated learning for healthcare: Systematic review and architecture proposal," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1–23, 2022.

[18] U. Fiege, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 210–217.

[19] S. Pandey, B. Bhushan, and A. A. Hameed, "Securing healthcare 5.0: Zero-knowledge proof (zkp) and post quantum cryptography (pqc) solutions for medical data security," in *Soft Computing in Industry 5.0 for Sustainability*. Springer, 2024, pp. 339–355.

[20] S. Nakamoto, "Bitcoin whitepaper," *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, vol. 9, p. 15, 2008.

[21] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[22] I. D. Federation, "IDF Diabetes Atlas, 10th edn. Brussels, Belgium: 2021. Available at: https://www.diabetesatlas.org," 2021.

[23] L. Reid, F. Baxter, and S. Forbes, "Effects of islet transplantation on microvascular and macrovascular complications in type 1 diabetes," *Diabetic Medicine*, vol. 38, no. 7, p. e14570, 2021.

[24] e. a. J. T. Warshauer, "New Frontiers in the Treatment of Type 1 Diabetes," *Cell Metabolism*, vol. 31, no. 1, pp. 46–61, 2020.

[25] e. a. B. J. von Scholten, "Current and future therapies for type 1 diabetes." *Diabetologia*, vol. 64, no. 5, pp. 1037–1048, 2021.

[26] e. a. Wu Zhixian, "Prevention of chronic diabetic complications in type 1 diabetes by co-transplantation of umbilical cord mesenchymal stromal cells and autologous bone marrow: a pilot randomized controlled open-label clinical study with 8-year follow-up." *Cytotherapy*, vol. 24, no. 4, pp. 421–427, 2022.

[27] e. a. T. Danne, "International Consensus on Use of Continuous Glucose Monitoring." *Diabetes Care*, vol. 40, no. 12, pp. 1631–1640, 2017.

[28] e. a. S. Charleer, "Effect of Continuous Glucose Monitoring on Glycemic Control, Acute Admissions, and Quality of Life: A Real-World Study." *The Journal of Clinical Endocrinology and Metabolism*, vol. 103, no. 3, pp. 1224–1232, 2018.

[29] E. Toschi, A. Michals, A. Adam, D. Davis, A. Atakov-Castillo, C. Slyne, and M. Munshi, "Usefulness of cgm-derived metric, the glucose management indicator, to assess glycemic control in non-white individuals with diabetes," *Diabetes Care*, vol. 44, no. 12, pp. 2787–2789, 2021.

[30] e. a. O. Mujahid, "Machine Learning Techniques for Hypoglycemia Prediction: Trends and Challenges." *Sensors (Basel, Switzerland)*, vol. 21, no. 2, p. 546, 2021.

[31] e. a. H. Rubin-Falcone, "Forecasting with Sparse but Informative Variables: A Case Study in Predicting Blood Glucose." arXiv preprint arXiv:2304.08593, 2023.

[32] e. a. H. Butt, "Feature Transformation for Efficient Blood Glucose Prediction in Type 1 Diabetes Mellitus Patients." *Diagnostics*, vol. 13, no. 3, p. article 340, 2023.

[33] V. Piemonte and other, "A novel three-compartmental model for artificial pancreas: development and validation," *Artificial organs*, vol. 41, no. 12, pp. E326–E336, 2017.

[34] S. M. Lynch and B. W. Bequette, "Estimation-based model predictive control of blood glucose in type i diabetics: a simulation study," in *Proceedings of the IEEE 27th Annual Northeast Bioengineering Conference (Cat. No. 01CH37201)*. IEEE, 2001, pp. 79–80.

[35] S. Oviedo *et al.*, "A review of personalized blood glucose prediction strategies for T1DM patients," *International journal for numerical methods in biomedical engineering*, vol. 33, no. 6, p. e2833, 2017.

[36] B. De Paoli, F. D'Antoni, M. Merone, S. Pieralice, V. Piemonte, and P. Pozzilli, "Blood glucose level forecasting on type-1-diabetes subjects during physical activity: A comparative analysis of different learning techniques," *Bioengineering*, vol. 8, no. 6, p. 72, 2021.

[37] R. Bunescu, N. Struble, C. Marling, J. Shubrook, and F. Schwartz, "Blood glucose level prediction using physiological models and support vector regression," in *12th Int. Conf. on Machine Learning and Applications*, vol. 1. IEEE, 2013, pp. 135–140.

[38] E. I. Georga, V. C. Protopappas, D. Polyzos, and D. I. Fotiadis, "A predictive model of subcutaneous glucose concentration in type 1 diabetes based on random forests," in *Int. Conf. of the IEEE Eng. in Medicine and Biology Society*, 2012, pp. 2889–2892.

[39] G. Sparacino, F. Zanderigo, S. Corazza, A. Maran, A. Facchinetti, and C. Cobelli, "Glucose concentration can be predicted ahead in time from continuous glucose monitoring sensor time-series," *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 5, pp. 931–937, 2007.

[40] F. D'Antoni, M. Merone, V. Piemonte, G. Iannello, and P. Soda, "Autoregressive time delayed jump neural network for blood glucose levels forecasting," *Knowledge-Based Systems*, p. 106134, 2020.

[41] K. Li, C. Liu, T. Zhu, P. Herrero, and P. Georgiou, "Glunet: A deep learning framework for accurate glucose forecasting," *IEEE journal of biomedical and health informatics, 2019*, 2019.

[42] C. Mosquera-Lopez, R. Dodier, N. Tyler, N. Resalat, and P. Jacobs, "Leveraging a big dataset to develop a recurrent neural network to predict adverse glycemic events in type 1 diabetes," *IEEE Journal of Biomedical and Health Informatics*, 2019.

[43] K. Li, J. Daniels, C. Liu, P. Herrero, and P. Georgiou, "Convolutional recurrent neural networks for glucose prediction," *IEEE journal of biomedical and health informatics*, vol. 24, no. 2, pp. 603–613, 2019.

[44] R. Visentin, E. Campos-Náñez, M. Schiavon, D. Lv, M. Vettoretti, M. Breton, B. P. Kovatchev, C. Dalla Man, and C. Cobelli, "The uva/padova type 1 diabetes simulator goes from single meal to single day," *Journal of diabetes science and technology*, vol. 12, no. 2, pp. 273–281, 2018.

[45] J. Martinsson, A. Schliep, B. Eliasson, and O. Mogren, "Blood glucose prediction with variance estimation using recurrent neural networks," *Journal of Healthcare Informatics Research*, vol. 4, pp. 1–18, 2020.

[46] H. Khadem, H. Nemat, J. Elliott, and M. Benaissa, "Blood glucose level time series forecasting: nested deep ensemble learning lag fusion," *Bioengineering*, vol. 10, no. 4, p. 487, 2023.

[47] H. Nemat, H. Khadem, M. R. Eissa, J. Elliott, and M. Benaissa, "Blood glucose level prediction: advanced deep-ensemble learning approach," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 6, pp. 2758–2769, 2022.

[48] E. Acuna, R. Aparicio, and V. Palomino, "Analyzing the performance of transformers for the prediction of the blood glucose level considering imputation and smoothing," *Big Data and Cognitive Computing*, vol. 7, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2504-2289/7/1/41

[49] G. Muhammad and M. Hossain, "Emotion recognition for cognitive edge computing using deep learning," *IEEE Internet Things J.*, vol. 8, pp. 16 894–16 901, 2021.

[50] M. Chen, Y. Hao, L. Hu, M. Hossain, and A. Ghoneim, "Edge-cocaco: Toward joint optimization of computation, caching, and communication on edge cloud," *IEEE Wirel. Commun.*, vol. 25, pp. 21–27, 2018.

[51] C. Banbury, V. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, and et al., "Benchmarking tinyml systems: Challenges and direction," 2020.

[52] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Ambient Assisted Living and Home Care*, J. Bravo, R. Hervás, and M. Rodríguez, Eds. Berlin/Heidelberg, Germany: Springer, 2012, pp. 216–223.

[53] S. Moon, M. Min, J. Nam, J. Park, D. Lee, and D. Kim, "Drowsy driving warning system based on gs1 standards with machine learning," in *Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, HI, USA, 2017, pp. 289–296.

[54] Y. Nair, S. Kumar, and K. Soman, "Real-time automotive engine fault detection and analysis using bigdata platforms," in *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*, S. Satapathy, V. Bhateja, S. Udgata, and P. Pattnaik, Eds. Bhubaneswar, India: Springer, 2017, pp. 507–514.

[55] A. Luckow, K. Kennedy, M. Ziolkowski, E. Djerekarov, M. Cook, E. Duffy, M. Schleiss, B. Vorster, E. Weill, A. Kulshrestha, and et al., "Artificial intelligence and deep learning applications for automotive manufacturing," in *Proceedings of the 2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 3144–3152.

[56] J. Queralta, L. Qingqing, Z. Zou, and T. Westerlund, "Enhancing autonomy with blockchain and multi-access edge computing in distributed robotic systems," in *Proceedings of the 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, Paris, France, 2020, pp. 180–187.

[57] R. David, J. Duke, A. Jain, V. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, and et al., "Tensorflow lite micro: Embedded machine learning on tinyml systems," 2020.

[58] Y. Ran, Q. Zheng, R. Chellappa, and T. Strat, "Applications of a simple characterization of human gait in surveillance," *IEEE Trans. Syst. Man, Cybern. Part B (Cybern.)*, vol. 40, pp. 1009–1020, 2010.

[59] J. Candamo, M. Shreve, D. Goldgof, D. Sapper, and R. Kasturi, "Understanding transit scenes: A survey on human behavior-recognition algorithms," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, pp. 206–224, 2010.

[60] M. Waqas, S. Tu, Z. Halim, S. Rehman, G. Abbas, and Z. Abbas, "The role of artificial intelligence and machine learning in wireless networks security: Principle, practice and challenges," *Artif. Intell. Rev.*, vol. 55, pp. 1–47, 2022.

[61] J. Shuja, M. Humayun, W. Alasmary, H. Sinky, E. Alanazi, and M. Khan, "Resource efficient geo-textual hierarchical clustering framework for social iot applications," *IEEE Sens. J.*, vol. 21, pp. 25 114–25 122, 2021.

[62] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions," *ACM Comput. Surv.*, vol. 53, pp. 1–37, 2020.

[63] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*. Montreal, QC, Canada: MIT Press, 2015, pp. 1135–1143.

[64] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Proceedings of the Advances in Neural Information Processing Systems*, Barcelona, Spain, 2016.

[65] G. Li, C. Qian, C. Jiang, X. Lu, and K. Tang, "Optimization based layer-wise magnitude-based pruning for dnn compression," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, Stockholm, Sweden, 2018, pp. 2383–2389.

[66] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, 2017, pp. 2755–2763.

[67] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, 2017, pp. 5068–5076.

[68] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Red Hook, NY, USA: Curran Associates, Inc., 2015.

[69] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proceedings of the European Conference on Computer Vision*, Munich, Germany, 2016.

[70] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016.

[71] B. Sau and V. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," 2016.

[72] A. Romero, N. Ballas, S. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," 2015.

[73] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[74] F. Iandola, M. Moskewicz, K. Ashraf, S. Han, W. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1 mb model size," 2016.

[75] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, pp. 1943–1955, 2016.

[76] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, Montreal, QC, Canada, 2014.

[77] A. R. Nasser, A. M. Hasan, A. J. Humaidi, A. Alkhayyat, L. Alzubaidi, M. A. Fadhel, J. Santamaría, and Y. Duan, "Iot and cloud computing in health-care: A new wearable device and cloud-based deep learning algorithm for monitoring of diabetes," *Electronics*, vol. 10, no. 21, p. 2719, 2021.

[78] G. M. Bhat and N. G. Bhat, "A novel iot based framework for blood glucose examination," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. IEEE, 2017, pp. 205–207.

[79] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[80] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[81] W. R. Hersh, M. Helfand, J. Wallace, D. Kraemer, P. Patterson, S. Shapiro, and M. Greenlick, "Clinical outcomes resulting from telemedicine interventions: a systematic review," *BMC Medical Informatics and Decision Making*, vol. 1, no. 1, pp. 1–8, 2001.

[82] C. Scott Kruse, P. Karem, K. Shifflett, L. Vegi, K. Ravi, and M. Brooks, "Evaluating barriers to adopting telemedicine worldwide: a systematic review," *Journal of telemedicine and telecare*, vol. 24, no. 1, pp. 4–12, 2018.

[83] D. Sasso, M. Merone, E. Nicolai, L. Vollero, and A. Sabatini, "A benchmarking on optofluidic microplastic pattern recognition: A systematic comparison between statistical detection models and ml-based algorithms," *IEEE Access*, 2024.

[84] K. M. Dungan, C. Sagrilla, M. Abdel-Rasoul, and K. Osei, "Prandial insulin dosing using the carbohydrate counting technique in hospitalized patients with type 2 diabetes," *Diabetes Care*, vol. 36, no. 11, pp. 3476–3482, 2013.

[85] H. Zisser, L. Robinson, W. Bevier, E. Dassau, C. Ellingsen, F. J. Doyle III, and L. Jovanovic, "Bolus calculator: a review of four "smart" insulin pumps," *Diabetes technology & therapeutics*, vol. 10, no. 6, pp. 441–444, 2008.

[86] C. Toffanin, H. Zisser, F. J. Doyle III, and E. Dassau, "Dynamic insulin on board: incorporation of circadian insulin sensitivity variation," *Journal of diabetes science and technology*, vol. 7, no. 4, pp. 928–940, 2013.

[87] M. Merone, A. Graziosi, V. Lapadula, L. Petrosino, O. d'Angelis, and L. Vollero, "A practical approach to the analysis and optimization of neural networks on embedded systems," *Sensors*, vol. 22, no. 20, p. 7807, 2022.

[88] A. Chan and N. Vasconcelos, "Bayesian poisson regression for crowd counting," in *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision*, Kyoto, Japan, 2009, pp. 545–551.

[89] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, pp. 1137–1149, 2015.

[90] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision*, Munich, Germany, 2016, pp. 8–14 September.

[91] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.

[92] V. Pham, T. Kozakaya, O. Yamaguchi, and R. Okada, "Count forest: Co-voting uncertain number of targets using random forest for crowd density estimation," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, 2015, pp. 3253–3261.

[93] G. Gao, J. Gao, Q. Liu, Q. Wang, and Y. Wang, "Cnn-based density estimation and crowd counting: A survey," 2020.

[94] H. Bai and S. Chan, "Cnn-based single image crowd counting: Network design, loss function and supervisory signal," 2020.

[95] A. Graziosi, G. Iannello, V. Lapadula, M. Merone, M. Sabatini, and L. Vollero, "Edge computing optimization method: Analyzed task–crowd counting," in *Proceedings of the 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT*, Rome, Italy, 2021, pp. 7–9 June.

[96] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, 2017.

[97] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," https://arxiv.org/abs/1409.1556, 2015, arXiv:1409.1556.

[98] R. Reed and R. MarksII, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA: MIT Press, 1999.

[99] M. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016, pp. 1135–1144.

[100] T. Danne, T. Battelino, P. Jarosz-Chobot, O. Kordonouri, E. Pánkowska, J. Ludvigsson, E. Schober, E. Kaprio, T. Saukkonen, M. Nicolino *et al.*, "Establishing glycaemic control with continuous subcutaneous insulin infusion in children and adolescents with type 1 diabetes: experience of the pedpump study in 17 countries," *Diabetologia*, vol. 51, no. 9, pp. 1594–1601, 2008.

[101] C. Worth, M. Dunne, A. Ghosh, S. Harper, and I. Banerjee, "Continuous glucose monitoring for hypoglycaemia in children: perspectives in 2020," *Pediatric Diabetes*, vol. 21, no. 5, pp. 697–706, 2020.

[102] S. G. Mougiakakou, A. Prountzou, D. Iliopoulou, K. S. Nikita, A. Vazeou, and C. S. Bartsocas, "Neural network based glucose-insulin metabolism models for children with type 1 diabetes," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2006, pp. 3545–3548.

[103] E. Dassau, F. Cameron, H. Lee, B. W. Bequette, H. Zisser, L. Jovanovič, H. P. Chase, D. M. Wilson, B. A. Buckingham, and F. J. Doyle III, "Real-time hypoglycemia prediction suite using continuous glucose monitoring: a safety net for the artificial pancreas," *Diabetes care*, vol. 33, no. 6, pp. 1249–1254, 2010.

[104] M. De Bois, M. A. El Yacoubi, and M. Ammi, "Study of short-term personalized glucose predictive models on type-1 diabetic children," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[105] W. L. Clarke, "The original clarke error grid analysis (ega)," *Diabetes technology & therapeutics*, vol. 7, no. 5, pp. 776–779, 2005.

[106] F. D'Antoni, L. Petrosino, F. Sgarro, A. Pagano, L. Vollero, V. Piemonte, and M. Merone, "Prediction of glucose concentration in children with type 1 diabetes using neural networks: An edge computing application," *Bioengineering*, vol. 9, no. 5, p. 183, 2022.

[107] T. E. Idrissi and A. Idri, "Deep learning for blood glucose prediction: Cnn vs lstm," in *International Conference on Computational Science and Its Applications*. Springer, 2020, pp. 379–393.

[108] TensorFlow, "Post-training quantization," *https://www.tensorflow.org/lite/performance/post_training_quantization*, 2022.

[109] Google, "Frequently asked questions," *available online at: https://coral.ai/docs/edgetpu/faq/how-is-the-edge-tpu-different-from-cloud-tpus*, 2022.

[110] L. Petrosino, G. Iannello, M. Merone, and L. Vollero, "Image sensors and vpu acceleration for data analysis and classification," in *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT)*. IEEE, 2021, pp. 392–396.

[111] I. Corporation, "Openvino toolkit documentation," 2020, https://docs.openvinotoolkit.org/latest/documentation.html.

[112] L. Lebreton, M. Egger, and B. Slat, "A global mass budget for positively buoyant macroplastic debris in the ocean," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[113] J. N. Hahladakis, C. A. Velis, R. Weber, E. Iacovidou, and P. Purnell, "An overview of chemical additives present in plastics: Migration, release, fate and environmental impact during their use, disposal and recycling," *Journal of hazardous materials*, vol. 344, pp. 179–199, 2018.

[114] M. Cole, P. Lindeque, C. Halsband, and T. S. Galloway, "Microplastics as contaminants in the marine environment: a review," *Marine pollution bulletin*, vol. 62, no. 12, pp. 2588–2597, 2011.

[115] United Nations, "The 17 goals - sustainable development," 2024, accessed: 2024-07-11. [Online]. Available: https://sdgs.un.org/goals

[116] A. Baruah, A. Sharma, S. Sharma, and R. Nagraik, "An insight into different microplastic detection methods," *International Journal of Environmental Science and Technology*, vol. 19, no. 6, pp. 5721–5730, 2022.

[117] M. Kedzierski, M. Falcou-Préfol, M. E. Kerros, M. Henry, M. L. Pedrotti, and S. Bruzaud, "A machine learning algorithm for high throughput identification of ftir spectra: Application on microplastics collected in the mediterranean sea," *Chemosphere*, vol. 234, pp. 242–251, 2019.

[118] V. H. da Silva, F. Murphy, J. M. Amigo, C. Stedmon, and J. Strand, "Classification and quantification of microplastic (< 100 $\mu$m) using fpa-ftir imaging system and machine learning," *Analytical Chemistry*, vol. 92, no. 20, pp. 13 724–13 733, 2020.

[119] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022.

[120] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, 2023.

[121] V. Tsoukas, E. Boumpa, G. Giannakas, and A. Kakarountas, "A review of machine learning and tinyml in healthcare," in *Proceedings of the 25th Pan-Hellenic Conference on Informatics*, 2021, pp. 69–73.

[122] Y. Y. Siang, M. R. Ahamd, and M. S. Z. Abidin, "Anomaly detection based on tiny machine learning: A review," *Open International Journal of Informatics*, vol. 9, no. Special Issue 2, pp. 67–78, 2021.

[123] T. Flores, M. Silva, P. Andrade, J. Silva, I. Silva, E. Sisinni, P. Ferrari, and S. Rinaldi, "A tinyml soft-sensor for the internet of intelligent vehicles," in *2022 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)*. IEEE, 2022, pp. 18–23.

[124] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[125] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, p. 1789–1819, Mar. 2021. [Online]. Available: http://dx.doi.org/10.1007/s11263-021-01453-z

[126] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[127] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf

[128] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," 2015.

[129] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," 2017.

[130] S. You, C. Xu, C. Xu, and D. Tao, "Learning from multiple teacher networks," in ., ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1294. [Online]. Available: https://doi.org/10.1145/3097983.3098135

[131] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distillation," 2019.

[132] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," 2015.

[133] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that?" 2017.

[134] R. Alharbi, M. N. Vu, and M. T. Thai, "Learning interpretation with explainable knowledge distillation," 2021.

[135] J. Rao, X. Meng, L. Ding, S. Qi, X. Liu, M. Zhang, and D. Tao, "Parameter-efficient and student-friendly knowledge distillation," *IEEE Transactions on Multimedia*, 2023.

[136] X. Sha, Y. Wang, and T. Fang, "Decoupled knowledge distillation in data-free federated learning," in *International Artificial Intelligence Conference*. Springer, 2023, pp. 164–177.

[137] H. Kim, T.-Y. Kwak, H. Chang, S. W. Kim, and I. Kim, "Rckd: Response-based cross-task knowledge distillation for pathological image analysis," *Bioengineering*, vol. 10, no. 11, p. 1279, 2023.

[138] M. Frasca, D. La Torre, G. Pravettoni, and I. Cutica, "Explainable and interpretable artificial intelligence in medicine: a systematic bibliometric review," *Discover Artificial Intelligence*, vol. 4, no. 1, p. 15, 2024.

[139] C. Marling and R. Bunescu, "The OhioT1DM dataset for blood glucose level prediction," in *3rd International Workshop on Knowledge Discovery in Healthcare Data at IJCAI-ECAI*, 2018, pp. 60–63. [Online]. Available: http://ceur-ws.org/Vol-2148/paper09.pdf

[140] ——, "The ohiot1dm dataset for blood glucose level prediction: Update 2020," *5th International Workshop on Knowledge Discovery in Healthcare Data, ECAI*, 2020. [Online]. Available: http://smarthealth.cs.ohio.edu/bglp/OhioT1DM-dataset-paper.pdf

[141] S. Monaco, N. Bussola, S. Buttò, D. Sona, F. Giobergia, G. Jurman, C. Xinaris, and D. Apiletti, "Ai models for automated segmentation of engineered polycystic kidney tubules," *Scientific Reports*, vol. 14, no. 1, p. 2847, 2024.

[142] T. Heimann, B. Van Ginneken, M. A. Styner, Y. Arzhaeva, V. Aurich, C. Bauer, A. Beck, C. Becker, R. Beichel, G. Bekes *et al.*, "Comparison and evaluation of methods for liver segmentation from ct datasets," *IEEE transactions on medical imaging*, vol. 28, no. 8, pp. 1251–1265, 2009.

[143] B. Shirokikh, A. Shevtsov, A. Kurmukov, A. Dalechina, E. Krivov, V. Kostjuchenko, A. Golanov, and M. Belyaev, "Universal loss reweighting to balance lesion size inequality in 3d medical image segmentation," in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part IV 23*, Springer. Lima, Peru: Springer, 2020, pp. 523–532.

[144] T. Rahman, A. Khandakar, Y. Qiblawey, A. Tahir, S. Kiranyaz, S. B. Abul Kashem, M. T. Islam, S. Al Maadeed, S. M. Zughaier, M. S. Khan, and M. E. Chowdhury, "Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images," *Computers in Biology and Medicine*, vol. 132, p. 104319, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S001048252100113X

[145] M. E. H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M. A. Kadir, Z. B. Mahbub, K. R. Islam, M. S. Khan, A. Iqbal, N. A. Emadi, M. B. I. Reaz, and M. T. Islam, "Can ai help in screening viral and covid-19 pneumonia?" *IEEE Access*, vol. 8, pp. 132 665–132 676, 2020.

[146] R. Visentin, E. Campos-Náñez, M. Schiavon, D. Lv, M. Vettoretti, M. Breton, B. P. Kovatchev, C. Dalla Man, and C. Cobelli, "The uva/padova type 1 diabetes simulator goes from single meal to single day," *Journal of diabetes science and technology*, vol. 12, no. 2, pp. 273–281, 2018.

[147] S. J. MacEachern and N. D. Forkert, "Machine learning for precision medicine," *Genome*, vol. 64, no. 4, pp. 416–425, 2021.

[148] S. Monaco, L. Petrosino, C. Xinaris, and D. Apiletti, "Tandem: a confidence-based approach for precise medical image segmentation," in *Artificial Intelligence and Data Science for Healthcare: Bridging Data-Centric AI and People-Centric Healthcare*, 2024.

[149] *Cyst segmentation on kidney tubules by means of U-Net deep-learning models*, IEEE. IEEE, 2021.

[150] P. Bilic, P. Christ, H. B. Li, E. Vorontsov, A. Ben-Cohen, G. Kaissis, A. Szeskin, C. Jacobs, G. E. H. Mamani, G. Chartrand *et al.*, "The liver tumor segmentation benchmark (lits)," *Medical Image Analysis*, vol. 84, p. 102680, 2023.

[151] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. .: IEEE, 2018, pp. 6546–6555.

[152] M. Wadghiri, A. Idri, T. El Idrissi, and H. Hakkoum, "Ensemble blood glucose prediction in diabetes mellitus: A review," *Computers in Biology and Medicine*, vol. 147, p. 105674, 2022.

[153] M. Xu, W. C. Ng, W. Y. B. Lim, J. Kang, Z. Xiong, D. Niyato, Q. Yang, X. Shen, and C. Miao, "A full dive into realizing the edge-enabled metaverse: Visions, enabling technologies, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 656–700, 2022.

[154] I. Longo, F. D'Antoni, L. Petrosino, V. Piemonte, M. Merone, and L. Pecchia, "Development of an explainable deep learning-based decision support system for blood glucose levels forecasting in type 1 diabetes using edge computing," in *European Medical and Biological Engineering Conference*. Springer, 2024, pp. 316–326.

[155] S. Del Giorno, F. D'Antoni, V. Piemonte, and M. Merone, "A new glycemic closed-loop control based on dyna-q for type-1-diabetes," *Biomedical Signal Processing and Control*, vol. 81, p. 104492, 2023.

[156] J. R. Lupton, J. Brooks, N. Butte, B. Caballero, J. Flatt, S. Fried *et al.*, "Dietary reference intakes for energy, carbohydrate, fiber, fat, fatty acids, cholesterol, protein, and amino acids," *National Academy Press: Washington, DC, USA*, vol. 5, pp. 589–768, 2002.

[157] C. Zecchin, A. Facchinetti, G. Sparacino, G. De Nicolao, and C. Cobelli, "Neural network incorporating meal information improves accuracy of short-time prediction of glucose concentration," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 6, pp. 1550–1560, 2012.

[158] Arduino. [Online]. Available: https://store.arduino.cc/products/portenta-h7

[159] A. Kurniawan and A. Kurniawan, "Bluetooth low energy (ble)," *Beginning Arduino Nano 33 IoT: Step-By-Step Internet of Things Projects*, pp. 157–181, 2021.

[160] A. Ahmed, S. Aziz, A. Abd-Alrazaq, F. Farooq, M. Househ, and J. Sheikh, "The effectiveness of wearable devices using artificial intelligence for blood glucose level forecasting or prediction: Systematic review," *Journal of Medical Internet Research*, vol. 25, p. e40259, 2023.

[161] J. A. Dendy, V. Chockalingam, N. N. Tirumalasetty, A. Dornelles, L. Blonde, P. M. Bolton, R. Y. Meadows, and S. S. Andrews, "Identifying risk factors for severe hypoglycemia in hospitalized patients with diabetes," *Endocrine Practice*, vol. 20, no. 10, pp. 1051–1056, 2014.

[162] G. Freckmann, "Basics and use of continuous glucose monitoring (cgm) in diabetes therapy," *Journal of Laboratory Medicine*, vol. 44, no. 2, pp. 71–79, 2020.

[163] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "Xai—explainable artificial intelligence," *Science robotics*, vol. 4, no. 37, p. eaay7120, 2019.

[164] T. Sun, H. Chen, G. Hu, and C. Zhao, "Explainability-based knowledge distillation," *Pattern Recognition*, vol. 159, p. 111095, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S003132032400846X

[165] Z. Guo, H. Yan, H. Li, and X. Lin, "Class attention transfer based knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 11 868–11 877.

[166] A. Parchami-Araghi, M. Böhle, S. Rao, and B. Schiele, "Good teachers explain: Explanation-enhanced knowledge distillation," *arXiv preprint arXiv:2402.03119*, 2024.

[167] S. Chowdhury, B. Liang, A. Tizghadam, and I. Albanese, "Improving knowledge distillation with teacher's explanation," 2023.

[168] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[169] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[170] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *2015 ieee information theory workshop (itw)*. IEEE, 2015, pp. 1–5.

[171] H. Liu, X. Su, X. Shen, L. Chen, and X. Zhou, "Bigset: Binary mask-guided separation training for dnn-based hyperspectral anomaly detection," *arXiv preprint arXiv:2307.07428*, 2023.

[172] R. Seelaboyina and R. Vishwakarma, "Different thresholding techniques in image processing: A review," in *ICDSMLA 2021: Proceedings of the 3rd International Conference on Data Science, Machine Learning and Applications*. Springer, 2023, pp. 23–29.

[173] F. Najjar, S. Waheed, and D. Mahdi, "Coronavirus classification based on enhanced x-ray images and deep learning," *Malaysian Journal of Fundamental and Applied Sciences*, vol. 19, pp. 369–378, 05 2023.

[174] N. Ullah, J. A. Khan, S. Almakdi, M. S. Khan, M. Alshehri, D. Alboaneen, and A. Raza, "A novel coviddetnet deep learning model for effective covid-19 infection detection using chest radiograph images," *Applied Sciences*, vol. 12, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/12/6269

[175] F. Najjar, K. Abdulameer, M. Hamza, H. Abbas, D. Mahdi, and H. Al-Hindawi, "Classification of covid-19 from x-ray images using glcm features and machine learning," *Malaysian Journal of Fundamental and Applied Sciences*, vol. 19, pp. 389–398, 05 2023.

[176] J. Amann, A. Blasimme, E. Vayena, D. Frey, V. I. Madai, and P. Consortium, "Explainability for artificial intelligence in healthcare: a multidisciplinary perspective," *BMC medical informatics and decision making*, vol. 20, pp. 1–9, 2020.

[177] e. a. C. Zhang, "A Survey on Federated Learning." *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[178] e. a. H. Brendan McMahan, "Communication-Efficient Learning of Deep Networks from Decentralized Data." arXiv preprint arXiv:1602.05629, 2023.

[179] e. a. M. Moshawrab, "Reviewing Federated Machine Learning and Its Use in Diseases Prediction." *Sensors*, vol. 23, no. 4, p. article 2112, 2023.

[180] e. a. B. A. Malin, "Biomedical Data Privacy: Problems, Perspectives, and Recent Advances." *Journal of the American Medical Informatics Association (JAMIA)*, vol. 20, no. 1, pp. 2–6, 2013.

[181] e. a. Q. Li, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection." *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3347–3366, 2023.

[182] e. a. I. De Falco, "A Federated Learning-Inspired Evolutionary Algorithm: Application to Glucose Prediction." *Sensors*, vol. 23, no. 6, p. article 2957, 2023.

[183] e. a. Dinh C. Nguyen, "Federated Learning for Smart Healthcare: A Survey." *ACM Computing Surveys*, vol. 55, no. 3, pp. article 60, 37 pages, March 2023.

[184] L. Gao, L. Li, Y. Chen, C. Xu, and M. Xu, "Fgfl: A blockchain-based fair incentive governor for federated learning," *Journal of Parallel and Distributed Computing*, vol. 163, pp. 283–299, 2022.

[185] P. V. Astillo, D. G. Duguma, H. Park, J. Kim, B. Kim, and I. You, "Federated intelligence of anomaly detection agent in iotmd-enabled diabetes management control system," *Future Generation Computer Systems*, vol. 128, pp. 395–405, 2022.

[186] H. Islam, A. Mosa *et al.*, "A federated mining approach on predicting diabetes-related complications: Demonstration using real-world clinical data," in *AMIA Annual Symposium Proceedings*, vol. 2021. American Medical Informatics Association, 2021, p. 556. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8861723/

[187] S. Singh, S. Rathore, O. Alfarraj, A. Tolba, and B. Yoon, "A framework for privacy-preservation of iot healthcare data using federated learning and blockchain technology," *Future Generation Computer Systems*, vol. 129, pp. 380–388, 2022.

[188] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, "Biscotti: A blockchain system for private and secure federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, 2020.

[189] Y. Chang, C. Fang, W. Sun *et al.*, "A blockchain-based federated learning method for smart healthcare," *Computational Intelligence and Neuroscience*, vol. 2021, 2021.

[190] Z. Xing, Z. Zhang, M. Li, J. Liu, L. Zhu, G. Russello, and M. R. Asghar, "Zero-knowledge proof-based practical federated learning on blockchain," *arXiv preprint arXiv:2304.05590*, 2023.

[191] G. Almashaqbeh and Z. Ghodsi, "Anofel: Supporting anonymity for privacy-preserving federated learning," *arXiv preprint arXiv:2306.06825*, 2023.

[192] W. Yang, Y. Yin, G. Zhu, H. Gu, L. Fan, X. Cao, and Q. Yang, "Fedzkp: Federated model ownership verification with zero-knowledge proof," *arXiv preprint arXiv:2305.04507*, 2023.

[193] e. a. P. de Chazal, "A Comparison of Extreme Learning Machines and Back-Propagation Trained Feed-Forward Networks Processing the MNIST Database." in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 2165–2168.

[194] e. a. E. I. Georga, "Online Prediction of Glucose Concentration in Type 1 Diabetes Using Extreme Learning Machines." in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015, pp. 3262–3265.

[195] e. a. X. Mo, "Hypoglycemia Prediction Using Extreme Learning Machine (ELM) and Regularized ELM." in *Proceedings of the 2013 25th Chinese Control and Decision Conference (CCDC)*, 2013, pp. 4405–4409.

[196] e. a. N. Elsayed, "Early Stage Diabetes Prediction via Extreme Learning Machine." arXiv preprint arXiv:2202.11216, 2022.

[197] C. Marling and R. Bunescu, "The OhioT1DM Dataset for Blood Glucose Level Prediction: Update 2020." in *CEUR Workshop Proceedings*, vol. 2675, 2020, pp. 71–74.

[198] D. Dave, K. Vyas, J. K. Jayagopal, A. Garcia, M. Erraguntla, and M. Lawley, "Fedglu: A personalized federated learning-based glucose forecasting algorithm for improved performance in glycemic excursion regions," *arXiv preprint arXiv:2408.13926*, 2024.

[199] C. Piao, T. Zhu, Y. Wang, S. E. Baldeweg, P. Taylor, P. Georgiou, J. Sun, J. Wang, and K. Li, "Privacy preserved blood glucose level cross-prediction: An asynchronous decentralized federated learning approach," *arXiv preprint arXiv:2406.15346*, 2024.

[200] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[201] A. Ahi and A. V. Singh, "Role of distributed ledger technology (dlt) to enhance resiliency in internet of things (iot) ecosystem," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*. IEEE, 2019, pp. 782–786.

[202] T. R. Gadekallu, Q.-V. Pham, D. C. Nguyen, P. K. R. Maddikunta, N. Deepa, B. Prabadevi, P. N. Pathirana, J. Zhao, and W.-J. Hwang, "Blockchain for edge of things: Applications, opportunities, and challenges," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 964–988, 2022.

[203] A. O. Joseph, J. Raffety, P. Morrow, L. Zhiwei, C. Nugent, S. McClean, and G. Ducatel, "Securing self-organizing iot ecosystem: A distributed ledger technology approach," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019, pp. 809–814.

[204] S. Otoum, I. A. Ridhawi, and H. Mouftah, "Securing critical iot infrastructures with blockchain-supported federated learning," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2592–2601, 2022.

[205] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of things (iot): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 474–10 498, 2021.

[206] IOTA, "Learn about iota," in *Link to the official website, accessed on 14/06/2023*, 2023, pp. https://wiki.iota.org/learn/about–iota/an–introduction–to–iota.

[207] T. Alsboui, Y. Qin, R. Hill, and H. Al-Aqrabi, "Towards a scalable iota tangle-based distributed intelligence approach for the internet of things," in *Science and Information Conference*. Springer, 2020, pp. 487–501.

[208] W. F. Silvano and R. Marcelino, "Iota tangle: A cryptocurrency to communicate internet-of-things data," *Future generation computer systems*, vol. 112, pp. 307–319, 2020.

[209] F. Guo, X. Xiao, A. Hecker, and S. Dustdar, "Characterizing iota tangle with empirical data," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[210] G. Pescetelli, L. Petrosino, S. Della Valle, G. Ronga, M. Merone, and L. Vollero, "Framework for iot ecosystems based on distributed ledger technologies and decentralized identifiers," in *2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT)*. IEEE, 2022, pp. 87–91.

[211] L. Petrosino, G. Pescetelli, Q. Fieramosca, S. Della Valle, M. Merone, and L. Vollero, "drain: A distributed reliable architecture for iot networks," *IEEE Internet of Things Journal*, 2023.

[212] B. Kusmierz, P. Staupe, and A. Gal, "Extracting tangle properties in continuous time via large-scale simulations," *Technical Report. working paper*, pp. 2018–08, 2018.

[213] S. Popov, "The tangle," *White paper*, vol. 1, no. 3, p. 30, 2018.

[214] G. Bu, Ö. Gürcan, and M. Potop-Butucaru, "G-iota: Fair and confidence aware tangle," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 644–649.

[215] S. Moon and W. H. Lee, "Privacy-preserving federated learning in healthcare," in *2023 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE, 2023, pp. 1–4.

[216] E. Goldman, "An introduction to the california consumer privacy act (ccpa)," *Santa Clara Univ. Legal Studies Research Paper*, 2020. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3211013

[217] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 2938–2948.

[218] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, no. 2, pp. 513–535, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s13042-022-01647-y

[219] L. Petrosino, L. Masi, F. D'Antoni, M. Merone, and L. Vollero, "A zero-knowledge proof federated learning on dlt for healthcare data," *Journal of Parallel and Distributed Computing*, vol. 196, p. 104992, 2025.

[220] W3C, "Decentralized identifiers (dids)," *online*, 2022. [Online]. Available: https://www.w3.org/TR/did-core/

[221] J. C. Benaloh, "Secret sharing homomorphisms: Keeping shares of a secret secret," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 251–260.

[222] L.-J. Pang and Y.-M. Wang, "A new (t, n) multi-secret sharing scheme based on shamir's secret sharing," *Applied Mathematics and Computation*, vol. 167, no. 2, pp. 840–848, 2005.

[223] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*. Springer, 2008, pp. 486–498.

[224] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: The garbled circuit approach," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 591–602.

[225] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso *et al.*, "Privacy-preserving federated brain tumour segmentation," in *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*. Springer, 2019, pp. 133–141.

[226] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *International journal of medical informatics*, vol. 112, pp. 59–67, 2018.

[227] F. D'Antoni, L. Petrosino, A. Marchetti, L. Bacco, S. Pieralice, L. Vollero, P. Pozzilli, V. Piemonte, and M. Merone, "Layered meta-learning algorithm for predicting adverse events in type 1 diabetes," *IEEE Access*, vol. 11, pp. 9074–9094, 2023.

[228] A. Z. Woldaregay, E. Årsand, S. Walderhaug, D. Albers, L. Mamykina, T. Botsis, and G. Hartvigsen, "Data-driven modeling and prediction of blood glucose dynamics: Machine learning applications in type 1 diabetes," *Artificial intelligence in medicine*, vol. 98, pp. 109–134, 2019.

[229] E. Kraegen, D. Chisholm, and M. E. McNamara, "Timing of insulin delivery with meals," *Hormone and Metabolic Research*, vol. 13, no. 07, pp. 365–367, 1981.

[230] D. Boiroux, D. A. Finan, J. B. Jørgensen, N. K. Poulsen, and H. Madsen, "Optimal insulin administration for people with type 1 diabetes," *IFAC Proceedings Volumes*, vol. 43, no. 5, pp. 248–253, 2010, 9th IFAC Symposium on Dynamics and Control of Process Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016300428

[231] W. Wang, M. Tong, and M. Yu, "Blood glucose prediction with vmd and lstm optimized by improved particle swarm optimization," *IEEE Access*, vol. 8, pp. 217 908–217 916, 2020.

[232] M. F. Rabby, Y. Tu, M. I. Hossen, I. Lee, A. S. Maida, and X. Hei, "Stacked lstm based deep recurrent neural network with kalman smoothing for blood glucose prediction," *BMC Medical Informatics and Decision Making*, vol. 21, pp. 1–15, 2021.

[233] F. D'Antoni, L. Petrosino, A. Velieri, D. Sasso, O. D'Angelis, T. Boscarino, L. Vollero, M. Merone, and V. Piemonte, "Identification of optimal training for prediction of glucose levels in type-1-diabetes using edge computing," in *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, 2022, pp. 1–5.

[234] E. community developers: rileyannon Crosstons pettinarip minimalsm jmcook1186 koonopek corwintines wenceslas sanchez vardhanshorewala 0xKai27 Pandapip1 theSamPadilla medardm wackerow lawlesx, "Gas and fees," *Ethereum org developers docs*, last visit 10/07/2024. [Online]. Available: https://ethereum.org/en/developers/docs/gas/

[235] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 2016, pp. 305–326.

[236] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[237] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 2, 2004, pp. 985–990.

[238] e. a. Guang-Bin Huang, "Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks." in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 2, 2004, pp. 985–990.

[239] e. a. H. Zhou, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting." arXiv preprint arXiv:2012.07436, 2021.

[240] e. a. T. Lin, "A Survey of Transformers." arXiv preprint arXiv:2106.04554, 2021.

[241] e. a. H. Nemat, "Blood Glucose Level Prediction: Advanced Deep-Ensemble Learning Approach." *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 6, pp. 2758–2769, 2022.

[242] e. a. H. Khadem, "Blood Glucose Level Time Series Forecasting: Nested Deep Ensemble Learning Lag Fusion." *Bioengineering*, vol. 10, no. 4, p. article 487, 2023.

[243] e. a. J. Martinsson, "Blood Glucose Prediction with Variance Estimation Using Recurrent Neural Networks." *Journal of Healthcare Informatics Research*, vol. 4, no. 1, pp. 1–18, 2019.

[244] e. a. J. Sevilla, "Estimating Training Compute of Deep Learning Models. 2022. [Online]. Available: https://epochai.org/blog/estimating-training-compute. Accessed: 2024-06-13."

[245] e. a. Guang-Bin Huang, "Extreme Learning Machine: Theory and Applications." *Neurocomputing*, vol. 70, pp. 489–501, 2006.

[246] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *IEEE Transactions on Information Theory*, vol. 68, pp. 8076–8091, 2020.

[247] J. Liu, F. Lai, Y. Dai, A. Akella, H. Madhyastha, and M. Chowdhury, "Auxo: Efficient federated learning via scalable client clustering," *Proceedings of the 2023 ACM Symposium on Cloud Computing*, 2022.

[248] E. Rocheteau, C. Tong, P. Veličković, N. Lane, and P. Liò, "Predicting patient outcomes with graph representation learning," *arXiv preprint arXiv:2101.03940*, 2021.

[249] Y. Liu, B. He, D. Dong, Y. Shen, T. Yan, R. Nian, and A. Lendasse, *ROS-ELM: A Robust Online Sequential Extreme Learning Machine for Big Data Analytics*. ., 01 2015, pp. 325–344.

[250] E. Kristiani, C. Yang, and C. Huang, "isec: An optimized deep learning model for image classification on edge computing," *IEEE Access*, vol. 8, pp. 27 267–27 276, 2020.