Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Artificial Intelligence (37^{th}cycle)

Learning-Based Methods for Enabling On-Edge, Accurate, Sustainable, and Human-Centered Intelligent Manufacturing

By

Luigi Capogrosso

******

Supervisors:
Prof. Marco Cristani, Supervisor
Prof. Franco Fummi, Co-Supervisor

Doctoral Examination Committee:
Prof. Christoph Lüth, *Referee*, University of Bremen
Prof. Nicola Mazzocca, *Referee*, University of Naples Federico II
Prof. Alessandro Beghi, *Member*, University of Padua
Prof. Stefano Di Carlo, *Member*, Polytechnic University of Turin
Prof. Sebastiano Vascon, *Member*, Ca' Foscari University of Venice

Politecnico di Torino
2025

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Luigi Capogrosso
2025

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

First and foremost, I extend my deepest gratitude to my supervisors, Prof. Marco Cristani and Prof. Franco Fummi, for their support throughout my Ph.D. Their advice guided me in my research and made me the researcher that I am today. They trusted in me and the scientific freedom they have given me has been a decisive factor in all my work so far. I think this is the essence of a Ph.D.: fearlessly exploring and, in the meantime, improving ourselves. Most importantly, they changed my life by convincing me to pursue a Ph.D. As of this writing, I cannot imagine myself doing anything else, and for this, I will be grateful forever.

I also want to express my appreciation to Geri Skenderi, who is not just a colleague but a true friend. Our discussions, exchange of ideas, and late-night work sessions in the lab made this journey more enriching. My heartfelt thanks also go to Andrea Avogaro and Andrea Toaiari, who have been true friends from the very beginning.

The interdisciplinary nature of my Ph.D. allowed me the unique privilege of sharing this journey with an exceptional group of labmates. A special thanks to my colleagues from the Intelligo ML Lab: Federico Girella, Uzair Khan, Ziyue Liu, Francesco Taioli, and Federico Cunico. And from the ESD Lab: Francesco Biondani, Enrico Fraccaroli, Nicola Dall'Ora, Sebastiano Gaiardelli, Michele Lora, Francesco Tosoni, and Mario Libro. Your curiosity and passion for learning have continually motivated me to push forward.

I also express my sincere gratitude to Prof. Mário A. T. Figueiredo for warmly welcoming me into his lab at Instituto Superior Técnico. The opportunity to explore new problems and perspectives there was invaluable. I am especially grateful to all my colleagues at the Pattern Recognition and Automatic Learning Lab, particularly André Gomes and Mohammadreza Ghafari, for shared moments that enriched my academic journey.

Throughout this Ph.D., I have had the privilege of collaborating with many outstanding individuals. I am especially grateful to Prof. Tiziano Villa for their guidance, for allowing me to be their coauthor, and for teaching me so much along the way. Your contributions have had an important impact on my growth as a researcher and, more importantly, as a person.

Another special thanks goes to my friends, who have always supported me in the last few years. There is no point in naming you all; I know that each of you reading these lines will know that I am talking about him. Thanks, guys, for always being there for me.

However, no one has been more important to me in the pursuit of this thesis than the members of my family. I would like to thank my parents for their love and guidance in whatever I pursue. I have an amazing family, unique in many ways and the stereotype of a perfect family in many others. Their support has been unconditional for all these years. They are my ultimate role models.

Abstract

Four major evolutions of industrialization have occurred throughout human history, impacting economic growth, population expansion, and significant social transformations. Industry 5.0 is regarded as the next industrial revolution, and its objective is to leverage the creativity of human experts in collaboration with efficient, accurate, and intelligent machines. In this context, the transformation of industrial resources into intelligent and adaptive objects with sensing and acting capabilities leads to intelligent manufacturing. To comprehensively enhance manufacturing systems capabilities, this thesis presents cutting-edge learning-based techniques around the four key pillars of intelligent manufacturing: ***efficient on-edge computing***, ***accurate anomaly detection***, ***sustainable new products performance forecasting***, *and* ***human-centered systems design***.

In the efficient on-edge computing domain [1], we explore the application of Split Computing (SC), where a Deep Neural Network (DNN) is intelligently split with a part of it deployed on an edge device and the rest on a remote server, enabling real-time processing on large-scale industrial data. Firstly, we developed the ***I-SPLIT*** *[2] and* ***Split-Et-Impera*** *[3] techniques to partition learning models efficiently. However, in many industrial scenarios, the same DNN is used for multiple inference tasks. So, in* ***MTL-Split*** *[4], we studied how to partition such a multi-tasking DNN to be deployed within an SC framework. In* ***Sparsity-SC&EE*** *[5], we present the effect of predefined sparsity within SC to significantly reduce computational, storage, and energy demands during training and inference. We also introduce* ***LO-SC*** *[6], a methodology that leverages the principles of SC to split a DNN for execution entirely on an edge device without sacrificing model accuracy. Finally, we present* ***LE-CPSs*** *[7], examining SC architectures and their impact on controller design.*

Predictive Maintenance (PdM) plays a fundamental role in intelligent manufacturing since it guarantees the ongoing reliability and efficiency of advanced

technological systems [8]. Central to PdM is anomaly detection, which serves as the initial step in identifying potential issues. This thesis explores both vision-based and time-series-based approaches to tackle this task. In the vision domain, our research focuses on Surface Defect Detection (SDD). Specifically, we develop ***In&Out*** *[9] and* ***DIAG*** *[10], two diffusion-based data augmentation methodologies to enhance the performance of learning classifiers. For time-series data, we present* ***GAIA*** *[11], in which we highlight the role of Generative Artificial Intelligence (GenAI) in overcoming the limitations of real-world data, and* ***ChronosAD*** *[12], the first application of time-series foundation models for fault classification.*

Another core principle of intelligent manufacturing is sustainability. In particular, the fast fashion industry represents the second most pollutive industry in the world [13]. Accordingly, we focus on addressing sustainability challenges within this sector. In particular, we target the New Fashion Products Performance Forecasting (NFPPF) task and introduce ***MDiFF*** *[14] and* ***Dif4FF*** *[15], two novel pipelines based on multimodal diffusion models for forecasting sales of new fashion products.*

The final aspect centers on fully integrating humans within intelligent manufacturing environments. In ***WiFi-Based Detection*** *[16] and* ***SITUATE*** *[17], we develop systems to predict environmental conditions and human movements in indoor settings, ensuring worker safety while maintaining operational efficiency. Finally, in* ***HC-DT*** *[18], we introduce the first pipeline for implementing human-centered Digital Twins (DTs).*

Most of the presented works were examined into the manufacturing infrastructure of the ***Industrial Engineering Laboratory (ICE Lab)****. This practical integration allowed for the evaluation of the proposed systems within an operational environment, ensuring their effectiveness and functionality.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Four major industrial revolutions have shaped economies, accelerated population growth, and driven profound social changes throughout history. Specifically, with each industrial revolution, new technological advances are made possible by a better understanding of the natural world and its resources. One of the most recent, known as Industry 4.0 [24], has process automation as its main priority, thereby reducing the intervention of humans in the manufacturing process [25]. Thus, Industry 4.0 focuses on improving mass productivity and performance by providing intelligence between devices and applications using Machine Learning (ML) techniques [26].

Industry 5.0 is considered the next industrial revolution, and its objective is to harness the creativity of human experts in collaboration with efficient, intelligent, and accurate machines to obtain resource-efficient and user-preferred manufacturing solutions compared to Industry 4.0 [27]. At the heart of Industry 5.0 is intelligent manufacturing, a vision of the future factory that uses advanced computational techniques to meet the increasingly complex demands of production and supply chains. This digital transformation helps industries achieve greater sustainability and adopt more environmentally friendly practices [28]. The primary goal of intelligent manufacturing within Industry 5.0 is to create environments where "smart" objects can autonomously sense, analyze, and interact within interconnected systems. This involves transforming conventional industrial assets into Cyber-Physical Systems (CPSs) incorporating data and intelligence, which in manufacturing have been dubbed Cyber-Physical Production Systems (CPPSs) [29]. Such systems can communicate with each other, acquiring and transmitting real-time production data

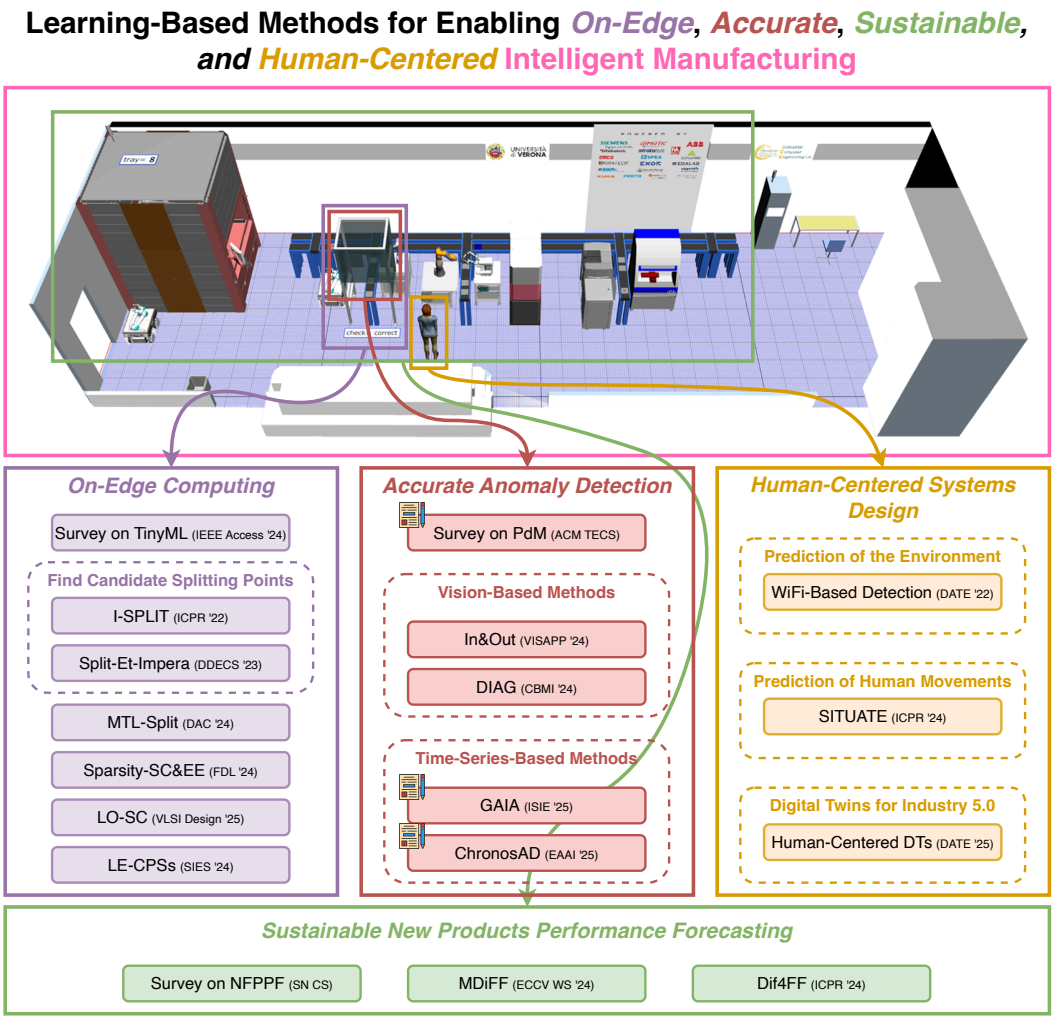


Figure 1.1 The real-world setup of the Industrial Computer Engineering (ICE) Laboratory at the University of Verona, where most of the developed contributions of this thesis have been contextualized and tested. We focused our research on the four key pillars of intelligent manufacturing: edge computing, accurate anomaly detection, sustainable new products performance forecasting, and human-centered systems design. The notebook icon with the pencil indicates that the contribution is currently under submission.

used to optimize production processes. This main contribution increases production throughput while reducing costs and waste [30].

However, achieving this transformation requires robust frameworks that integrate advanced Artificial Intelligence (AI) techniques in the following key functional areas: *on-edge computing*, *accurate anomaly detection*, *sustainable new products performance forecasting*, and *human-centered systems design*. Integrating these four pillars represents a crucial advance in unlocking the full potential of intelligent

manufacturing within Industry 5.0. This thesis introduces a set of methodologies that harness these principles to develop resilient, adaptive, and sustainable manufacturing systems, enhancing operational efficiency while prioritizing both worker welfare and environmental sustainability.

We aim to achieve these objectives, as illustrated in Figure 1.1, which presents the real-world setup of the Industrial Computer Engineering (ICE) Laboratory of the University of Verona[1], where most of the presented contributions of this thesis were contextualized and evaluated. Specifically, the ICE Laboratory was designed to develop cutting-edge Information Technology (IT) solutions and methodologies in Industry 4.0 and 5.0 with respect to automation and efficiency of production lines. The laboratory's goals are research, technology transfer, and teaching, fostering an environment that bridges theoretical advancements with practical applications [31]. By focusing on Industry 4.0 and 5.0, the ICE Laboratory aims to drive innovations that improve the automation, flexibility, and sustainability of production processes. This thesis aligns with these objectives, contributing to the lab's mission by developing frameworks and methodologies that not only improve operational efficiency but also address emerging challenges in sustainable product design and human-centric AI integration.

Motivations for on-edge computing. In intelligent manufacturing, on-edge computation is used to enable real-time processing and decision-making by distributing computational tasks to devices at the network edge, such as sensors and machines, rather than relying solely on centralized cloud resources [1]. This decentralization not only reduces latency but also improves the resilience of the system, enabling autonomous decision-making even in environments with limited connectivity. Traditional cloud computing approaches, which send all data to a central server for processing, are often ill-suited for high-speed, data-intensive industrial applications, where delays can lead to bottlenecks or even system failures [6]. On-edge computing frameworks address these challenges by creating hybrid models that split tasks across edge and cloud components, allowing for scalable and efficient data management without sacrificing performance [32].

The design and deployment of on-edge computing architectures introduce unique challenges in terms of resource allocation, network constraints, and computational

[1] https://www.icelab.di.univr.it/.

limitations of edge devices [3]. Standard deep learning models typically require substantial processing power and memory, which are not readily available in edge devices due to size, power, and cost constraints [1]. Consequently, a major focus of this thesis is exploring Split Computing (SC) architectures, a promising approach that divides Deep Neural Networks (DNNs) into parts executed on both edge devices and cloud servers [32]. By strategically selecting the split points within a DNN, it is possible to balance the workload across the network, ensuring that resource-intensive tasks are handled by the cloud while preserving real-time prediction capabilities on the edge.

Motivations for accurate anomaly detection. Beyond on-edge computing, Predictive Maintenance (PdM) is another field that plays a crucial role in modern manufacturing since it enables proactive identification and resolution of potential failures before they impact production [8]. Maintenance is so important because it ensures smooth operation and long-life of the equipment, leading to increased productivity and safety. Minimize downtime and prevent costly breakdowns, supporting a sustainable and efficient industrial environment. Specifically, PdM is highly dependent on anomaly detection techniques, which are the basis for identifying deviations in system behavior indicative of faults or malfunctions [8].

Motivations for sustainable new products performance forecasting. Sustainability has emerged as another imperative pillar in intelligent manufacturing, especially in industries with significant environmental impacts, such as fast fashion [33]. The fashion industry is known to be an important contributor to pollution, responsible for substantial carbon emissions, water consumption, and waste generation [34]. Addressing these issues requires not only changes in production practices, but also advances in demand forecasting methods that can guide sustainable production decisions.

Motivations for human-centered systems design. The evolution from Industry 4.0 to Industry 5.0 represents a significant shift in industrial practices [35, 36]. This new paradigm emphasizes human-machine collaboration, where technology improves productivity and ensures adaptability, resilience, and sustainability in industrial operations. Consequently, the objective is not only to optimize machine

performance, but also to ensure human operators' safety, well-being, and active participation in these highly automated environments. This shift introduces unique challenges, particularly in the integration of the human element within digital data-driven industrial ecosystems.

1.1 Contributions

The contributions of this thesis align with the four pillars outlined above, each targeting a specific aspect of intelligent manufacturing.

Innovations in on-edge computing. The thesis presents advanced methodologies for SC and Early Exit (EE), aimed at distributing deep learning tasks between edge devices and remote servers efficiently. Specifically, we develop the following frameworks and methodologies:

- **I-SPLIT [2] (Section 2.2):** A novel method that optimally divides a DNN into edge-deployed and server-deployed segments based on interpretability metrics, specifically focusing on maximizing accuracy without repeated trial-and-error splits. **I-SPLIT** calculates optimal split points by assessing the saliency of the neuron, enabling robust resource allocation for edge-based inference.
- **Split-Et-Impera [3] (Section 2.3):** A user-oriented framework for configuring SC systems with multi-dimensional constraints, including computation, network protocols, and Quality of Service (QoS) requirements. Dynamically suggests configurations that balance latency, accuracy, and resource allocation.
- **MTL-Split [4] (Section 2.4):** A Multi-Task Learning (MTL) model within a SC framework that allows multiple related tasks to be learned jointly by sharing common features. This approach reduces the need for multiple DNNs by employing a single model capable of efficiently performing several inference tasks in parallel.
- **Sparsity-SC&EE [5] (Section 2.5):** A framework that incorporates predefined sparsity in the SC and EE strategies, achieving significant reductions in computational demands and energy use during the model training and inference phases.

- **LO-SC [6] (Section 2.6):** In this article, we show how to optimally split the neural network based on the computational capabilities of the edge device while minimizing a desired metric.
- **LE-CPSs [7] (Section 2.7):** Finally, in this paper, we introduce "Split Computing CPS", examining various SC architectures and their impact on controller design.

Innovations in accurate anomaly detection. In industrial contexts, anomalies can manifest in various forms, such as sensor data inconsistencies, unusual machine vibrations, or atypical production patterns, each of which requires specialized algorithms for accurate detection and classification. As a result, in this thesis, we address the task of anomaly detection by exploring both vision-based and time-series-based techniques. By addressing both modalities, the research ensures a more comprehensive and robust detection framework capable of handling a wide range of anomalies that may arise in industrial settings. Specifically, we develop the following frameworks and methodologies:

- **In&Out [9] (Section 3.2):** A diffusion-based data augmentation method designed for anomaly detection in vision-based applications. By blending in-distribution and out-of-distribution samples, it improves the robustness of the classifiers in Surface Defect Detection (SDD).
- **DIAG [10] (Section 3.3):** An interactive protocol that uses Vision Language Model (VLM) to enhance anomaly detection by generating realistic and context-specific images from text prompts. **DIAG** broadens the anomaly detection landscape, especially in environments where labeled data may be sparse.
- **GAIA [11] (Section 3.4):** The first comprehensive pipeline to allow the creation of Digital Twins (DTs) to support PdM in the aircraft domain.
- **ChronosAD [12] (Section 3.5):** Introduces time-series foundation models for fault classification, applying these innovations to predictive maintenance across various industrial processes.

Innovations in sustainable new products performance forecasting. This thesis tackles the challenge of New Fashion Products Performance Forecasting (NFPPF) [37], *i.e.*, predicting the performance of a brand-new clothing probe with no available past observations. By improving the precision of demand forecasts, these models help mitigate overproduction and reduce waste, aligning the fashion industry with broader goals of environmental and economic sustainability [13]. Specifically, we develop the following frameworks and methodologies:

- **MDiFF [14] (Section 4.2):** A multimodal diffusion-based forecasting framework specifically developed for the NFPPF task. **MDiFF** uses a two-step pipeline that fuses time-series data and product images to accurately predict product performance. This model supports more sustainable production decisions by minimizing overproduction and waste.
- **Dif4FF [15] (Section 4.3):** An extension of **MDiFF** that increases conditioning modalities by adding Google Trends to generate more accurate samples. Furthermore, it introduces a more powerful model to refine multiple predictions that factorize the temporal and spatial dimensions of the samples, reducing the overall error of the pipeline and improving the model adaptability for the prediction of diverse product lines. **Dif4FF** aims to generalize the **MDiFF** pipeline, making it more versatile and suitable for broader applications within the fast fashion industry.

Innovations in human-centered systems design. As intelligent manufacturing increasingly prioritizes worker safety and intuitive human-machine collaboration, we developed the following frameworks and methodologies:

- **WiFi-Based Detection [16] (Section 5.2) and SITUATE [17] (Section 5.3):** Systems to predict environmental conditions and human movements in indoor settings, ensuring worker safety while maintaining operational efficiency.
- **HC-DT [18] (Section 5.4):** The first pipeline for the creation of a human-centered DT, leveraging Unreal Engine MetaHuman technology to track worker alertness in real-time. Our findings demonstrate the potential for the integration of AI and human-centered design within Industry 5.0 to enhance both worker safety and industrial efficiency.

1.2 Outline

Chapter 2 explores the development of on-edge computing solutions with a focus on the SC and EE frameworks. It discusses multiple methodologies for splitting deep learning models between edge devices and cloud servers, highlighting the trade-offs between computation and network latency.

In Chapter 3, anomaly detection techniques for predictive maintenance are presented. The chapter covers both vision-based and time-series-based anomaly detection, focusing on data augmentation, interactive learning protocols, and the application of generative AI techniques for real-world data scarcity challenges.

Chapter 4 addresses sustainability challenges in intelligent manufacturing, using as a use case the fast fashion industry. Specifically, this chapter presents methodologies for forecasting the performance of a brand-new clothing probe without previously available observations using multimodal diffusion models. The chapter details their application in resource-efficient and environmentally conscious manufacturing.

Chapter 5 examines human-centered systems within intelligent manufacturing. It highlights safety-focused applications and human movement prediction models that enhance worker-machine interaction and ensure safety in industrial settings. Furthermore, it presents the first comprehensive pipeline proposed for implementing a human-centered implementation DTs.

Finally, Chapter 6 summarizes the key findings, reflects on the implications of the proposed methodologies for the future of intelligent manufacturing, and suggests potential directions for further research in intelligent manufacturing.

1.3 Publications

In the following section, the author's publications are listed. Note that some articles have not been included in the thesis. The included articles are reported in bold.

1. Skenderi, Geri and Bozzini, Alessia and Capogrosso, Luigi and Agrillo, Enrico Carlo and Perbellini, Giovanni and Fummi, Franco and Cristani, Marco. (2021)

DOHMO: Embedded Computer Vision in Co-Housing Scenarios
In Forum on Specification & Design Languages (FDL).

2. Capogrosso, Luigi and Skenderi, Geri and Girella, Federico and Fummi, Franco and Cristani, Marco. (2022)
Toward Smart Doors: A Position Paper
In International Conference on Pattern Recognition (ICPR) International Workshops and Challenges.

3. Cunico, Federico and Capogrosso, Luigi and Setti, Francesco and Carra, Damiano and Fummi, Franco and Cristani, Marco. (2022)
I-SPLIT: Deep Network Interpretability for Split Computing
In International Conference on Pattern Recognition (ICPR).

4. Turetta, Cristian and Skenderi, Geri and Capogrosso, Luigi and Demrozi, Florenc and Kindt, Philipp H. and Masrur, Alejandro and Fummi, Franco and Cristani, Marco and Pravadelli, Graziano. (2023)
Towards Deep Learning-based Occupancy Detection Via WiFi Sensing in Unconstrained Environments
In Design, Automation & Test in Europe Conference & Exhibition (DATE).

5. Cunico, Federico and Capogrosso, Luigi and Castellini, Alberto and Setti, Francesco and Pluchino, Patrik and Zordan, Filippo and Santus, Valeria and Spagnolli, Anna and Cordibella, Stefano and Gennari, Giambattista and Borgo, Mauro and Sozza, Alberto and Troiano, Stefano and Flor, Roberto and Zanella, Andrea and Farinelli, Alessandro and Gamberini, Luciano and Cristani, Marco. (2023)
The Post-pandemic Effects on IoT for Safety: The Safe Place Project
In Design, Automation & Test in Europe Conference & Exhibition (DATE).

6. Capogrosso, Luigi and Cunico, Federico and Lora, Michele and Cristani, Marco and Fummi, Franco and Quaglia, Davide. (2023)
Split-Et-Impera: A Framework for the Design of Distributed Deep Learning Applications
In International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS).

7. Capogrosso, Luigi and Geretti, Luca and Cristani, Marco and Fummi, Franco and Tiziano, Villa. (2023)

HermesBDD: A Multi-Core and Multi-Platform Binary Decision Diagram Package
In International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS).

8. Morra, Lia, and Azzari, Alberto, and Bergamasco, Letizia, and Braga, Marco, and Capogrosso, Luigi, and Delrio, Federico, ..., and Vercellino, Chiara. (2023)
Designing Logic Tensor Networks for Visual Sudoku puzzle classification
In International Workshop in Neural-Symbolic Learning and Reasoning (NeSy).

9. Capogrosso, Luigi and Mascolini, Alessio and Girella, Federico and Skenderi, Geri and Gaiardelli, Sebastiano and Dall'Ora, Nicola and Ponzio, Francesco and Fraccaroli, Enrico and Di Cataldo, Santa and Vinco, Sara and Macii, Enrico and Fummi, Franco and Cristani, Marco. (2023)
Neuro-Symbolic Empowered Denoising Diffusion Probabilistic Models for Real-Time Anomaly Detection in Industry 4.0: Wild-and-Crazy-Idea Paper
In Forum on Specification & Design Languages (FDL).

10. Capogrosso, Luigi and Vivenza, Alvise and Chiarini, Andrea and Setti, Francesco, and Cristani, Marco. (2024)
Exploiting Multimodal Latent Diffusion Models for Accurate Anomaly Detection in Industry 5.0
In National Conference on Artificial Intelligence organized by CINI (Ital-IA).

11. Capogrosso, Luigi and Cunico, Federico and Cheng, Dong Seon and Fummi, Franco and Cristani, Marco. (2024)
A Machine Learning-Oriented Survey on Tiny Machine Learning
In IEEE Access. (pp. 23406-23426). IEEE.

12. Capogrosso, Luigi and Girella, Federico and Taioli, Francesco and Chiara, Michele and Aqeel, Muhammad and Fummi, Franco and Setti, Francesco and Cristani, Marco. (2024)
Diffusion-Based Image Generation for In-Distribution Data Augmentation in Surface Defect Detection
In International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP).

13. Girella, Federico and Liu, Ziyue and Fummi, Franco and Setti, Francesco and Cristani, Marco and Capogrosso, Luigi. (2024)
Leveraging Latent Diffusion Models for Training-Free In-Distribution Data Augmentation for Surface Defect Detection
In International Conference on Content-based Multimedia Indexing (CBMI).

14. Capogrosso, Luigi and Fraccaroli, Enrico and Chakraborty, Samarjit and Fummi, Franco and Cristani, Marco. (2024)
MTL-Split: Multi-Task Learning for Edge Devices using Split Computing
In Design Automation Conference (DAC).

15. Capogrosso, Luigi and Toaiari, Andrea and Avogaro, Andrea and Khan, Uzair and Jivoji, Aditya and Fummi, Franco and Cristani, Marco. (2024)
SITUATE: Indoor Human Trajectory Prediction through Geometric Features and Self-Supervised Vision Representation
In International Conference on Pattern Recognition (ICPR).

16. Capogrosso, Luigi and Fraccaroli, Enrico and Petrozziello, Giulio and Setti, Francesco and Chakraborty, Samarjit and Fummi, Franco and Cristani, Marco. (2024)
Enhancing Split Computing and Early Exit Applications through Predefined Sparsity
In Forum on Specification & Design Languages (FDL).

17. Avogaro, Andrea and Capogrosso, Luigi and Fummi, Franco and Cristani, Marco. (2024)
MDiFF: Exploiting Multimodal Score-based Diffusion Models for New Fashion Product Performance Forecasting
In European Conference on Computer Vision (Workshops) (ECCV).

18. Capogrosso, Luigi and Xu, Shengjie and Fraccaroli, Enrico and Cristani, Marco and Fummi, Franco and Chakraborty, Samarjit. (2024)
Learning-Enabled CPS for Edge-Cloud Computing
In International Symposium on Industrial Embedded Systems (SIES).

19. Avogaro, Andrea and Capogrosso, Luigi and Fummi, Franco and Cristani, Marco. (2024)

Dif4FF: Leveraging Multimodal Diffusion Models and Graph Neural Networks for Accurate New Fashion Product Performance Forecasting
In International Conference on Pattern Recognition (ICPR).

20. Cunico, Federico and Aldegheri, Stefano and Avogaro, Andrea and Boldo, Michele and Bombieri, Nicola and Capogrosso, Luigi, . . . , and Quaglia, Davide. (2024)
Enhancing Safety and Privacy in Industry 4.0: The ICE Laboratory Case Study
In IEEE Access. (pp. 154570 - 154599).

21. Capogrosso, Luigi and Fraccaroli, Enrico and Cristani, Marco and Fummi, Franco and Chakraborty, Samarjit. (2025)
LO-SC: Local-only Split Computing for Accurate Deep Learning on Edge Devices
In International Conference on VLSI Design (VLSID) [proceedings were not yet available].

22. Biondani, Francesco and Capogrosso, Luigi and Dall'Ora, Nicola and Fraccaroli, Enrico and Cristani, Marco and Fummi, Franco. (2025)
Human-Centered Digital Twin for Industry 5.0
In Design, Automation & Test in Europe Conference & Exhibition (DATE) [proceedings were not yet available].

23. Avogaro, Andrea and Capogrosso, Luigi and Toaiari, Andrea and Fummi, Franco and Cristani, Marco. (2025)
New Fashion Products Performance Forecasting: A Survey on Evolution, Models, and Emerging Trends
In Springer Nature on Computer Science [proceedings were not yet available].

24. Khan, Uzair and Cheng, Dong Seon and Setti, Francesco and Fummi, Franco and Cristani, Marco and Capogrosso, Luigi. (2025)
A Comprehensive Survey on Deep Learning-based Predictive Maintenance
In ACM Transactions on Embedded Computing Systems [currently under review].

25. Khan, Uzair and Cheng, Dong Seon and Setti, Francesco and Fummi, Franco and Cristani, Marco and Capogrosso, Luigi. (2025)

ChronosAD: Leveraging Time-Series Foundation Models for Anomaly Detection
In Engineering Applications of Artificial Intelligence (EAAI) [currently under review] [currently under review].

26. Biondani, Francesco and Capogrosso, Luigi and Dall'Ora, Nicola and Fraccaroli, Enrico and Cristani, Marco and Fummi, Franco. (2025)
GAIA: A Comprehensive Pipeline for Enabling Aircraft Digital Twin Creation
In 34th IEEE International Symposium on Industrial Electronics (ISIE) [currently under review] [currently under review]

27. Tingan, Zhu and Prateek, Ganguli and Arkaprava, Gupta and Shengjie, Xu and Luigi, Capogrosso and Enrico, Fraccaroli and Marco, Cristani and Franco, Fummi and Samarjit, Chakraborty. (2025)
Controllers for Edge-Cloud Cyber-Physical Systems
In IEEE 17th International Conference on COMmunication Systems & NETworkS (COMSNETS).

Chapter 2

On-Edge Computing

In the last decade, DNNs has made remarkable progress in solving various complex problems. The widespread implementation of DNN-based applications on edge devices has brought about what are commonly referred to as Local-only Computing (LoC) approaches. Unfortunately, the computational demands of DNN models often exceed the capabilities of the resource-constrained edge devices available today [1]. Solutions to this problem involve using simplified models, resulting in a lower overall accuracy. Consequently, the most common deployment approach for DNN-based applications is the Remote-only Computing (RoC) approach. In this paradigm, the DNN runs on a server, and the input is directly transferred from the acquisition device to the server through a network connection. The server computes the inferences and transmits the output to the device. Although this approach guarantees the general accuracy of the system, complete data transfer may lead to significant end-to-end delays in adverse channel conditions and compromise the task in extreme circumstances.

As a compromise between the LoC and the RoC approaches, the SC frameworks [32] propose to split the DNN models into a head and a tail, deployed on an edge device and a remote server, respectively. Early implementations of SC, like the work in [38], select a layer and divide the model to define the head and tail submodels. Instead, more recent SC frameworks introduce bottlenecks in order to minimize the overall latency time between the edge and the cloud [39]. Another approach that is usually applied together SC is EE. The core idea is to create models with multiple "exits" in the model, where each exit can produce the output of the

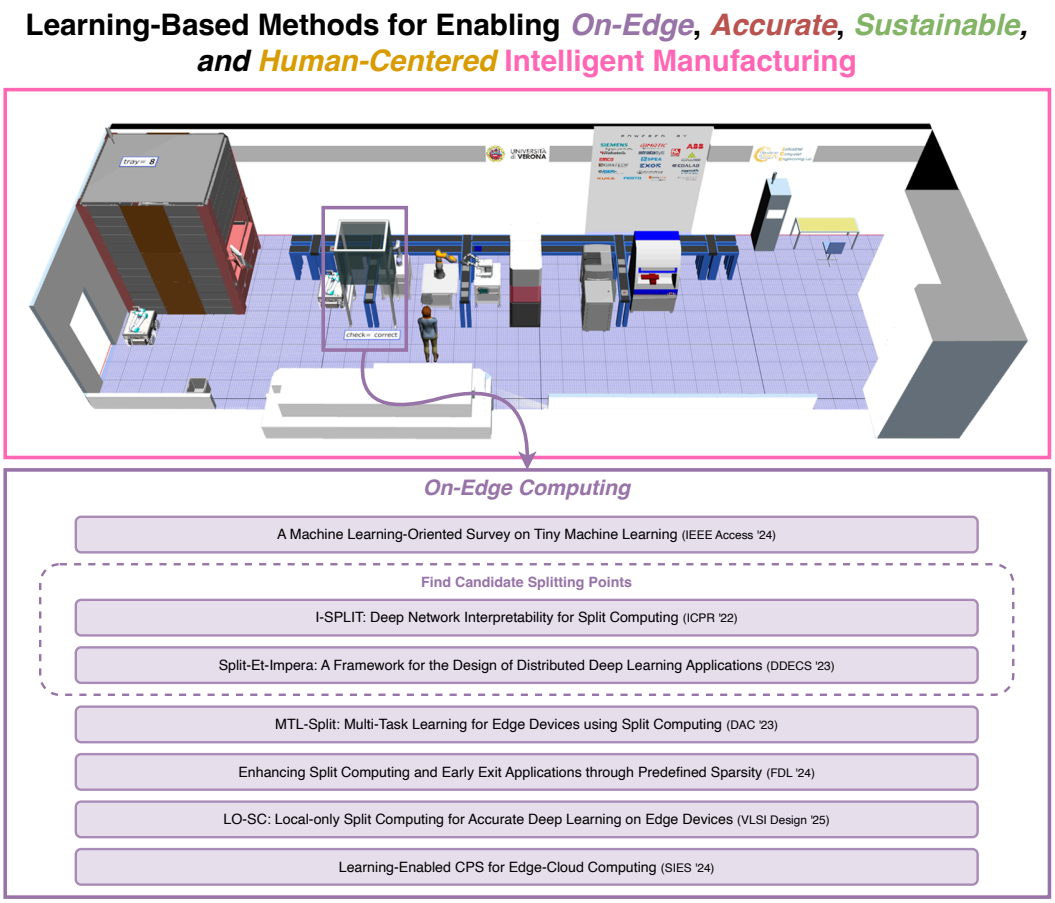


Figure 2.1 Overview of research contributions in on-edge computing.

model. Then, the first exit that provides good confidence with respect to the target is selected [32].

Such frameworks are crucial in intelligent manufacturing since the ability to balance accuracy, latency, and resource utilization enables manufacturers to deploy sophisticated AI models without sacrificing production efficiency or system reliability.

In Figure 2.1, we present all publications related to on-edge computing. The following section outlines the motivation behind each work, followed by a review of the relevant literature. Then, we provide a detailed explanation of each work, encompassing both methodologies and the achieved results.

Motivations for I-SPLIT [2] (Section 2.2). Setting up a SC system requires determining the optimal splitting point: so far, this choice was driven primarily by architectural considerations such as memory capabilities of the edge device, desired transmission rate between head and tail, and size of the layers [40]. In all of these cases, deciding on a splitting point is blind to a crucial feature that a deep network has to preserve: its classification accuracy. For example, two consecutive layers of the same size are equally likely to be chosen as splitting points, and the accuracy of the final system has to be verified only after retraining the head, bottleneck, and tail. This leads to a cumbersome trial-and-error cycle.

As a result, we propose a fast procedure to select the split location for a generic DNN architecture that, for the first time, predicts the accuracy that the system will have once retrained, thus eliminating the need for multiple trials and error retraining sessions of the system. The procedure is dubbed **I-SPLIT**, where "I" stands for interpretability. **I-SPLIT** builds upon the concept of importance or saliency of a neuron [41], which is related to the gradient it possesses with respect to the decision towards the correct class for specific input. Importance is successfully exploited in the Grad-CAM approach [41]: Grad-CAM creates an input neuron saliency map that indicates which parts of an input image are more important to decide a specific class. In particular, the Grad-CAM approach has been shown to be strongly dependent on the given trained model on which it runs (it passes the "sanity check" test [42]), while other approaches do not, making it perfectly suited for our purposes.

Motivations for Split-Et-Impera [3] (Section 2.3). However, the design of distributed deep learning applications via SC and EE involves exploring a three-dimensional design space. In fact, the final implementation will be determined by the choices of the computing platform, the communication architecture, and not only the DNN.

As a result, we propose **Split-Et-Impera**: a fast and user-friendly framework that eases the design of a distributed architecture that executes one or more DNNs. In addition to accurately mimicking diverse communication protocols and application requirements, **Split-Et-Impera** introduces a unique feature: it suggests the proper configuration to match the application's Quality of Service (QoS) requirements and provide optimal performance in terms of accuracy and latency time. Furthermore, since manipulating various SC configurations may require days of computation, **Split-**

Et-Impera allows eliminating several configurations through communication-aware simulations.

Motivations for MTL-Split [4] (Section 2.4). At the same time, current state-of-the-art approaches in different ML applications rely on advanced learning procedures, such as the MTL [43]. In particular, MTL is a paradigm in which multiple related tasks are jointly learned to improve the generalizability of a model by using shared knowledge across different aspects of the input. This is achieved by jointly optimizing the model's parameters across all tasks, allowing the model to learn both task-specific and shared representations simultaneously.

As a result, we propose **MTL-Split**, a novel model to solve multiple tasks simultaneously in a SC paradigm, *i.e.*, $T_1 \dots T_N$, where N represents the number of tasks, using only a single neural network, in contrast to current methods, where the emphasis is on Single-Task Learning (STL), which would need N neural networks to solve the tasks. Using MTL, we enhance performance in multiple tasks, elevating design challenges beyond that of preserving the performance of a single task, as in regular SC. In addition, this allows systems to operate effectively even in scenarios where data is scarce for specific tasks but abundant for others, as explained and theoretically demonstrated in [44].

Motivations for Sparsity-SC&EE [5] (Section 2.5). The importance of model reduction techniques for accelerating DNNs is widely acknowledged nowadays, aiming to optimize their performance on embedded devices [1].

As a result, in **Sparsity-SC&EE**, we show how, in the context of SC and EE, predefined sparsity can significantly reduce computational demands, storage requirements, and energy consumption compared to state-of-the-art approaches. Furthermore, regardless of the underlying implementation platform, our approach yields advantages for both the training and the inference stages.

Motivations for LO-SC [6] (Section 2.6). At the same time, current state-of-the-art approaches in different ML applications often face severe latency constraints. They cannot afford potential disruptions from network limitations, such as traffic congestion, which prevent their seamless execution on cloud-based platforms. In addition, maintaining the maximum accuracy of the model is essential to ensure

reliable and precise results. An example may be found in the industrial domain. For these settings, there is often a precise and enforced segmentation between corporate Information Technology (IT) networks and safety-critical Operational Technology (OT) networks. Certain pieces of equipment are segregated in the OT network. Here, computation should occur near the machine, on the edge system, maintain localized and secure data, and avoid risks associated with edge-to-cloud transfers.

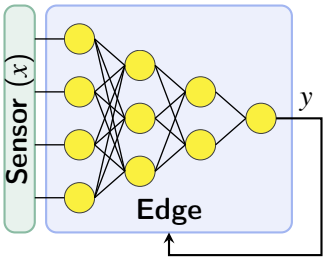
As a result, we explore the application of SC to systematically split a DNN to execute on a single edge device, introducing a new architecture called **LO-SC**. Among the optimization techniques used to address this problem, we formalize and solve a Mixed-Integer Linear Problem (MILP). Specifically, we adapt the knapsack optimization problem [45], resulting in a multi-constrained ordered knapsack problem.

Motivations for LE-CPSs [7] (Section 2.7). While such DNN [32] architectures have been studied in the past, how they should be designed when used in conjunction with feedback controllers in a CPS has not been investigated, and it remains unclear how these partitions should be "connected" to the performance of a feedback controller. Furthermore, the impact of different types of connection on the resulting dynamics of the closed-loop system is not well understood.

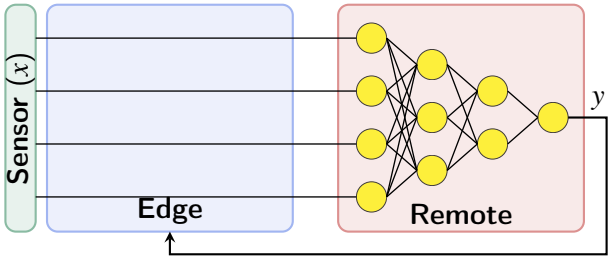
As a result, in this section, we present some of these connections between SC architectures and feedback controllers and discuss their implications on the behavior of the closed-loop dynamics of the system. We believe that this discussion will trigger useful research and open up a potentially new field – "Split Computing CPS" – that has not been explored until now.

2.1 Related Works

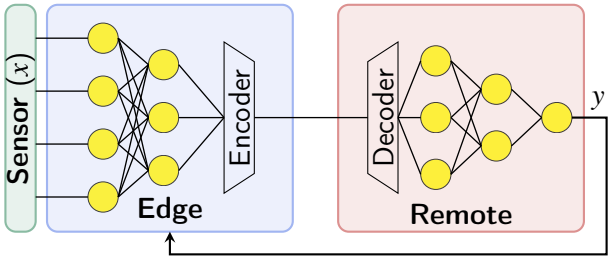
We focus on architectures operating through a DNN model $\mathscr{M}(\cdot)$, whose task is to produce the inference output y from an input x. We can identify three major types of architectures used for distributed deep learning applications in the literature: LoC, RoC, SC, and EE.



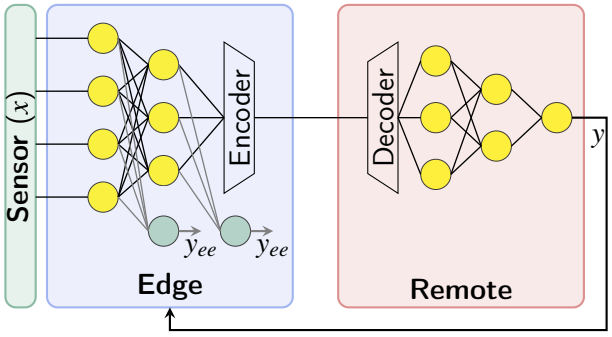
(a) With LoC, the edge device entirely executes the model inference.



(b) With RoC, the input is transferred to the remote device, where it is processed, and the result is then sent back to the edge device.



(c) With SC, the computation is split between the edge and the remote devices. The transmission bandwidth is reduced by compressing it using encoder and decoder models.



(d) When combining SC and early-exit, the computation is split between the edge and the remote devices; however, we have intermediate classification branches, producing an estimate y_{ee} of the desired output y.

Figure 2.2 Comparing three different approaches to distributed deep learning.

Local-only Computing (LoC). Under this policy, the entire computation is performed on the edge devices. As shown in Figure 2.2a, the edge device fully executes

the inference function $\mathscr{M}(x)$. Its advantage lies in its low latency due to the proximity of the computing element to the sensor [1]. However, it may not be compatible with DNN-based architectures that demand robust hardware capabilities. Usually, simpler DNN models $\bar{\mathscr{M}}(x)$ that use specific techniques (*e.g.*, depth-wise separable convolutions) are used to build lightweight networks, such as MobileNetV3 [46].

In addition to designing lightweight neural models, in the last few years, great progress has been made in the area of compression of DNN. Compression techniques, such as network pruning and quantization [47], or knowledge distillation [48] achieve a more efficient representation of one or more layers of the neural network, but with a possible quality degradation.

Remote-only Computing (RoC). As shown in Figure 2.2b, the input x is transferred from the edge device through the communication network and then is processed at the remote system through the function $\mathscr{M}(x)$. This architecture preserves full accuracy considering the higher power budget of the remote system but leads to high latency and bandwidth consumption due to the input transfer.

Split Computing (SC). The general structure of SC is shown in Figure 2.2c, which shows how the SC paradigm divides the DNN model into a head, executed by the edge sensing device, and a tail, executed by the remote system. It combines the advantages of both LoC and RoC thanks to the lower latency and, more importantly, drastically reduces the required transmission bandwidth by compressing the input to be sent x by using an autoencoder [39]. Formally, the encoder and decoder models are defined as $z_l = \mathscr{F}(x)$ and $\bar{x} = \mathscr{G}(z_l)$, which are executed at the edge and remotely, respectively. The distance $d(x,\bar{x})$ defines the performance of the encoding-decoding process.

One of the first works on SC is the study by Kang *et al.* [38], in which the authors show that the initial layers of a DNN are the most suitable candidates for partitioning, as they optimize both latency and energy consumption. Furthermore, latency reduction is usually achieved through quantization, as explored in [49], and the utilization of lossy compression techniques before data transmission, as investigated in [50]. In [51], the authors also explore lossless techniques to encode intermediate results without modifying the ML model. The concept of employing

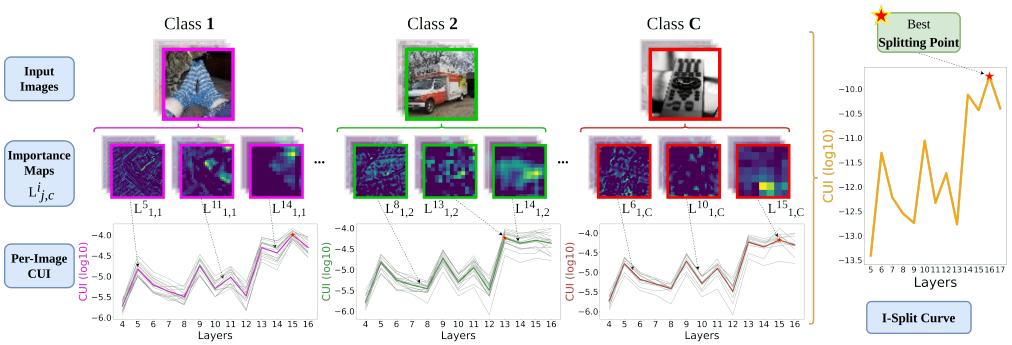


Figure 2.3 Overview of our **I-SPLIT** framework. The input images are fed into a neural network to extract high-resolution importance maps using the Grad-CAM algorithm at each layer. Then, we average over all the image pixels of each map to produce per-image CUmulated Importance (CUI) curves. Finally, all curves are fused to generate the general CUI curve. The best splitting point for the network is the global maximum of the CUI curve.

autoencoders to further compress the data is discussed in various studies, such as [52].

Early Exit (EE). As shown in Figure 2.2d adds an early exiting branch to a standard SC architecture. Formally, we can define $B_i, i = 1 \ldots N$ (with $N = L$, and L is the number of layers of the DNN) as the branch model that takes as input z_l and produces an estimate of the desired output y. In practice, the EE architecture is a modification of an existing neural network, adding one or more classification branches where, before the computation of all layers of the network, the confidence of the intermediate result is checked to be sufficient to consider the final result [53].

2.2 I-SPLIT

The final goal of **I-SPLIT** is to individuate and rank potential splitting points in a generic DNN architecture so that the accuracy values obtained by the DNN, once split at those points, follow a similar ranking. The input of the approach is a network that has been trained beforehand on some training images. The overview of the approach is shown in Figure 2.3: for each image in a pool of evaluation images, we compute importance maps associated with each layer of the network instead of being associated with input images only. These importance maps are then aggregated to form per-image CUI curves (light-colored curves). Then, these curves are averaged

together over the classes, giving the general CUI curve on the right in yellow. Each point of the CUI curve is found to be proportional to the accuracy that the network will have on the same data once the split has been applied and the network is re-trained. The highest point on the CUI curve identifies the location where the split will provide the best-performing system.

Section Organization

In the following, we first provide the mathematical details to generate the CUI curve (Section 2.2.1). Next, we detail how to compute the split in correspondence to a given location (Section 2.2.2) and how to re-train the network (Section 2.2.3). Finally, we present the experimental results in Section 2.2.4.

2.2.1 CUI Curve Computation

We assume that our neural network model is pre-trained on a given training set. The CUI curve is computed on some validation set composed of C classes, each formed by N_C images. For each c-th class, we do the following.

For a given i-th layer of the neural network, $i = 1, \ldots, I$, we extract the class-discriminative activation map $L^i_{j,c}$ for each j-th image belonging to class $c \in C$. For this sake, we start by computing the feature map importance coefficient of Grad-CAM $\alpha^c_{i,j}$:

$$\alpha^c_{i,j} = \frac{1}{z} \sum_n \sum_m \frac{\partial y^c}{\partial F^{i,j}_{n,m}} \,, \tag{2.1}$$

where $F^{i,j} \in R^{n \times m \times z}$ is the feature map of the convolutional layer i for the image j.

The weight $\alpha^c_{i,j}$ represents a partial linearization of the deep network downstream of F and captures the value of the feature map of the i-th layer for a target class c. Then, we perform a weighted sum between the just calculated value and the feature maps $F^{i,j}$ of the chosen layer. Finally, Rectified Linear Activation Unit (ReLU) activation function is applied to reset the negative values of the gradient to zero, obtaining the class activation map $L^i_{j,c}$ for a specific query image j:

$$L^i_{j,c} = ReLU \left(\sum_{k=i}^{I} \alpha^c_{k,j} F^{k,j} \right) \,. \tag{2.2}$$

This represents the analog of the saliency map of the standard Grad-CAM [41], which, instead of being computed on the input image, is focused on the i-th layer, summing from the class activations of the networks back until i.

Now we aim to obtain a single value for our class activation map $L^i_{j,c}$: thus, we sum over the dimensions of $F^{i,j} \in R^{n \times m \times z}$, obtaining the per-image CUI values $\mathrm{CUI}^i_{j,c}$. Ideally, computing these values for each i-th layer of the network gives a curve showing how the image has triggered the different layers of the network (see Figure 2.3, the pool of curves below each class). At this point, averaging all images from all classes provides our final cumulated importance curve CUI (Figure 2.3 the yellow curve on the right), where the i-th point CUI^i of the curve is a surrogate of the information conveyed through the i-th layer towards the decision for the correct class, for all classes into play. It is worth noting that, possibly, one may limit oneself to computing the CUI curve only for specific classes (or a set of classes), obtaining per-class curves CUI^i_c.

2.2.2 Split Application

Once the CUI curve has been computed, candidate splitting points can be individuated by *i)* choosing the layer that gives the highest peak or *ii)* selecting layers that give local CUI maxima if other constraints than the accuracy have to be taken into account.

Let T^i be the target layer for splitting the model at index i and T^{i+1} the subsequent layer. We divide the network into three main blocks: the head, running on the edge device, is composed of the first layers of the original DNN architecture, up to layer T^i; the bottleneck, an undercomplete Autoencoder (AE) [54] that learns low-dimensional latent attributes which explain the input data; and the tail, from layer T^{i+1} to the very end of the network, which is executed on the server side. The encoder part of the bottleneck is deployed to the edge device, while the decoder is executed on the server side. Encoders and decoders use both spatial and channel-wise reduction/restoration units, respectively.

2.2.3 Re-Training Strategy

In order to train the entire model M, we first train our bottleneck, and then we fine-tune end-to-end the entire network. We create an undercomplete AE, which acts as a bottleneck inserted after T^i. This bottleneck should learn to replicate the input, which is the feature map in the output at layer T^i. Therefore, given $\{I_j, j = 1,2,...,n\}$ as the n number of training data, we train the sole bottleneck freezing the remaining network with the following loss:

$$\mathscr{L}_{AE} = \frac{1}{n}\sum_{j=1}^{n} ||\Phi_{T^i}(I_j) - \Psi(\Phi_{T^i}(I_j); W_{AE})||^2 , \tag{2.3}$$

with Φ_{T^i} the model layers up to the i-th layer T^i (target layer), Ψ is the AE part of the model with weights W_{AE}.

After training the bottleneck, we perform a fine-tuning of the network with the following loss function:

$$\mathscr{L}_{task} = \frac{1}{n}\sum_{j=1}^{n} ||\Phi_M(I_j; W_M) - \hat{y}_j||^2 , \tag{2.4}$$

with Φ_M the full M's DNN layers, W_M the weights of the model M and $\hat{y}_j$ the ground truth label for the image j.

2.2.4 Experimental Results

In this section, we show how **I-SPLIT** individuates the split points by the CUI curve so that the associated CUI values are predictive of the classification accuracy when the network is split at those points.

In all the experiments, the bottlenecks injected in the split networks are standard convolutional undercomplete autoencoders with two convolutional layers, with stride 2 for the encoder and two layers for the decoder. The number of filters depends on the desired compression rate: specifically, we keep the compression rate constant at 90%, which means that the dimensionality of the encoded data is 10% of the dimension of the splitting layer.

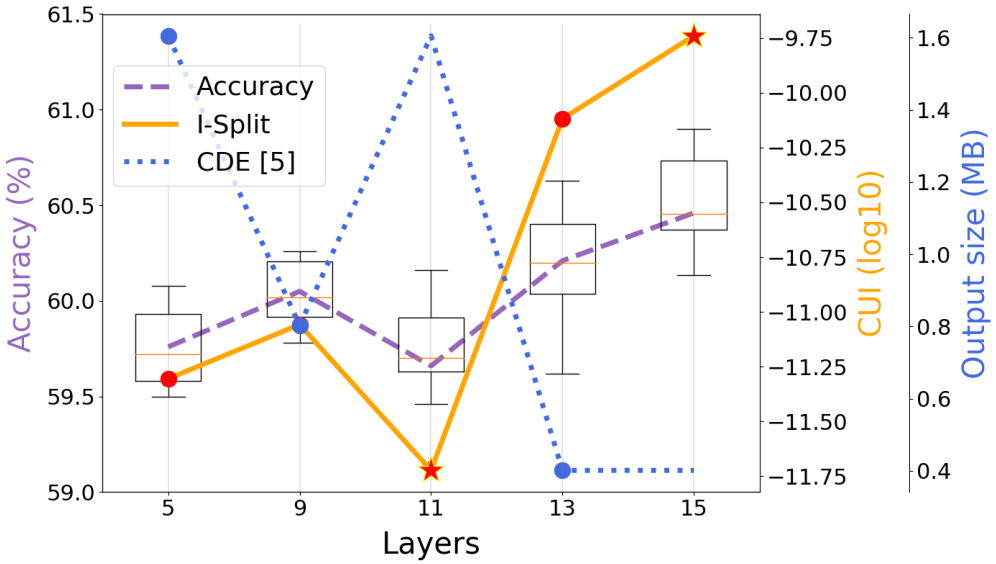


Figure 2.4 Candidate splitting points identified by CDE and the CUI curve of **I-SPLIT** on the VGG16. In dashed purple, the accuracy values are obtained by splitting the network at that point, re-training it, and testing on a validation set. The "star" markers show the extra points that our method is able to identify and that were not identified by the CDE approach.

Splitting Point Analysis

Since our framework is agnostic with respect to the network architecture, we consider the VGG16 and ResNet-50 networks, and most of the experiments are run on the first one. In addition to being very popular architectures, they are also interesting for SC: they exhibit a large number of parameters and good depth in terms of the number of layers, with a dramatic variation in the dimensionality of the layers as the depth increases.

The first experiment focuses on the Tiny-Imagenet-200 dataset [55], a compact version of ImageNet that comprises a subset of 200 classes, with a training set of 100,000 images, and a test set of 10,000 images each. All images are 64×64 pixels.

As a comparative approach, we consider CDE [40], which identifies splitting points, whereas there is a decrease in the size of the layers (see Section 2.1 for details). The multi-axis Figure 2.4 reports the CDE curve in blue, with the markers indicating the candidate split points. Our **I-SPLIT** is reported with the curve in solid yellow, with the red markers identifying the local maxima of the curve CUI to simplify visualization. Importantly, we also report the accuracy curve, in dashed purple, obtained by retraining the network after splitting it at the selected locations.

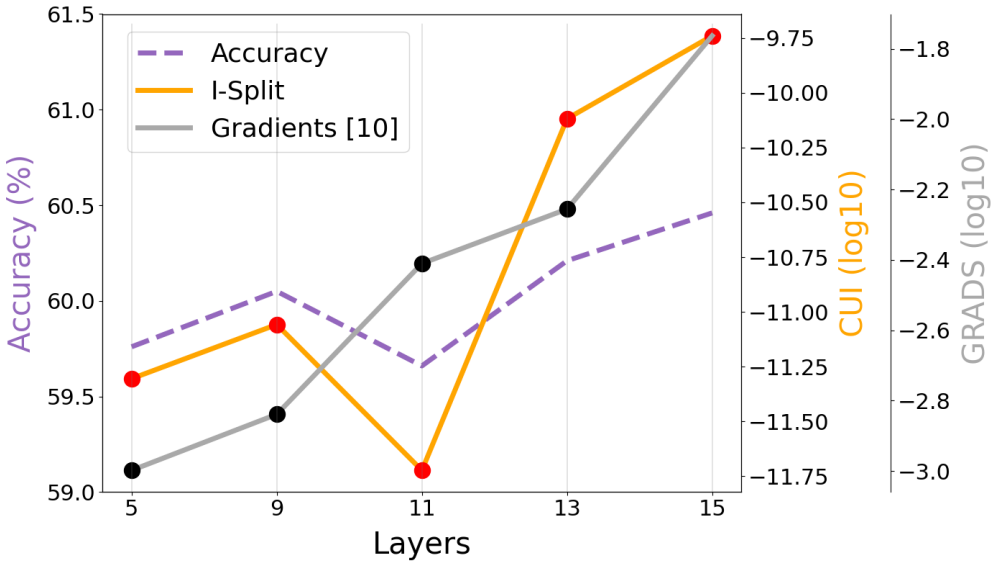


Figure 2.5 The sensibility of our **I-SPLIT** method to the chosen interpretability approach: in gray the CUI using the *Gradients* approach. In yellow, Grad-CAM. As visible, Grad-CAM gives a better curve in terms of proportionality w.r.t. the final class accuracy, while *Gradients* is growing as soon as the layer is deeper.

Several facts do emerge. *i)* Our **I-SPLIT** identifies all candidate splitting points output by CDE (namely layers 5, 9 and 13, corresponding to *block2_pool*, *block3_pool*, and *block4_pool*, respectively): these are max-pooling layers in the VGG architecture, which are worth conveying more information-per-pixel since there is a local dimensionality reduction with limited loss of information. *ii)* **I-SPLIT** finds two additional points at layers 11 and 15, or *block4_conv2* and *block5_conv2*, which do not correspond to a decrease in the layer size. *iii)* Looking at the accuracy curve, the significance of the CUI curve becomes apparent: all candidate splitting points are proportional to the post hoc accuracy obtained by the split network. At the same time, CDE does not exhibit such behavior at all. *iv)* While the absolute values of the classification accuracy are relatively close to each other, we provided a statistical analysis that generated 15 test sets by randomly sampling 800 images from the validation set. The box plots in Figure 2.4 demonstrate how the distributions are well separated and thus the relatively small improvement in accuracy is statistically relevant.

I-SPLIT strongly relies on the Grad-CAM interpretability approach. In order to evaluate the efficacy of another interpretability technique if injected in **I-SPLIT**, we perform a second experiment. In Figure 2.5, we report the result obtained with

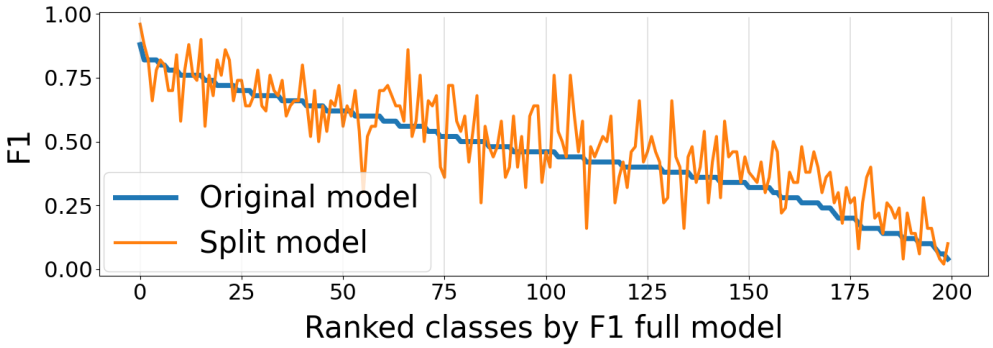


Figure 2.6 The behavior of **I-SPLIT** on preserving the per-class performance before the split (original model, in blue) and after the split (split model, in orange). As expected, no dramatic changes in the single performance are visible, especially with the best and worst performing classes: the best performance does not become the worst after the split, and vice-versa.

Gradients, the other approach that together with Grad-CAM passes the "sanity check" of the interpretability approaches [42]. In practice, this approach amounts to removing from Equation (2.2) the feature map variable $F^{k,j}$. This brings us to a sort of CUI curve, where we sum the gradients associated with a given input without multiplying it by the features themselves.

As visible, Grad-CAM gives a better curve in terms of the proportionality w.r.t. of the accuracy of the final class, while *Gradients* increases as the layer becomes deeper, making it difficult to spot the best-split point. This is due to the vanishing gradient effect: gradients are much higher when closer to the network's end. For this reason, we prefer Grad-CAM since the multiplication of the gradients by the values of the feature (close to 0) provides a more restricted range of values of CUI.

The third experiment investigates how the classification performance of a DNN changes once it is divided by our approach. In particular, we focus on the performance of every single class. As a split location, we select the layer 15, where the CUI curve has the highest peak. After training the original VGG network, we rank the 200 classes in descending order F1 (the blue line of Figure 2.6). As visible, performance is unbalanced since the F1-scores range from a few decimals to 87%. In the (re-trained) split version, we calculate new F1-scores, reporting them in the figure while keeping the previous class ranking to highlight possible variations in the performance (the orange line of Figure 2.6). The plot is very interesting: in fact, the classes that performed the best or worst before the split did not significantly change their performance after the split, providing F1-scores that are still high/low. For

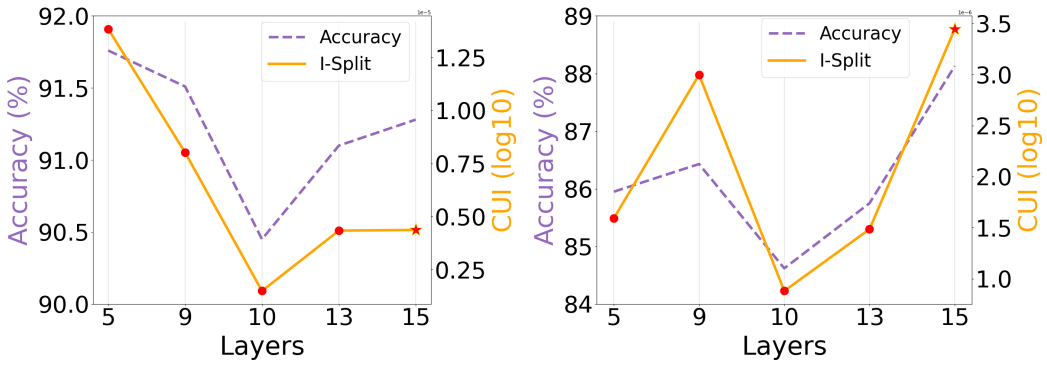


Figure 2.7 The efficacy of **I-SPLIT** on different datasets. Both the figures show the CUI curve of **I-SPLIT** (solid yellow) and the accuracy when splitting between a particular layer, retraining, and testing (dashed purple). On the left, the notMNIST [19] dataset. On the right, the Chest X-Ray Pneumonia [20] dataset. In both cases, the CUI curve is shown to be proportional to the accuracy.

classes with average performance, the split slightly shuffled the F1-scores. Overall, a certain degree of variability is expected, since *i)* the CUI curve indicating the optimal split point requires a summation over all classes, so the per-class performance cannot be precisely preserved. *ii)* It seems that for specific classes, the optimal splitting point may be different. We will focus on this aspect in the second part of the experiments.

The fourth and fifth experiments check how **I-SPLIT** will perform with datasets with fewer classes, possibly unbalanced in terms of cardinality. For this reason, we apply our method to notMNIST [19] (10 classes) and Chest X-Ray Pneumonia [20] (2 classes, unbalanced). The notMNIST dataset is focused on the task of digit recognition with heterogeneous fonts and graphics, while Chest X-Ray focuses on classifying pneumonia cases from chest X-rays. The Chest X-Ray is unbalanced (cases: 3,883, controls: 1,349), and this is a further challenge to the robustness of **I-SPLIT**.

In both cases, the deep network employed is the VGG16. The results are reported in the form of CUI curves in Figure 2.7, paired with a posteriori accuracy. The CUI curves are proportional to the accuracy in both cases. This further promotes **I-SPLIT** as a prognostic tool toward a split configuration, which maximizes the accuracy of the classification.

On the ResNet-50 architecture [56], we show the CUI curve, the CDE comparative approach, and the post-split accuracy curve (Figure 2.8). As visible, the local maxima of the CUI curve (indicated by the red markers) indicate plausible split locations.

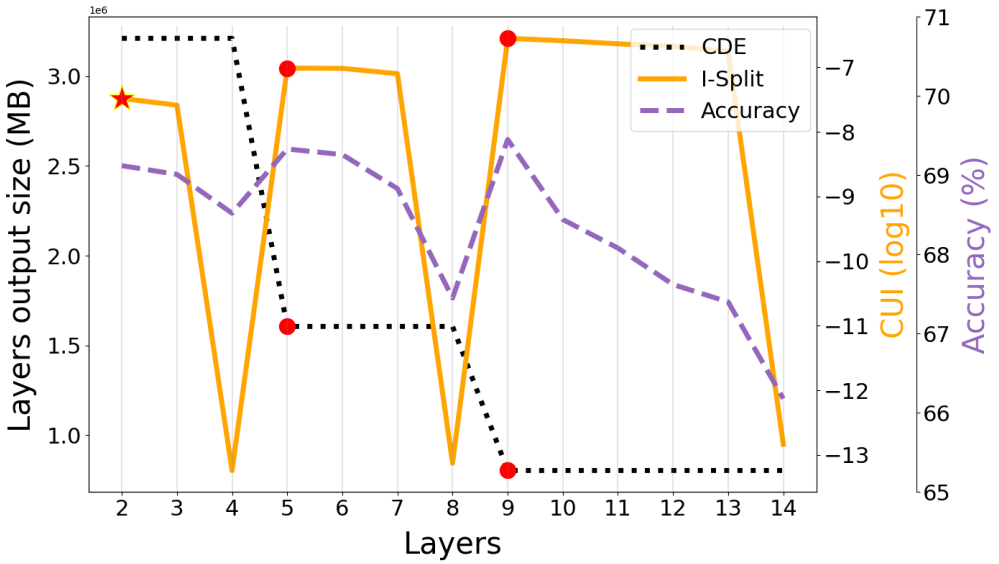


Figure 2.8 Results of the CDE, **I-SPLIT**, and accuracy on the ResNet-50. In particular, the local minimum points of CDE correspond to the local maximum points of **I-SPLIT**, and the CUI curve shows to be once again proportional to the accuracy. The "star" markers show the extra points that our method is able to identify and that were not identified by the CDE approach.

In general, the CUI curve follows the slope of the accuracy, even showing, in this case, nice predictive properties. Furthermore, we can observe that the CUI is almost constant in layers with the same dimensionality of the features, but it is also relatively uniform between different blocks.

Class-Dependent Splitting Point Selection

In this work, we also claim that different subsets of classes can lead to different selections of the best splitting point, as guessed in the previous section. To validate this claim, we analyzed the **I-SPLIT** curve for each class separately and identified two different behaviors in the last portion of the network. In particular, when referring to *block5*, we saw that most of the classes achieved a higher CUI at layer 15, while 15 classes had a global maximum at layer 13 (see Figure 2.9-top). We investigated this phenomenon by generating two alternative models by splitting the network at both points and then retraining these models with two subsets of 15 classes, one for each different behavior. The results, shown in Figure 2.9-bottom, demonstrate that the same shape of the **I-SPLIT** curve translates to the accuracy curve.

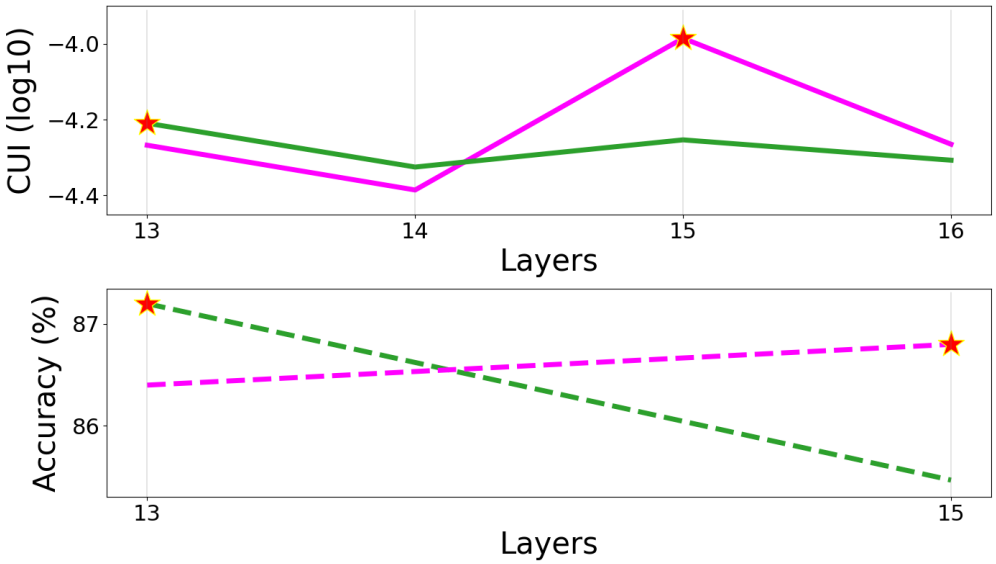


Figure 2.9 The comparison of splitting results for different subsets of classes. Top: the CUI curve computed on two different subsets of 15 classes of the VGG16 architecture. Bottom: the classification accuracy for two models trained on the same subsets of 15 classes on layers 13 and 15.

2.3 Split-Et-Impera

Section Organization

In the following, we first present the **Split-Et-Impera** framework (Section 2.3.1). Then, we present the experimental results in Section 2.3.2.

2.3.1 The Split-Et-Impera Framework

The architecture of **Split-Et-Impera** is presented in Figure 2.10. To offer a large state-space of the model, it is divided into five main layers: *supervisor*, *sensing*, *transmitter*, *netsim*, and *receiver*. The network simulation is based on the SCNSL [57] library, while the simulator is implemented in Python.

The review provided in Section 2.1 led us to aim to develop a communication-aware simulation platform for distributed deep learning applications that accurately emulates the behavior and timing of the computation, communication, and inference performed by the system. The simulator has to be modular, portable, and language

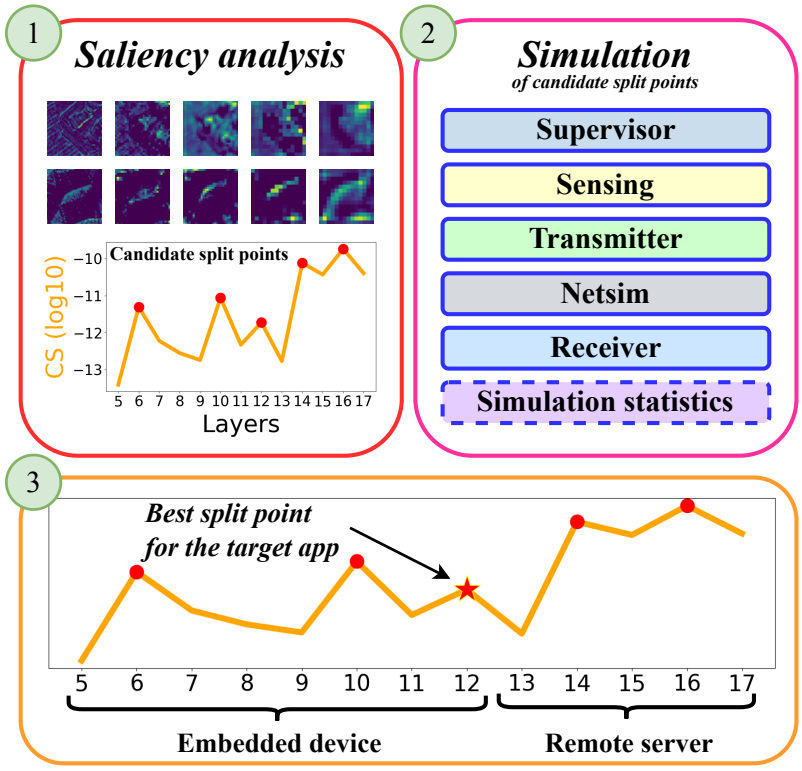


Figure 2.10 The **Split-Et-Impera** framework. *i)* Determine the saliency-based split point candidates. The inputs are fed into a neural network to extract the saliency maps using the Grad-CAM algorithm at each layer. Then, we average over all the maps to generate the final Cumulative Saliency (CS) curve with the candidate split points. *ii)* Simulate each candidate by reproducing its computation and its transmission. *iii)* Use the simulation statistics to find the best-split point satisfying the application constraints on accuracy and latency.

independent, *i.e.*, the functional blocks of the framework should be integrated into the model independently of their implementation programming language. Furthermore, it must allow the integration of real-world components, such as a real computing system: the so-called Hardware-In-The-Loop (HIL) [58].

The *supervisor* controls all the events and operations that occur during the simulations. The *sensing* layer is a high-level wrapper that encodes the application in the architecture. The *transmitter* module is responsible for the XMTR. The *netsim* must guarantee that the simulation emulates the behavior of an actual communication channel: it has to execute events in the correct temporal order while taking into account the physical features of the channel, such as propagation delay, and signal loss. Finally, the *receiver* is responsible for the RCVR.

Specifically, **Split-Et-Impera** requires the following inputs:

1. **Test scenario:** LoC, RoC, or SC (as described in Section 2.1).
2. **Training set:** The set of data used to train the model and make it learn the hidden features/patterns in the data.
3. **Trained instance of a DNN:** The network parameters once it has been trained on a general purpose machine.
4. **Test set:** The data with which the simulation is run; is a proxy of the real setting on which the framework will work.
5. **Communication network modeling:** Through some parameters which will be explained in the following.

The communication network modeling parameters are the following.

1. **Communication protocol:** The transport layer protocol that must be used, *i.e.*, TCP, or UDP.
2. **Channel latency:** The time it takes for each packet to travel from the sender to the receiver, *e.g.*, 100 μs.
3. **Channel capacity:** The bandwidth available in the link.
4. **Interface speed:** To better model different hardware devices, even the physical speed can be set to match the desired target hardware, *e.g.*, 1000 Mb/s to represent a Gigabit connection, 100 Mb/s for Fast-Ethernet, 160 Mb/s to represent Wi-Fi, etc.
5. **Saboteur:** The network loss rate.

The output of **Split-Et-Impera** is of two types: *i)* the suggested configurations to simulate, ranked by the classification accuracy that the network is assumed to achieve. The engineer may then decide to simulate all or only a subset of them. *ii)* The simulation results of the configurations selected in the previous step to deploy the application using the best design.

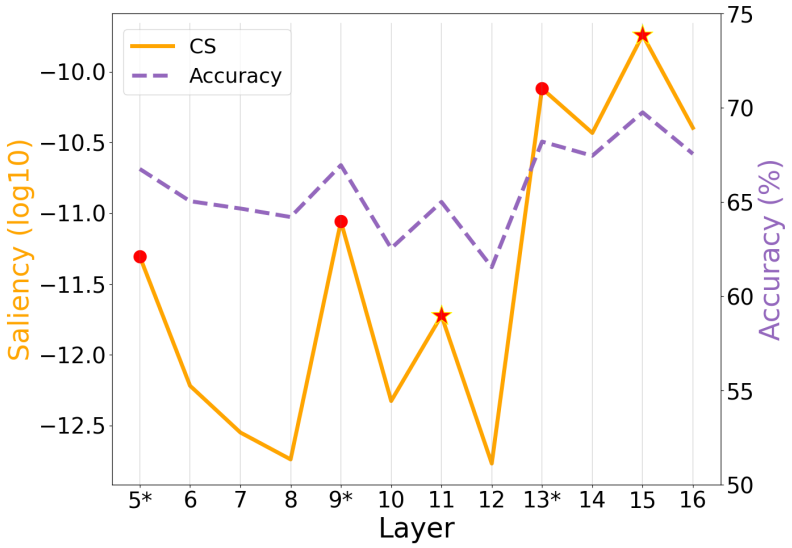


Figure 2.11 *Cumulative Saliency* (CS) as a function of the layer compared with the accuracy of the DNN split in that layer. The peaks of the CS curve correspond to the points in which accuracy is preserved despite split injection. Thus, the layers in which CS has a local maximum are the best candidates for splitting. (*) represents a max pooling layer.

2.3.2 Experimental Results

In this section, we show how the design of a typical real application can be carried out by **Split-Et-Impera**. In particular, we focus on the classification task of children's toys (such as boats, airplanes, etc.) passing on a conveyor belt within a real Industry 4.0 scenario: the ICE Laboratory at the University of Verona.

Experimental details. As DNN, we use the PyTorch implementation of the VGG16 [59]. In all experiments, the split has been realized by placing an undercomplete autoencoder with a compression rate 50%. For training of the encoder/decoder pair, we run 50 epochs with a learning rate of 5×10^{-4}, always using Adam [60] as optimizer.

Saliency-Based Split Point Search

The first output of **Split-Et-Impera** is a set of candidate split points that are worth being simulated. Figure 2.11 shows the results of the saliency-based split point search.

We can clearly appreciate how the CS saliency approach identifies as candidate split points (in red) the layers 5, 9 and 13, corresponding to *block2_pool*, *block3_pool* and *block4_pool* (dense data), and two additional points at layers 11 and 15 (informative data), corresponding to *block4_conv2* and *block5_conv2*, respectively. It is worth noting how layers with the same dimensionality, *i.e.*, convolutional layers belonging to the same block, do not express the same importance as shown by the CS curve. In these regions of uniform dimensionality, we select layers that are of greater importance and will show that the CS values directly translate into higher classification accuracy. Figure 2.11 confirms that CS is a good proxy for the overall classification accuracy, and therefore it is worth splitting the network into the layers in which CS exhibits a local maximum.

Given the results of the CS computing procedure, which we recall is done without having retrained the network, then we can explore the complete behavior of the application, including its transmission aspects, just splitting into the candidate layers. In particular, in the next section, we highlight simulation results only after splitting at layers 11 and 15.

Communication-Aware Split Point Selection

Figure 2.12 refers to the experiment of the SC scenario for the classification task of the images captured inside the ICE Laboratory. The network channel is 1 GB/s Full-Duplex with the TCP protocol. The application presents a constraint on the maximum frame latency of 0.05 s (*i.e.*, 20 FPS), given by the speed of the conveyor belt. The split point is in layers 11 and 15, respectively.

Figure 2.12 highlights how the latency increases with the packet loss rate due to TCP retransmission in the case of packet loss. However, this preserves the accuracy of the application. In particular, the solid pink curve shows that with the split in layer 15, the application requirements are always satisfied independently of the packet loss rate. Vice versa, the dotted blue curve shows that with a split in layers 11, the 20 FPS constraint cannot be satisfied when the packet loss rate is greater than 3%. The behavior of the simulator meets expectations *i.e.*, by splitting the network at layer 11, the amount of transmitted data is greater than that obtained by splitting the network at layer 15, and due to retransmissions, the latency increases, up to violating the application constraints represented by the dashed red line.

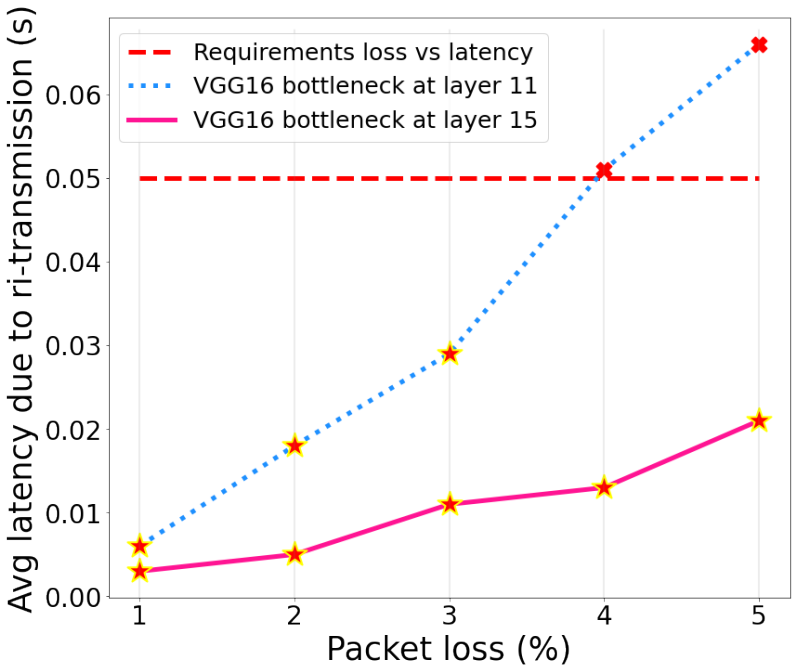


Figure 2.12 Evaluation of the impact of the network on split point selection for the classification task of the images captured inside the ICE Laboratory. The network channel is 1 GB/s Full-Duplex with TCP protocol.

This experiment shows the main claim of our work: given a set of possible split-point solutions (Figure 2.11), our framework through a rapid evaluation allows deciding which DNN configuration is compatible with the application requirements while considering the communication setup.

Network Protocol Selection

Figure 2.13-left and Figure 2.13-right show the system accuracy and the overall latency, related to the classification task in the full RoC scenario, this time on the CIFAR-10 [61] test set, with the use of the TCP and UDP protocols. The network channel is 1 GB/s Full-Duplex.

Figure 2.13-left shows that, using TCP, the application accuracy does not depend on the packet loss rate. However, Figure 2.13-right shows that this feature comes at a cost: with TCP, the overall latency is much greater, so it is necessary to make sure that this is compatible with the application requirements. UDP protocol shows dual behavior; the latency is minimized and kept independent of the packet loss rate,

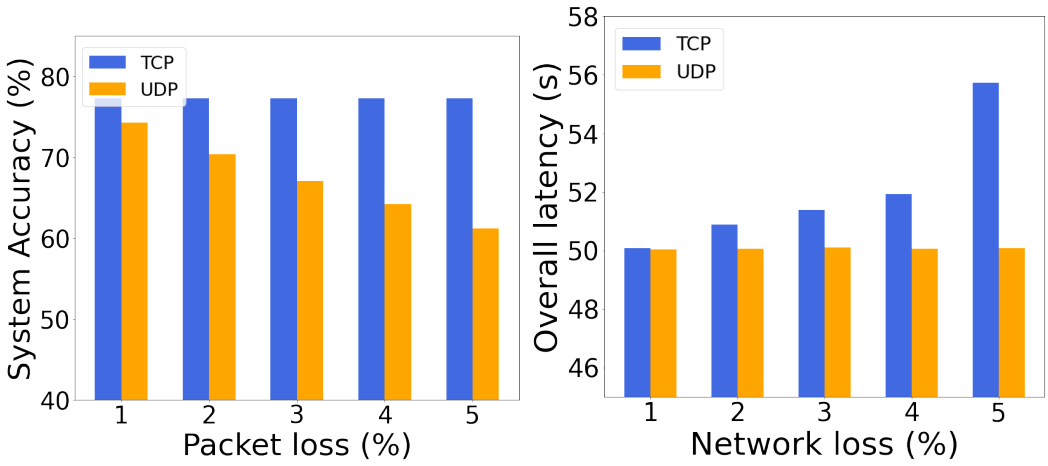


Figure 2.13 Accuracy (left) and latency (right) results on the CIFAR-10 test set for the classification task, contextualized in the RoC scenario with the use of the TCP and UDP protocols. The network channel is 1 GB/s Full-Duplex.

but the accuracy decreases in case of loss because no error checking and recovery services are provided.

These experiments show that the proposed framework allows modeling an application's transmission details to jointly perform split point selection and transmission protocol selection.

2.4 MTL-Split

This solution proposes to combine SC and MTL to execute complex inference tasks on edge devices. After outlining our notation, this section delineates the formal components of our proposal, shown in Figure 2.14. This architecture consists of two components: *i)* a shared backbone deployed on the edge device, and *ii)* a series of task-solving heads on a single or multiple remote devices. Orange trapezoids are DNN models, while their parameters are enclosed in red boxes. The green components on the right are the loss functions that are used to update the learnable parameters. A communication network separates edge devices and remote devices.

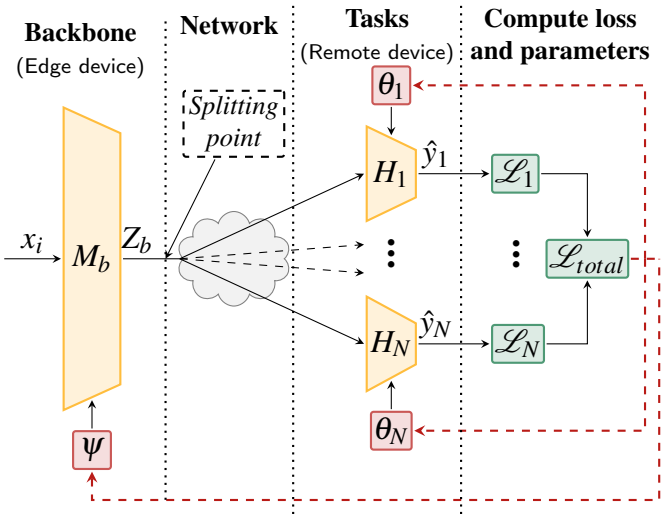


Figure 2.14 The proposed architecture for handling complex inference tasks on edge devices by integrating SC and MTL.

Setting and notation. We assume the existence of a labeled image dataset defined as follows:

$$D = \left\{ (x_i, y_i) \,|\, \forall i \in \{1 \dots K\}, x_i \in \mathbb{R}^{w \times h \times c}, y_i \in \mathbb{N}^N \right\} , \tag{2.5}$$

where K is the number of (image, labels) tuples, x_i is the input representing the image, and y_i a set of N labels associated with the i-th image, namely ground truth. The input x_i is a tensor with dimensions $w \times h \times c$, where w is the width, h is the height, and c is the number of channels (*e.g.*, red, green, blue). In this work, we consider the classification task that attempts to learn a mapping from the image space $\{x_i | \forall i \in \{1 \dots K\}\}$ to the corresponding set of labels $\{y_i | \forall i \in \{1 \dots K\}\}$.

Section Organization

In the following, we first provide a detailed explanation of the proposed **MTL-Split** architecture (Section 2.4.1). Next, we discuss the training strategy (Section 2.4.2) and outline the model fine-tuning process (Section 2.4.3). Finally, we present the experimental results in Section 2.4.4.

2.4.1 Proposed Architecture

Unlike a classic SC scenario, here we focus on architectures operating through a DNN model, whose task is to produce the inference outputs $\{\hat{y}_j | \forall j \in \{1 \dots N\}\}$ from an input x_i, where N is the number of tasks to be solved. In this way, one can build a single model that learns multiple tasks across the same input.

As shown in Figure 2.14, the first module is the *backbone*, a DNN model $M_b(\cdot)$ that shares hidden layers between all tasks. In this way, we greatly reduce the risk of overfitting: the more tasks we learn concurrently, the more our model has to find a representation that captures all tasks, and the less likely we are to overfit the original task. We describe the backbone operations on the i-th input as follows:

$$Z_b = M_b(x_i; \psi) \ , \tag{2.6}$$

where ψ are the set of shared learnable parameters, and Z_b is the backone's output. The output Z_b is typically a tensor that, in our approach, is flattened before being sent through the network. This output represents the point of the networks where the shared feature representation Z_b is extracted from the backbone and transferred to the task-solving heads.

Each task is implemented by its own *task-solving head*, or head hereafter, a DNN model H_j located outside the edge devices, *e.g.*, on a remote server. We describe the operations of the j-th head H_j, as follows:

$$\hat{y}_j = H_j(Z_b; \theta_j) \ , \tag{2.7}$$

where θ_j is its set of learnable parameters, and $\hat{y}_j$ its output.

Putting it all together, the overall system output is a collection of outputs from all heads, organized either as a list or as a single tensor.

2.4.2 Training Strategy

The proposed methodology is architecture-independent. Any neural network architecture can implement the backbone network and heads, such as a Convolutional Neural Network (ConvNet) or a Recurrent Neural Network (RNN), designed to capture useful features from the input data x_i. Regardless of the desired architecture, the

objective of the MTL system is to encourage the model to perform well on all tasks simultaneously. Let us denote the *task-specific loss function* for the i-th input and j-th task as $\mathscr{L}_j(y_i, \hat{y}_j)$, which measures the differences between the corresponding label inside the ground truth y_i and the predicted output $\hat{y}_j$. The *overall loss function* for the MTL system with the i-th input can be defined as the sum of losses from each task, as follows:

$$\mathscr{L}_{total} = \sum_{j=1}^{N} \mathscr{L}_j(y_i, \hat{y}_j) \ , \tag{2.8}$$

The training process updates the shared backbone parameters ψ and the head parameters θ_j by backpropagating the total loss gradient with respect to these parameters and using an optimization algorithm like Stochastic Gradient Descent (SGD). The specific DNN architecture, the activation functions, and the optimization methods can vary depending on the problem and the input data.

2.4.3 Fine-Tuning the Model

A key aspect of the proposed methodology, in addition to exploiting SC to enable MTL in edge devices, is the fine-tuning process explained in this section. There are several reasons for performing fine-tuning, such as if we aim to enhance task-specific performance or if we want to introduce new tasks to the system. During the fine-tuning phase, we update the heads' parameters θ_j while keeping the shared backbone parameters ψ relatively fixed.

During the fine-tuning process, heads' parameters are updated using gradients with respect to the task-specific loss:

$$\theta_j := \theta_j - \alpha \cdot \nabla_{\theta_j} \mathscr{L}_j(y_i, \hat{y}_j) \ , \tag{2.9}$$

where α is the learning rate for updating heads' parameters. The shared backbone's parameters are often kept fixed or updated conservatively during fine-tuning. As such, we need to define a separate update process as follows:

$$\psi := \psi - \eta \cdot \nabla_{\psi} \mathscr{L}_{total} \ , \tag{2.10}$$

where η is the learning rate for updating the shared parameters, a small value compared to the one used to update the heads' parameters shown in Equation (2.9).

Given the parameters update functions, we can now define the fine-tuning process as an optimization problem, which involves minimizing the sum of the total loss as follows:

$$\underset{\theta,\psi}{\text{minimize}} \ \mathscr{L}_{total} \ , \tag{2.11}$$

It is worth mentioning that fine-tuning should be approached with care. It is important to find a balance between adapting the model to task-specific characteristics and retaining general knowledge of the shared backbone. Too much fine-tuning can lead to overfitting on limited task-specific data, while too little fine-tuning might not fully harness the benefits of the MTL setup.

2.4.4 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Models details. In our experimental setup, we used three well-known DNNs, *i.e.*, VGG16 [59], MobileNetV3 [46], and EfficientNet [62], as a shared backbone. We chose the first one because it is a well-established and widely used architecture in many image processing tasks. The others represent cutting-edge DNNs for embedded system applications. The task-solving heads are custom Multi-Layer Perceptron (MLP) composed of two linear layers activated by the ReLU function. We point out that in this design, the task-solving heads are smaller than the backbone. However, even though task-solving heads are smaller than the backbone individually, if we consider a large number of tasks N, their combined size becomes larger than that of the backbone. Due to this rationale, our architecture provides the deployment of task-solving heads on the remote server.

Datasets descriptions. To effectively showcase the capabilities of our proposal, we begin our experiments with the 3D Shapes [63] dataset, a widely used toy benchmark in the ML literature. We further demonstrate the effectiveness of our proposed method by exploring its performance on the MEDIC [64] and FACES [65] datasets, one of the most well-known and newest MTL benchmarks, respectively.

3D Shapes is a dataset of 3D shapes generated from six independent factors. All possible combinations of these factors are present exactly once, resulting in 480,000 total images. These are the floor hue, wall hue, object hue, scale, shape, and orientation. Therefore, it is possible to treat the classification of each factor as a different task to solve, *i.e.*, $T = [T_1 \dots T_6]$. Due to the straightforward nature of the synthetic images in 3D Shapes, solving the classification tasks with a DNN can be easy, leaving a limited possibility of improvement through MTL. Thus, to make this setting more realistic, we add salt-and-pepper noise of 15% of the image pixels, making the classification more difficult. In particular, with the presence of noise, the classification of object size (8 classes) and object type (4 classes) becomes challenging.

MEDIC is the largest social media image classification dataset for humanitarian response, consisting of 71,198 samples to address four different tasks. Specifically, we decided to address only the damage severity (3 classes) and disaster type (4 classes) tasks, since information and humanitarian are somewhat trivial.

FACES is a set of 2,052 images of naturalistic faces. Here, the task corresponds to the classification of perceived ages (3 classes), genders (2 classes), and facial expressions (3 classes).

Training and inference details. All the code is implemented in PyTorch Lightning, and the used pre-trained network corresponds to the implementations in PyTorch [66]. On the 3D Shapes dataset, we train our models for 10 epochs, with a learning rate of 1×10^{-5}, using AdamW [67] as an optimizer, on a NVIDIA RTX 3090. On the MEDIC and FACES dataset, we train our models for 50 epochs, with a learning rate of 1×10^{-4}, always using AdamW as an optimizer, on an NVIDIA RTX 3090.

We run all the experiments on an NVIDIA Jetson Nano with a memory of 4 GB.

Multi-Task Learning Results

In this section, we validate our claims within the MTL context, evidencing the effectiveness of our proposal. Given the distinct nature of our methodology, a direct comparison with state-of-the-art MTL methods would be inadequate since these approaches are based on advanced loss functions [68] rather than models [69]. As

Table 2.1 Classification accuracy on the test partition of the 3D Shapes dataset considering the object size (T_1) and the object type (T_2). Values are reported as a percentage.

Model	Single-Task Learning		Multi-Task Learning (T_1 + T_2)	
	$T_1 \uparrow$	$T_2 \uparrow$	$T_1 \uparrow$	$T_2 \uparrow$
VGG16	12.50	25.50	**51.10** (+38.60)	**81.74** (+56.24)
MobileNetV3	74.85	93.95	**77.23** (+2.38)	**94.00** (+0.05)
EfficientNet	95.49	99.07	**96.66** (+1.17)	**99.48** (+2.28)

Table 2.2 Classification accuracy on the test set of the MEDIC dataset considering the damage severity (T_1) and disaster type (T_2). Values are reported as a percentage.

Model	Single Task Learning		Multi Task Learning (T_1 + T_2)	
	$T_1 \uparrow$	$T_2 \uparrow$	$T_1 \uparrow$	$T_2 \uparrow$
VGG16	61.78	59.14	**62.65** (+0.87)	**60.54** (+1.40)
MobileNetV3	61.73	52.66	**61.90** (+0.17)	52.29 (-0.37)
EfficientNet	61.00	53.94	**62.42** (+1.42)	**55.74** (+1.80)

a result, according to [43], our experimental protocol involves benchmarking our models against their respective single-task performance.

We begin our analysis with the 3D Shapes dataset. The results, in terms of accuracy, are shown in Table 2.1 and demonstrate that **MTL-Split** improves the performance on all tasks with respect to the choice of STL design. This confirms the first two claims of our proposal, *i.e.*, our architecture handles multiple tasks simultaneously and improves accuracy across the entire task set by collectively optimizing the model's parameters for all tasks. Hence, in the context of SC (which encompasses multiple tasks to be solved), our approach guarantees performance improvement rather than simply aiming to minimize performance degradation, which is the SC trend observed in all previous state-of-the-art methods. Furthermore, the ability to address multiple tasks within the same network simultaneously has resulted in space and computational savings during inference because it only requires the evaluation of a single network instead of N neural networks to solve each task.

Table 2.2 further demonstrates the efficacy of our proposal, exploring its performance on the MEDIC dataset. This experiment serves as a compelling validation of our previous claims, showcasing the robustness of our architecture even when

Table 2.3 Classification accuracy on the test set of the FACES dataset considering the perceived ages (T_1), genders (T_2), and facial expressions (T_3). Values are reported as a percentage.

Model	Single Task Learning			Multi Task Learning ($T_1 + T_3$)		Multi Task Learning ($T_2 + T_3$)		Multi Task Learning ($T_1 + T_2 + T_3$)		
	$T_1 \uparrow$	$T_2 \uparrow$	$T_3 \uparrow$	$T_1 \uparrow$	$T_3 \uparrow$	$T_2 \uparrow$	$T_3 \uparrow$	$T_1 \uparrow$	$T_2 \uparrow$	$T_3 \uparrow$
VGG16	96.83	95.61	19.02	**97.80** (+0.87)	**91.46** (+72.44)	**99.02** (+3.41)	**90.24** (+80.22)	**98.54** (+1.71)	**99.51** (+3.90)	**89.27** (+70.25)
MobileNetV3	97.07	99.51	95.12	**99.51** (+2.44)	95.12 (+0.00)	99.51 (+0.00)	**95.61** (+0.49)	**99.27** (+2.20)	99.51 (+0.00)	**95.85** (+0.73)
EfficientNet	99.76	99.76	94.63	**100** (+0.24)	**95.61** (+0.98)	99.76 (+0.00)	**97.32** (+2.96)	**100** (+0.24)	**100** (+0.24)	**95.61** (+0.98)

applied to complex datasets. In this case, it is important to highlight the inductive transfer between tasks, as even a small increase in decimal points in this challenging context represents a significant achievement.

The minor decrease of 0.37% of T_2's performance in the MTL setting does not represent a problem. Specifically, what is known in the MTL literature as negative transfer occurs when there is a significant deterioration in performance across all tasks, typically resulting from conflicting or unrelated task objectives. Since the performance improves in all the other cases, we can confidently say that negative transfer does not occur here. We attribute this result to gradient fluctuations. These results demonstrate the effectiveness of our approach, as it consistently yields significant improvements even in difficult scenarios.

Finally, Table 2.3 shows the results achieved on the FACES dataset using the fine-tuning strategy starting from pre-trained networks on ImageNet. The overall accuracies obtained are quite high, which was expected given the utilization of a pre-trained network as a starting point. However, once again, our approach demonstrated its efficacy in improving performance across all tasks. This is significant since it increases the accuracy by keeping it close to the maximum values. Usually, such improvements necessitate the network's ability to correctly classify the intricate corner cases within the datasets. In all instances where we do not achieve performance improvements, our results consistently align with the single-task performance (also in this case, ruling out the possibility that the non-performance improvements are due to negative transfer).

Table 2.4 Computing the size of the backbone M_b, and of its output Z_b. The reader should pay particular attention to the green columns in the table, as these are the columns displaying the results, which show that our proposal is really efficient for SC.

Model	M_b #params (M)	M_b #params size (MB)	F/B pass size (MB)	M_b est. size (MB)	Z_b #params (M)	Z_b size (MB)
MobileNetV3	0.9	3.58	724.08	727.66	55.3	0.21
EfficientNet	4	15.45	3452.09	3467.54	406.06	1.56

Split Computing Analysis

In this section, we examine the advantages of our approach in comparison to the other types of distributed deep learning paradigms, while also presenting deployment considerations.

Local-only Computing (LoC). Under this paradigm, for the 3D Shapes and the MEDIC, two distinct DNNs are required since we address two different tasks. Hence, the estimated memory size that uses MobileNetV3 as the backbone is ≈ 1.5 GB, while it is ≈ 6.9 GB for the EfficientNet. Instead, for FACES, which involves three different tasks (*i.e.*, three distinct DNNs are required), the estimated memory size using MobileNetV3 is ≈ 2.1 GB, and for EfficientNet is ≈ 10.3 GB.

As a result, due to memory constraints, the only feasible implementation on the Jetson Nano is restricted to MobileNetV3 on the 3D Shapes dataset. However, as indicated in Table 2.4, our approach, utilizing a single shared backbone on the edge device, enables the execution of all implementations on the same board. Specifically, using EfficientNet, we achieve memory size improvements of $\approx 38\%$ for the 3D Shapes and MEDIC datasets and $\approx 57\%$ for the FACES dataset. As mentioned above, VGG16 is not optimal for embedded system applications, so we do not report data on that model.

Remote-only Computing (RoC). Under this policy, the goal is to minimize the data sent from the backbone to task-solving heads.

In the FACES dataset, the images are RGB with 2835×3543 pixels. Consequently, transmitting each input from the edge to the cloud involves transefing a tensor of size $2835 \times 3543 \times 3$, equivalent to ≈ 115 MB over the network channel.

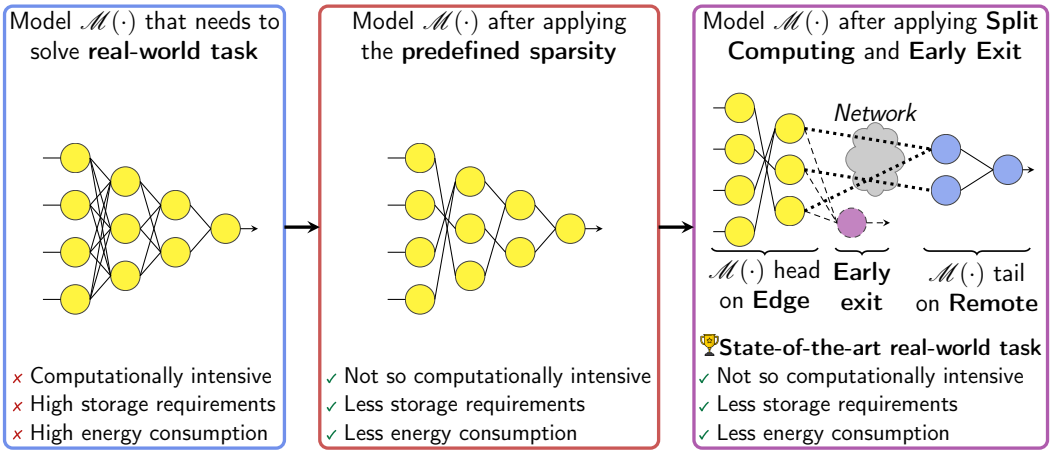


Figure 2.15 Starting from a DNN $\mathscr{M}(\cdot)$, we first apply the predefined sparsity, and then we train the network. After the training stage, we split the network following the SC and EE paradigm. As a result, the final architecture is not so computationally intensive, does not require huge storage spaces, and has less energy consumption, all without compromising the overall performance.

However, Table 2.4 also highlights the minimal burden placed on the network channel when employing our methodology, thanks to the neural processing of the shared backbone. For example, assuming a gigabit channel: the time required to transfer 100 inputs of size $\approx 115\,\text{MB}$ each is $\approx 98\,\text{s}$, whereas for our inputs of size $1.5\,\text{MB}$, it is $\approx 12\,\text{s}$, *i.e.*, we obtain an improvement of $\approx 87\%$ in the overall latency time. This is important as the Internet congestion will increasingly be driven by ML workloads.

This claim is of significant importance in a world that is increasingly dependent on efficient data transmission and reduced network congestion.

Discussion

The above analyses show the advantages of our proposal compared to LoC. Our approach also excels in terms of data transmission, resulting in reduced total latency compared to RoC. Furthermore, our design handles multiple tasks concurrently, thus enhancing overall accuracy across all tasks.

2.5 Sparsity-SC&EE

With this solution, our objective is to show how, in the context of SC and EE, predefined sparsity can significantly reduce computational demands, storage requirements, and energy consumption compared to state-of-the-art approaches. Furthermore, regardless of the underlying implementation platform, our approach yields advantages for both the training and the inference stages. In particular, our pipeline is exemplified by the flow presented in Figure 2.15.

Section Organization

In the following, we first provide the mathematical background behind **Sparsity-SC&EE** (Section 2.5.1). Then, we present the experimental results in Section 2.5.2.

2.5.1 Mathematical Background of Predefined Sparsity

To understand the core concepts of our research, let us take a look at the mathematical background. These concepts were initially presented in [70].

Let us take a $(L+1)$-layer MLP, described collectively by the following *neuronal configuration*:

$$N_{net} = (N_0, \ldots, N_{i-1}, N_i, \ldots, N_L) \,, \tag{2.12}$$

where N_i represents the number of nodes in the i-th layer. We use the convention that layer i is to the right of layer $i-1$. Given L *junctions* between layers, with junction i connecting the N_{i-1} nodes of its left layer $i-1$ with the N_i nodes of its right layer i.

We can define *predefined sparsity* as simply not having all $N_{i-1} \cdot N_i$ edges (or weights) present in the junction i. Furthermore, we can define *structured predefined sparsity* so that for a given junction i, each node in its left layer has fixed out-degree, *i.e.*, $d_i^{out} \in \mathbb{N}$ connections to its right layer, and each node in its right layer has fixed in-degree, *i.e.*, $d_i^{in} \in \mathbb{N}$ connections from its left layer.

In particular, a MLP has $d_i^{out} = N_i$ and $d_i^{in} = N_{i-1}$ with $N_{i-1} \cdot N_i$ edges present at the junction i^{th}. However, a sparse MLP has at least one junction with less than this

number of edges. The number of edges in junction i is given by the formula:

$$|W_i| = N_{i-1} \cdot d_i^{out} = N_i \cdot d_i^{in} \ . \tag{2.13}$$

The density of junction i is measured relative to MLP, and is denoted by the function:

$$\rho_i = \frac{|W_i|}{N_{i-1} \cdot N_i} \ . \tag{2.14}$$

In our structured predefined sparse network, the density of the i-th junction ρ_i cannot be arbitrary. By replacing Equation (2.13) in Equation (2.14), we can define:

$$\rho_i = \frac{d_i^{in}}{N_i} = \frac{N_i}{d_i^{out}} \ , \tag{2.15}$$

where d_i^{out} and d_i^{in} are natural numbers. The number of possible ρ_i values is the same as the number of (d_i^{out}, d_i^{in}) values satisfying the structured predefined sparsity constraints:

$$d_i^{out} = \frac{N_i \cdot d_i^{in}}{N_{i-1}} \ , \qquad d_i^{in} \leq N_{i-1} \ . \tag{2.16}$$

Now, the smallest value of d_i^{in} which satisfies the assignment to d_i^{out} in Equation (2.16), and $d_i^{out} \in \mathbb{N}$, is the following:

$$\frac{N_{i-1}}{\gcd(N_{i-1}, N_i)} \ , \tag{2.17}$$

and other values are integer multiples. Since d_i^{in} is above the bound of N_{i-1}, the total number of possible (d_i^{out}, d_i^{in}) is $\gcd(N_{i-1}, N_i)$. We can now define the set of possible densities ρ_i, as follows:

$$\left\{ \rho_i = \frac{k}{\gcd(N_{i-1}, N_i)} \ , \quad \rho_i \in (0, 1] \ , \quad k \in \mathbb{N} \right\} . \tag{2.18}$$

Specifying N_{net} and the *out-degree configuration* $d_{net}^{out} = (d_1^{out}, \ldots, d_L^{out})$ determines the density of each junction and the overall density, defined as:

$$\rho_{net} = \frac{\sum_{i=1}^{L} |W_i|}{\sum_{i=1}^{L} N_{i-1} \cdot N_i} \ . \tag{2.19}$$

It is worth noting that for an MLP using structured predefined sparsity, only the weights corresponding to connected edges are stored in memory and used in the computation. Specifically, the parameters are updated on the basis of the gradient of a loss function with respect to the parameters. Let us denote the parameters of the network as θ and the loss function as $\mathscr{L}(\theta)$. The loss function gradient for the parameters is denoted as $\nabla_\theta \mathscr{L}$. Then, the parameters are updated using a gradient descent step:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_\theta \mathscr{L} \,, \tag{2.20}$$

where η is the learning rate, controlling the step size in the direction opposite to the gradient.

2.5.2 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Models details. We use three types of MLPs in our experiments: shallow, deep, and sparse. We can characterize them as follows:

- **Shallow:** This is the simplest type. It contains only one hidden layer besides the input and output layers and represents the base case. The hidden layer neurons receive information from the input layer and process it before transmitting it to the final output layer;
- **Deep:** Each neuron within a hidden layer is fully connected to all neurons in the subsequent layer;
- **Sparse:** The predefined sparsity pattern is applied. This sparsity pattern essentially removes certain connections between neurons in adjacent layers. Specifically, the sparse MLP aims to balance the simplicity of the shallow MLP and the learning capacity of the deep MLP.

Each network configuration is defined using two lists. The first list identifies the number of neurons in the hidden layers:

$$H = [H_0, \ldots, H_n] \,, \tag{2.21}$$

where n is the number of hidden layers, and intuitively, H_0 and H_n represent the number of neurons in the input and output layers, respectively. The second list identifies the out-degree of each neuron for each layer:

$$G = [G_0, \ldots, G_{L-1}]\,. \tag{2.22}$$

Specifically, G_0 is the out-degree of each input layer node, and the output layer G_L is not reported because it is zero.

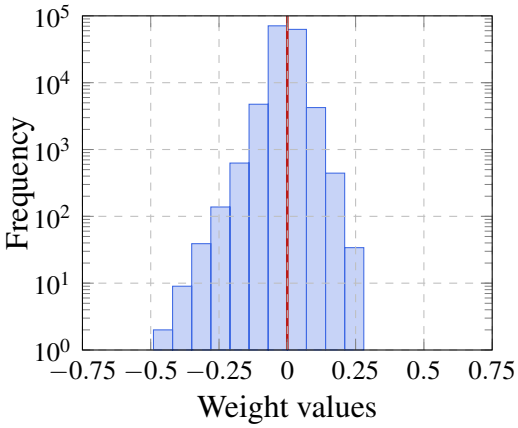
Then, the shallow, deep, and sparse MLP has been split. Our research focuses primarily on the use of existing sparsity patterns in split computing applications for these networks. Inspired by the approach presented in [38], we opt to split the network at the midpoint, ensuring uniformity in our approach. Although more recent studies, such as [40, 2], have proposed advanced techniques to select optimal split points, exploring these methods will be a part of future investigations.

Finally, the inserted EEs are made up of linear layers followed by ReLU activation functions, where the dimension of the last layer matches the desired output size of DNN.

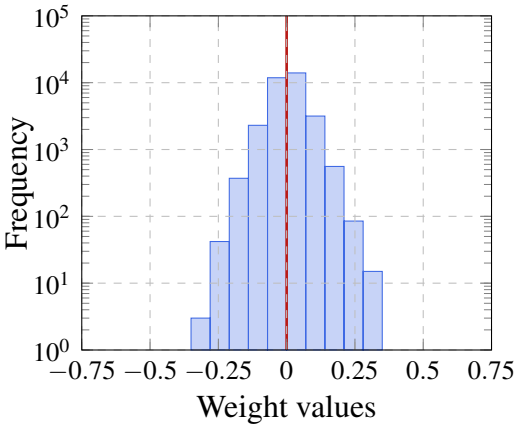
Even with the rise of specialized architectures such as ConvNets, MLPs remains a valuable tool for researchers. Their fundamental structure allows for in-depth exploration of core deep learning concepts without the added complexity of specialized architectures. This focus on MLPs aligns with the experimental nature of this research, in which we aim to isolate the effects of the proposed technique and gain a deeper understanding of its fundamental workings. Furthermore, recent advances in research in MLP have demonstrated their continued effectiveness in various real-world applications, highlighting their ongoing relevance in the field [16, 71].

Datasets descriptions. In this research, we focus on the image classification task. We used the MNIST dataset [72], a well-known collection of handwritten digits. This contains 60,000 training and 10,000 testing images for the multi-class image classification task. The images were centered on a 28×28 image by computing the center of mass of the pixels and translating the image to position this point at the center of the 28×28 field.

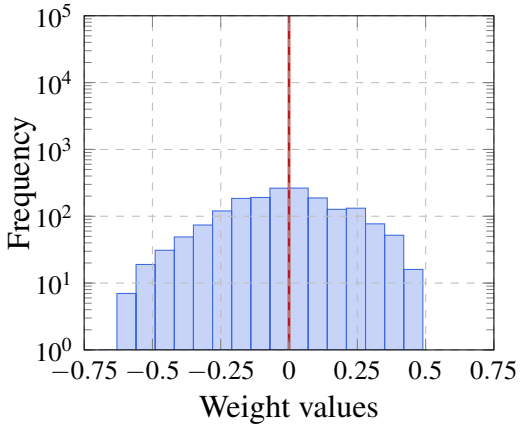
MNIST has to be considered as a placeholder for bigger datasets (*e.g.*, ImageNet [73]); nevertheless, the focus here is to show the potentialities of the prede-



(a) Weights distribution in layer 1 [800, 180].



(b) Weights distribution in layer 2 [180, 180].



(c) Weights distribution in layer 3 [800, 10].

Figure 2.16 Histograms of weights in each junction resulting from training a deep MLP on the MNIST dataset. The network configuration $H = [H_0, \ldots, H_n]$ used is [800, 180, 180, 10].

fined sparsity applied in SC and EE and not beat the state-of-the-art in a specific computer vision challenge.

Although this article focuses on a computer vision task, the concepts explored here remain valid for broader applications. For example, they can be used effectively for other tasks, such as time-series forecasting.

Why Predefined Sparsity in Split Computing and Early Exit?

The motivation behind the predefined sparsity can be exemplified by examining the weight histograms of a trained MLP shown in Figure 2.16. Specifically, the three

histograms show the weight distributions for each layer in a 3-layer MLP model trained on the MNIST dataset with hidden layers having the following number of neurons $H = [800, 180, 180, 10]$.

As we can see from Figures 2.16a and 2.16b, the first layers of the network have a significant concentration of weights around zero, suggesting that these weights might not be crucial for the network's performance. However, Figure 2.16c highlights how the weights in the last layer assume a wider spectrum of values.

This finding suggests that the benefits of predefined sparsity can be especially pronounced in the earlier layers of the network. As a result, in resource-constrained environments, such as those encountered in SC and EE applications, predefined sparsity offers significant advantages because, by reducing the number of connections, we can decrease the size and complexity of the network portion deployed on the edge device. This results in lower memory requirements and faster processing times during training and inference.

Furthermore, the storage footprint is directly proportional to the number of edges. Operating at a sparsity level of, for example, 50% results in a two-fold reduction in complexity. This results in significant efficiency gains, making it possible to deploy more complex models on devices with limited resources. These findings led us to want to study them in the context of SC and EE.

Sparsity-SC&EE Quantitative Results

Figures 2.17 and 2.18 show the accuracy plots against the number of parameters for the three neural network configurations. These two plots reveal the advantage of sparse split models: they present remarkable stability in accuracy even with significant reductions in trainable parameters. This characteristic results in two key advantages, particularly desired and pursued in resource-constrained settings such as SC and EE.

Unlike traditional deep and shallow DNN, where accuracy improvements often depend on a significant increase in the number of parameters, sparse split models achieve optimal inference performance without requiring massive parameter expansions. As a result, they are much more efficient when using the limited processing power and storage space available on edge devices, and so very suitable for distributed deep leasing scenarios through SC and EE.

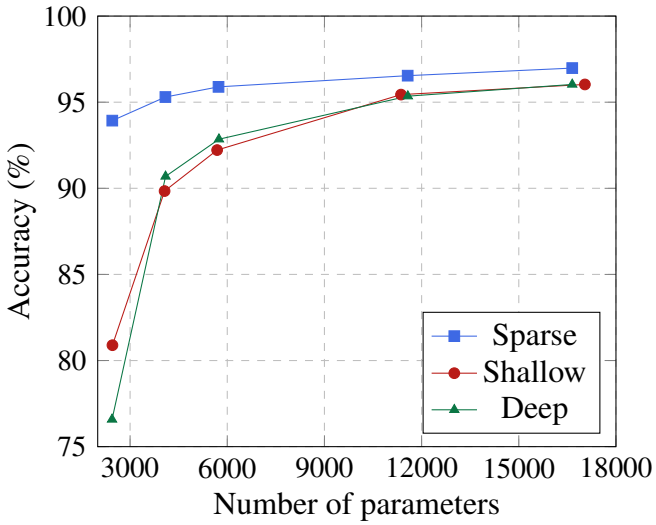


Figure 2.17 Accuracy tests by the number of parameters of deep, shallow, and sparse head MLPs.

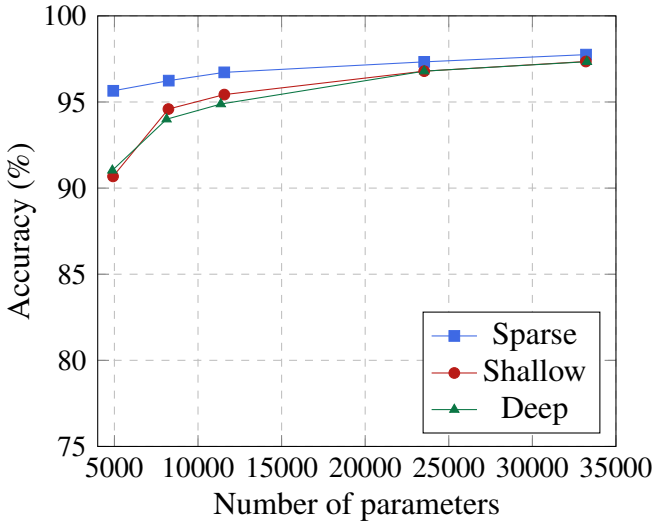


Figure 2.18 Accuracy tests by number of parameters of deep, shallow, and sparse tail MLPs.

Furthermore, the predefined sparsity of these networks leads to decreased memory usage, both during training and inference. With regard to training, this is an advantage because researchers with limited resources can also engage in cutting-edge research in deep learning without the need for expensive high-end computational resources. Instead, reduced complexity results in faster processing times, enabling real-time operation on edge devices with limited processing power.

Discussion

Early Exiting and reduced communication. In SC and EE applications, where data security and bandwidth limitations are paramount, sparse split models represent an ideal solution.

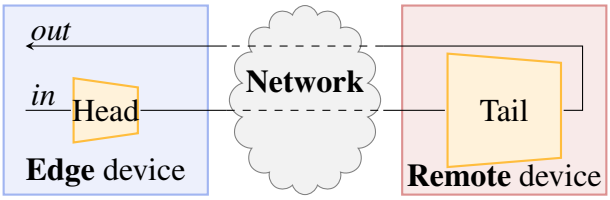
The early exit strategy applied to the head provides an acceptable level of accuracy with a smaller subset of parameters, as highlighted by the accuracy achieved shown in Figure 2.17. This allows the network to terminate computations early, significantly reducing the amount of data transmitted between the edge device and the server. This not only minimizes the risk of data breaches, but also conserves precious network bandwidth, enabling efficient communication even in low-connectivity environments.

Balancing training time and efficiency. While the benefits of sparse split models are evident, it's important to acknowledge the trade-off with training time. When we introduce predefined sparsity into a MLPs, our aim is to train the model to learn a function using a significantly reduced number of connections compared to a non-sparse network. This approach creates a more lightweight structure for the model, but also leads to a more challenging optimization problem. Consequently, the gradient descent algorithm may require more iterations to converge due to the increased complexity introduced by the sparsity constraints. In particular, the sparse split models in our experiments exhibited a $2\times$ slowdown in training time compared to their dense counterparts.

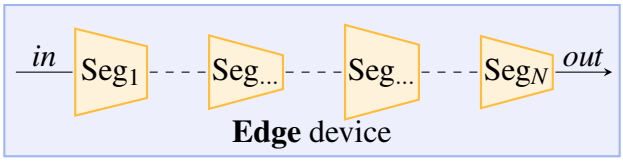
However, this drawback needs to be considered in the context of the specific application and its resource constraints. In many cases, the significant gains in memory usage, communication efficiency, and inference speed during deployment far outweigh the potential increase in training time.

2.6 LO-SC

Our goal with this proposal is to show how to optimally split the neural network based on the computation capabilities of the edge device while minimizing a desired metric. Figure 2.19 summarizes the differences between the SC architectures in the literature and our **LO-SC**. As shown in Figure 2.19a, the typical SC scenario



(a) Existing SC architecture, where the DNN is divided between a head (deployed on an edge device) and a tail (on a remote server), based on the edge device capabilities, network reliability, and delay.



(b) Proposed LoC, where the DNN is divided into multiple segments, all deployed only on the edge device.

Figure 2.19 Traditional SC approach versus the proposed LoC architecture.

focuses on splitting a network into two, between an edge device and a remote server; the overall accuracy of the model can be lower, and the latencies are typically unpredictable. Although Figure 2.19b depicts the **LO-SC** architecture proposed in this thesis. The power of this formalization is the ability to define a set of segments with heterogeneous computation resources that will be executed on the same edge device, *e.g.*, different memory sizes, or maximum execution times. This approach ensures that we do not compromise the model performance while attaining a predictable latency.

The splitting is similar to the well-known *knapsack problem*, where we want to pack a set of items of given sizes into containers with a specified capacity. In our case, the items are the *layers* of the neural network, the size is the *memory* each layer occupies, and the capacity of the containers is the *available memory* of our edge device. We refer to containers because if the neural network cannot be fully executed due to constrained resources, we must split it into segments. We can execute each segment one after the other sequentially to obtain the desired result from DNN. For the remainder of the section, we call *segment* a group of sequential layers executed together.

Section Organization

In the following, we first define the formulation of the **LO-SC** optimization problem in detail (Section 2.6.1). Next, we analyze the computational complexity (Section 2.6.2). Finally, we present the experimental results in Section 2.6.3.

2.6.1 Optimization Problem Formulation

There are several metrics that we could be interested in minimizing. However, the objective we discuss in this chapter is to minimize the output size of the last layer executed in each segment. This objective has a positive impact, regardless of the scenario. On the one hand, one could decide to maintain the output of the last layer of each segment in memory. However, occupying too much memory on a device with limited memory is not advisable. On the other hand, one could decide to store the output of each segment somewhere else (from GPU to RAM or from RAM to disk). However, moving large amounts of data directly affects the overall execution time of the DNN. However, the proposed formalization allows for the exploration of other metrics in the splitting problem.

Setting and notation. Given L layers in the DNN, we know the following information for each layer $l \in [0 \dots L]$:

- $m(l)$: The memory occupied by the layer l.
- $t(l)$: The worst-case execution time of the layer l.
- $o(l)$: The output memory size of the layer l.

Given a total of S segments, we know the following information for each segment $s \in [0 \dots S]$:

- $M(s)$: The available memory of the segment s.
- $T(s)$: The available execution time of the segment s.

Decision variables. Let us start by defining the decision variables of our problem. The first variable we define is x, which keeps track of where the layers are placed as follows:

$$x[l,s] = \begin{cases} 1, & \text{if layer } l \text{ is placed in segment } s \\ 0, & \text{otherwise} \end{cases} \qquad \forall l \in [0 \dots L], \forall s \in [0 \dots S] \,. \tag{2.23}$$

Once the solver has found a solution, reading the assignment it gave to these variables gives you the complete placement of layers in each segment. The second variable we introduce is u, which keeps track of whether a segment is in use as follows:

$$u[s] = \begin{cases} 1, & \text{if segment } s \text{ contains at least one layer} \\ 0, & \text{otherwise} \end{cases} \qquad \forall s \in [0 \dots S] \,. \tag{2.24}$$

Then, we define i, which contains the index of the last layer placed inside a segment as follows:

$$0 \leq i[s] \leq L, \qquad \forall s \in [0 \dots S] \,. \tag{2.25}$$

Finally, y keeps track of the output size of the last layer placed inside a segment as follows:

$$0 \leq y[s], \qquad \forall s \in [0 \dots S] \,. \tag{2.26}$$

Problem constraints. Following is the set of constraints that will ensure that the solutions abide by our resource constraints. The first constraint ensures that we place each layer in only one segment:

$$\sum_{s=0}^{S} x[l,s] = 1 \quad \forall l \in [0 \dots L] \,. \tag{2.27}$$

The next constraint ensures that the total memory occupied by the layers placed inside a segment is lower than its capacity:

$$\sum_{l=0}^{L} x[l,s] \cdot m(l) \leq M(s), \quad \forall s \in [0 \dots S] \,. \tag{2.28}$$

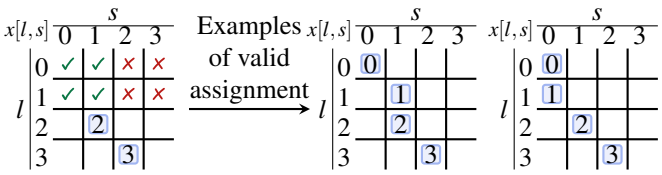


Figure 2.20 Two examples of valid assignments of layers to segments. We can place layers 0 and 1 in the same segment of layer 2 or the previous ones but not in the following ones.

A similar constraint ensures that the total execution time for all the layers placed inside a segment is lower than its total available time:

$$\sum_{l=0}^{L} x[l,s] \cdot t(l) \leq T(s), \quad \forall s \in [0 \dots S] \,. \tag{2.29}$$

Although the focus of this chapter centers on memory-constraint edge devices, placing a limit on the total execution time for each segment serves its purpose. This constraint ensures that the scheduler, responsible for executing layers based on the outcome of the MILP, maintains a consistent overall runtime. Producing consistent execution time and delay due to segment swapping improves the practical utility of this approach for real-time applications.

With the following constraint, u becomes $true$ only if there are layers placed inside the given segment:

$$u[s] = \max_{l \in [0 \dots L]} \quad x[l,s], \quad \forall s \in [0 \dots S] \,. \tag{2.30}$$

Since x is a Boolean, the maximum of a set of x results in a Boolean that reflects the presence of even just one layer in a segment. The next constraint ensures that if we do not use a segment, we are not going to use the subsequent ones as well:

$$\neg u[s] \implies \neg u[s+1], \quad \forall s \in [0 \dots S-1] \,. \tag{2.31}$$

This ensures that the segments are used incrementally, from the first to the last, without empty ones in the middle. The next constraint ensures that we execute the

layers in an ordered fashion:

$$\begin{aligned} x[l_1, s_1] &\implies \neg x[l_2, s_2], \\ \forall s_1 \in [0 \dots S], \forall s_2 &\in [s_1 + 1 \dots S], \\ \forall l_1 \in [0 \dots L], \forall l_2 &\in [0 \dots l_1] \, . \end{aligned} \tag{2.32}$$

To better understand this constraint, let us take the example with four layers and four segments as shown in Figure 2.20. Layers 2 and 3 are placed in segments 1 and 2, respectively. As such, layers 0 and 1 can only be placed in the same segment where layer 2 is or in those before it. Figure 2.20 also shows two examples of valid assignments of layers to segments on the right-hand side.

The following constraints are specific to keeping track of the last layer executed inside each segment. We start by storing the index of the last layer placed inside a segment as follows:

$$i[s] = \max_{l \in [0 \dots L]} \quad x[l, s] \cdot l, \quad \forall s \in [0 \dots S] \, . \tag{2.33}$$

Now, let us again take the example of Figure 2.20, where the values inside the blue boxes result from multiplying the decision variable $x[l, s]$ by the index l. The variable $i[s]$ is the maximum value inside the column s. Finally, we use the previously computed index to get the actual output size of that layer, if and only if the execution segment is actually in use, as follows:

$$y[s] = o(i[s]) \cdot u[s], \quad \forall s \in [0 \dots S] \, . \tag{2.34}$$

Optimization function. We aim to minimize the sum of the output size of the last layers placed inside each segment. The function we use is the following:

$$\text{minimize} \quad \sum_{s \in [0 \dots S]} y[s] \, . \tag{2.35}$$

This minimization objective is crucial for ensuring efficient computational resource utilization or reducing potential overhead due to transferring the output at the end of each segment.

The optimization output is an assignment to the decision variable $x[l, s]$, where the truth values identify when the layers should be executed, akin to a schedule. The

schedule is then read by a scheduler that takes care of *i*) loading the group of layers specified by a segment, *ii*) running the inference, *iii*) unloading the batch of layers, and *iv*) passing the partial output to the next batch of layers.

2.6.2 Computational Complexity

To get a rough idea of the size of instances that this approach can address, we should consider the asymptotic growth of the variables allocated for the MILP formulation. Since we do not know the number of segments we require apriori, we consider creating a number of segments S that can accommodate double the amount of memory required by the layers.

The worst-case upper bound on the size of the variable $x[l,s]$ is $O(L \cdot S)$, while that of the variables $u[s]$, $i[s]$, and $y[s]$, is $O(S)$.

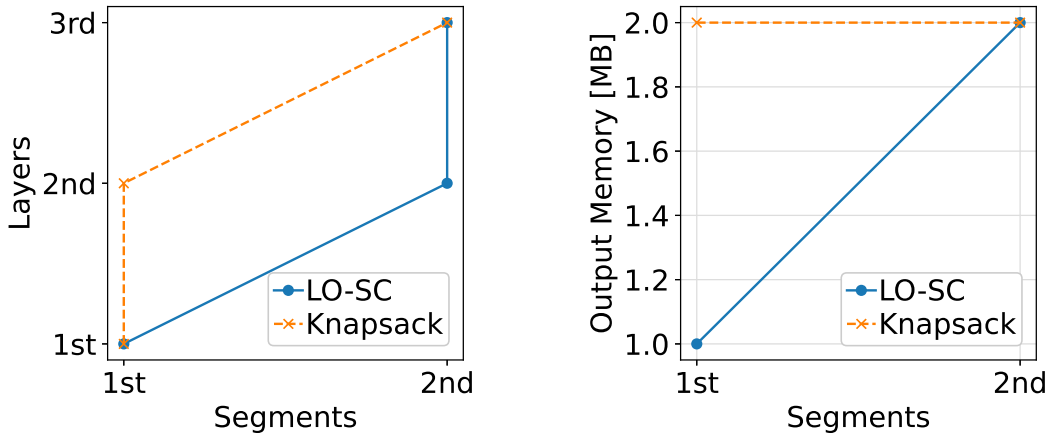
The worst-case upper bound of the constraints can be computed similarly. For Equations (2.27), (2.28) and (2.29) it is $O(S \cdot L)$, while for Equations (2.30), (2.31), (2.33) and (2.34) it is $O(S)$. Finally, for Equation (2.32), we can consider a worst-case upper bound of $O(S^2 \cdot L^2)$. Taking into account all these upper bounds, the setup phase takes approximately $O(S^2 \cdot L^2)$ in the worst case.

We specify that this evaluation does not consider the time that the solver takes to find an optimal solution to the MILP formulation, which is hardware and software dependent (even version dependent). We believe that these upper bounds could provide a reasonably accurate forecast regarding the overall performance of our code when contemplating the deployment of our model implementation as is, with different scenarios.

2.6.3 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Models details. In our experiments, we use two well-known DNNs, *i.e.*, VGG16 [59], MobileNetV1 [74]. We chose the VGG16 architecture because it is a well-established and widely used architecture in many image processing tasks. MobileNets, in con-



(a) Allocation of layers in segments.

(b) Output size for each segment.

Figure 2.21 Comparing our implementation of the multi-constrained ordered knapsack for **LO-SC** and knapsack.

trast, represent cutting-edge DNNs for embedded vision applications. In addition to being very popular architectures, they are also interesting for **LO-SC** because they exhibit a large number of parameters and good depth in terms of layers.

Implementation details. All the code regarding the DNN is implemented in PyTorch Lightning, and the pre-trained network used corresponds to the implementations contained in PyTorch [66]. The code for solving the MILP problem is implemented starting from the CP-SAT constraint programming solver of the OR-Tools open-source software suite [75].

Used edge devices. We run our experiments on an NVIDIA Jetson Nano with 4 GB of memory, except for the VGG16 with inputs of 1920px and 2048px, which we run on an NVIDIA Jetson Xavier NX with 8 GB of memory. For brevity, when we write 1280px (or other resolution), we intend an image of 1280 pixels in width and 1280 pixels in height.

Demonstrative Example

Figure 2.21 compares the solutions found using the proposed **LO-SC** approach and by implementing a knapsack that aims to minimize the number of occupied segments.

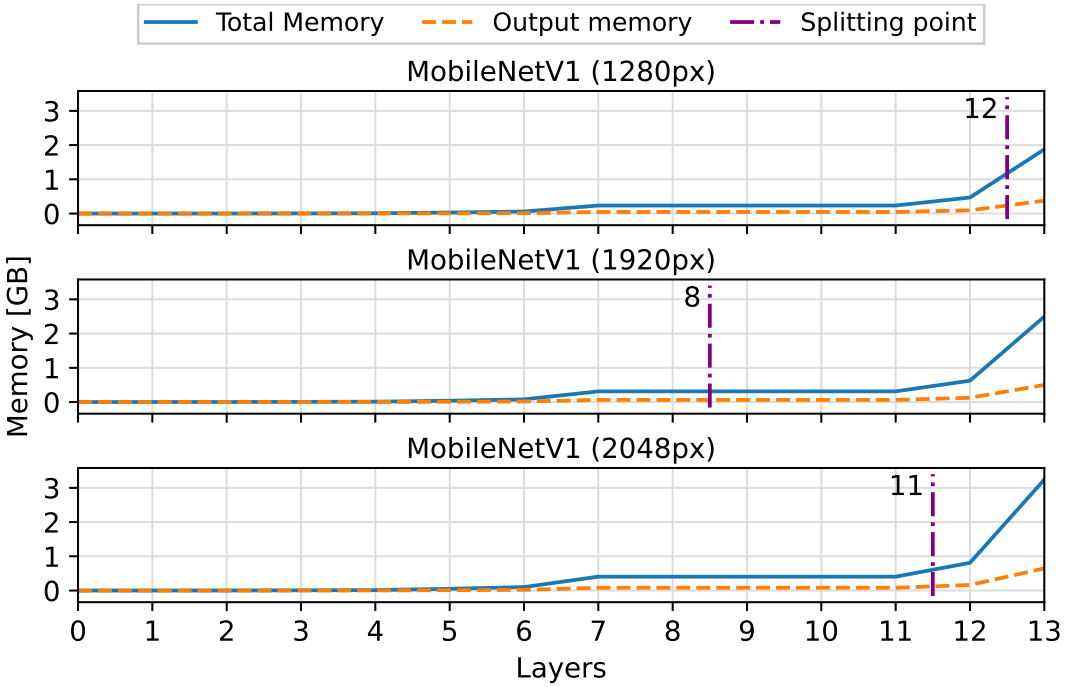


Figure 2.22 Splitting points determined by **LO-SC** when applied to the MobileNetV1 model. The task is to process images of various resolutions, *i.e.*, 1280px, 1920px, and 2048px, on an NVIDIA Jetson Nano.

Figure 2.21a shows the allocation between layers (y-axis) and segments (x-axis). The figure shows that **LO-SC** performs a split after the first layer, while the vanilla knapsack after the second layer. As a result, the output size of the last layer placed inside these segments changes as shown in Figure 2.21b. The segmentation proposed by **LO-SC** has an output size at the end of the first segment of 1 MB, while with a knapsack, it is 2 MB. Inevitably, the output size of the last segment is 2 MB in both cases.

LO-SC Quantitative Results

In these experiments, we focus on the classification task using images of different resolutions, *i.e.*, 1280px, 1920px, and 2048px, depicting objects passing on a conveyor belt within a real Industry 4.0 scenario: the ICE Laboratory at the University of Verona. In particular, Figure 2.22, and Figure 2.23 show the results of the splitting points determined by **LO-SC** when applied to MobileNetV1 and VGG16, respectively.

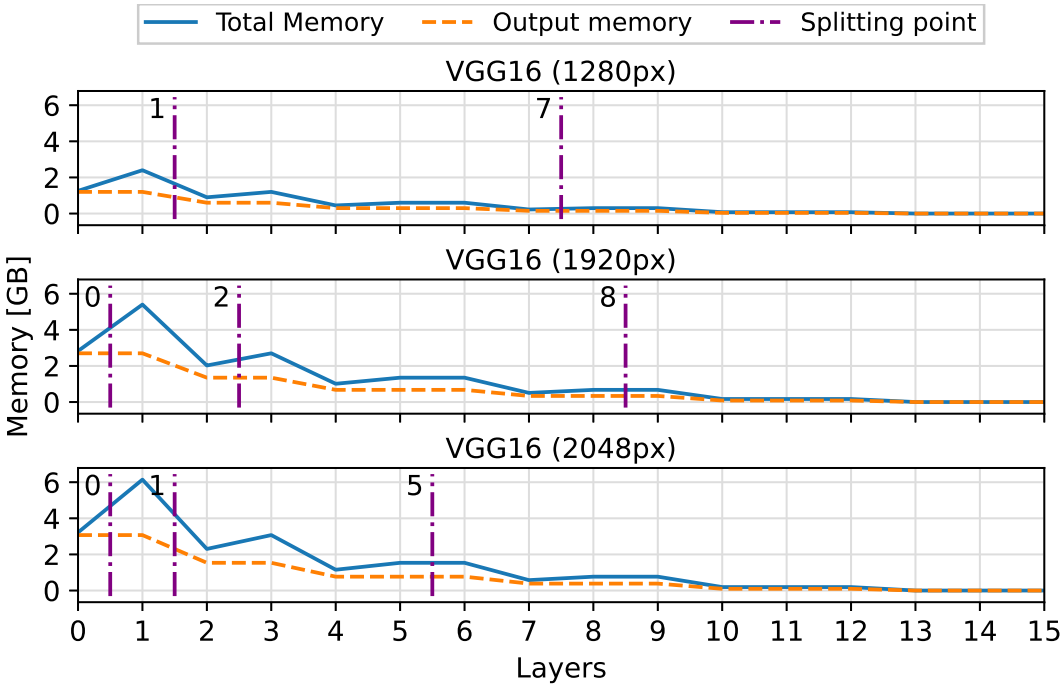


Figure 2.23 Splitting points determined by **LO-SC** when applied to the VGG16 model. The task is to process images with a resolution of 1280px on an NVIDIA Jetson Nano and with 1920px and 2048px on an NVIDIA Jetson Xavier NX.

Despite being a lightweight architecture, MobileNetV1 still exceeds the memory constraints of 4 GB of the Jetson Nano for all inputs. Therefore, **LO-SC** divides it into two different segments, each of which respects the memory restriction of the board. These results demonstrate the ability of **LO-SC** to enable execution of a DNN (in which the memory demands exceed the available resources of the device) on an edge device without relying on optimization techniques or the cloud.

On the other hand, VGG16 should be split into three segments in the case of 1280px and four in the case of 1920px and 2048px. In particular, experiments with resolutions of 1920px and 2048px must be conducted on the Jetson Xavier NX, as some segments, such as the first, reach a total memory of 5.52 GB for 1920px and 6.28 GB for 2048px, both exceeding the 4 GB of the Jetson Nano. This reveals another outcome of **LO-SC**: it delineates the specific characteristics of your DNN and aids in selecting the most appropriate hardware device.

It is worth noting how the design output of **LO-SC** varies not only based on the processed DNN, but also on the input being handled. This demonstrates the dynamic

Table 2.5 Accuracy, precision, recall, and F1-score comparing the VGG16: vanilla, on which quantization and pruning were applied, and on which **LO-SC** was applied.

VGG16	**Accuracy** ↑	**Precision** ↑	**Recall** ↑	**F1** ↑
Vanilla	85.55 %	0.86	0.86	0.86
Quantization and Pruning	77.23 %	0.79	0.77	0.77
LO-SC (ours)	85.55 %	0.86	0.86	0.86

nature of split strategies when accommodating different input dimensions within the network.

Comparing LO-SC against Local-only Computing

LO-SC can outperforms traditional LoC for the deployment of DNNs on resource-constrained edge devices. We demonstrate this by comparing a fine-tuned VGG16 on CIFAR-10 [61] using quantization and pruning vs. **LO-SC**. The pre-trained network corresponds to the implementations in PyTorch [66], and we train our models for 10 epochs, with a learning rate of 1×10^{-5}, using AdamW [67] as an optimizer, on an NVIDIA RTX 3090. The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. CIFAR-10 has to be considered as a placeholder for larger datasets (*e.g.*, ImageNet [73]); indeed, the focus is to show the comparison between the most used LoC methods and **LO-SC**, not beating the state-of-the-art in image classification. Performance is measured using several metrics: accuracy (correct predictions divided by total predictions), precision (proportion of true positives to predicted positives), recall (identifies true positives even with false positives), and F1-score (balances precision and recall).

In particular, we combine a uniform scalar quantization, where floating-point values are linearly compressed and rounded to low-precision quantized types, and magnitude-based pruning, following [76].

The results in Table 2.5 show that VGG16 can be deployed on Jetson using the **LO-SC** technique without losing accuracy. Since no memory optimizations are applied, the output remains uncompressed, ensuring the highest accuracy. However, applying quantization and pruning significantly reduces performance.

Comparing LO-SC against Remote-only Computing

With **LO-SC**, the latency for the switch between segments is entirely predictable, provided by prior knowledge of the number of segments and the transfer rate of the target hardware.

For example, consider the switch operation between segments 1 and 2 of VGG16 with a 1920px input. The amount of data to be transferred is 5.5 GB. Assuming a gigabit channel, the time required to transfer the data is ≈ 44 s (this is only a conservative estimate since we do not consider potential network congestion and delays). Instead, the switch between segments on the considered architectures takes ≈ 15.73 s.

The ability to provide an inference with a deterministic delay could make **LO-SC** suitable for real-time scenarios.

Comparing LO-SC against Split Computing

This section delves into a comparative analysis between **LO-SC** and SC. Specifically, we compare the splitting points identified by **LO-SC** and those identified by the two state-of-the-art methods of SC, namely CDE [40] and I-SPLIT [2] (see Section 2.1). We specify that [3] adopts the basic methodology of [2] with some engineering-specific modifications. Therefore, the evaluation conducted against [2] inherently includes a comparison with [3] as well. Both articles use a VGG16 for experimental purposes, so our comparison is based on this model. In particular, CDE suggests potential splitting points at layers 5, 9, and 13, whereas I-SPLIT recommends splitting the model at layers 5, 9, 11, 13, and 15.

Several facts do emerge. Firstly, none of the potential configurations suggested by the two methods would be feasible in a **LO-SC** scenario, as the two segments do not meet the constraints of both the Jetson Nano and the Jetson NX. Moreover, unlike **LO-SC**, none of them allows partitioning the DNN into more than two segments.

Secondly, unlike **LO-SC**, the final accuracy after the split operation might not necessarily be the highest. In the case of I-SPLIT, this occurs because the authors insert a bottleneck that employs lossy data compression to reduce the transmission data on the network channel. On the other hand, CDE utilizes advanced learning techniques, such as teacher-student approaches [48], to design an architecture that op-

timizes edge performance after splitting. However, these do not guarantee successful results.

Finally, the overall latency time of the system cannot be reliably estimated due to the potential network congestion and its unpredictable impact. In contrast, in **LO-SC**, the latency can be easily estimated.

2.7 LE-CPSs

This section introduces "Split Computing CPS", a topic that we believe has not been studied before. Although there is a rich literature on the implementation of DNNs in edge devices and more recently on SC, the focus has almost exclusively been on maximizing the inference accuracy of DNN. However, in the following, we will argue that an optimal DNN implementation architecture, when evaluated in isolation, might no longer be optimal when used as a part of a larger system, *e.g.*, or a CPS.

Section Organization

In the following, we first introduce the basics of controller design (Section 2.7.1). Then, we detail the controller design specifically for SC (Section 2.7.2).

2.7.1 Basics of Controller Design

When neural networks are used for sensor data processing before data is fed into a controller in a CPS, overall system performance is affected not only by the latency and accuracy of the neural network but also by the underlying dynamics of the physical system. This section introduces the basics of feedback control systems. One common representation of control systems is the state-space model, where the state of the system is represented by a state vector $x(t) \in \mathbf{R}^p$ and the input to the system by $u(t) \in \mathbf{R}^q$. For simplicity, we discuss Linear Time-Invariant (LTI) control systems here, but the principles discussed in this work apply to any type of control system with learning-enabled components.

The state-space model of a continuous LTI system is:

$$\dot{x}(t) = A_c x(t) + B_c u(t) \ , \tag{2.36}$$

where $A_c \in \mathbf{R}^{p \times p}$, and $B_c \in \mathbf{R}^{p \times q}$ are matrices encoding the system's dynamics. Equation (2.36) shows that the rate of change in the state of the system $\dot{x}(t)$ depends on both the current state $x(t)$ and the input of the control $u(t)$. To enable feedback control, the control input $u(t)$ is computed by a periodic real-time task running on a processor. Computing $u(t)$ requires discretizing the continuous state-space model with a constant sampling period h. Assuming periodic sampling, *i.e.*, $t_{k+1} - t_k = h$, matrices A and B can be derived from Equation (2.36) such that:

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k) \ . \tag{2.37}$$

For simplicity, we denote $x(t_k)$ as $x[k]$ and $u(t_k)$ as $u[k]$ to obtain the discrete state-space model:

$$x[k+1] = Ax[k] + Bu[k] \ . \tag{2.38}$$

In the simplest case, the control input $u[k]$ is computed by:

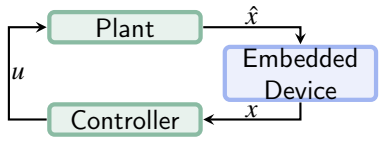
$$u[k] = Kx[k] \ , \tag{2.39}$$

where $K \in \mathbf{R}^{q \times p}$ is the feedback gain. There are many methods to design the feedback gain K with various considerations of stability, energy, and complexity.

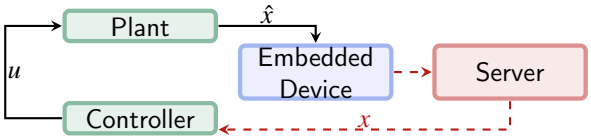
2.7.2 Controller Design for Split Computing

We now discuss multiple SC-augmented feedback controller architectures. These are shown in Figure 2.24 [77]. From their descriptions, it will become clear that these are not exhaustive, and variations of these architectures are possible. As mentioned earlier, our goal in this section is not to conduct an exhaustive study of this topic, but rather to initiate the first discussion.

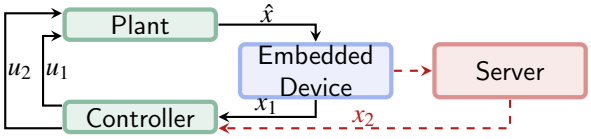
Figure 2.24a shows the most basic scenario, where all DNN is implemented locally in an embedded system. Any embedded platform's relatively low computational bandwidth will restrict the DNN's size, compromising its classification or estimation accuracy.



(a) Basic Edge Computing only scenario.



(b) SC scenario.



(c) SC and EE scenario.

Figure 2.24 SC architectures for a feedback controller.

Figure 2.24b shows a typical SC architecture. Part of the inference is performed on a neural network running locally on an embedded platform, and the subsequent inference is performed on a server in the cloud. Using the cloud and, thereby, supporting a bigger DNN the inference accuracy will be higher, and therefore the state estimation error will be lower than in the case shown in Figure 2.24a. However, a higher delay will be involved in communicating with and from the cloud, resulting in a higher sensor-to-actuator delay. Depending on the communication protocol used, as shown earlier, there might also be data loss, which will also impact the size of the reachable set and, therefore, the system's safety. The research question is to determine an appropriate split point to minimize the size of the system's reachable set and maximize safety. What is important to note here is that due to the delay associated with the sensor actuator at each splitting point, the DNN architecture with SC that maximizes the accuracy of inference will not necessarily be the optimal SC architecture to maximize the safety of the system, motivating a study of SC for CPS.

Finally, Figure 2.24c shows a scenario in which if sufficient inference accuracy is reached using the local DNN, then an early exit may be used, where x_1 is used to compute a control input u_1. This results in a lower sensor-to-actuator delay and avoids any communication with the cloud. But if the inference accuracy using a local-only DNN is not considered sufficient, then the additional computation is carried out in the cloud, and x_2 is instead used by the controller to compute u_2.

However, there could be several additional possibilities here. First, irrespective of the inference accuracy achieved locally using the edge DNN, a "preliminary" state estimate, henceforth referred to as x_1, can be used to calculate an earlier control input u_1. More accurate state estimates x_2 using the cloud DNN may be used later to apply a second control input u_2. This controller addresses the importance of shorter delays and the need for more accurate state estimates. The controller needs to be designed appropriately, since it could be that the control input u_2 is applied during a sampling period later than the one in which u_1 is applied. The entire $\hat{x}$ can be sent to the cloud, or as shown in Figure 2.24c, the data sent to the cloud can be first processed by the edge DNN.

As another possibility, some components of the sensed state $\hat{x}$ can be processed by a local DNN at the edge to compute x_1, and the remaining components of $\hat{x}$ can be computed by the DNN in the cloud to compute x_2. The set of $\hat{x}$ components sent to the edge and the cloud need not be disjoint. How to partition state components between the edge and the cloud and how to design the resulting controller to maximize system safety are again open questions that need to be studied.

Chapter 3

Accurate Anomaly Detection

Anomaly detection is a critical task in intelligent manufacturing, used to identify deviations from expected behavior in production processes, equipment performance, or product quality [8]. Today, this process uses data-driven methods to detect unusual patterns that could signal potential faults, inefficiencies, or even safety hazards. By identifying anomalies early, manufacturers can proactively address problems before they escalate, reducing downtime, and minimizing costly production halts. Moreover, integrating anomaly detection with PdM allows optimized resource use, as maintenance is performed only when necessary. This improves the overall reliability and efficiency of manufacturing systems, ultimately leading to higher product quality and a more resilient production environment.

In Figure 3.1, we present all publications related to accurate anomaly detection. In the following section, we outline the relevant literature and the motivation behind each work. Then, we provide a detailed explanation of each work, encompassing both methodologies and the achieved results.

3.1 Related Works

Computer Vision Domain

SDD is a challenging problem in the detection of visual anomalies in industrial scenarios, defined as the task of individuating samples containing a defect [78],

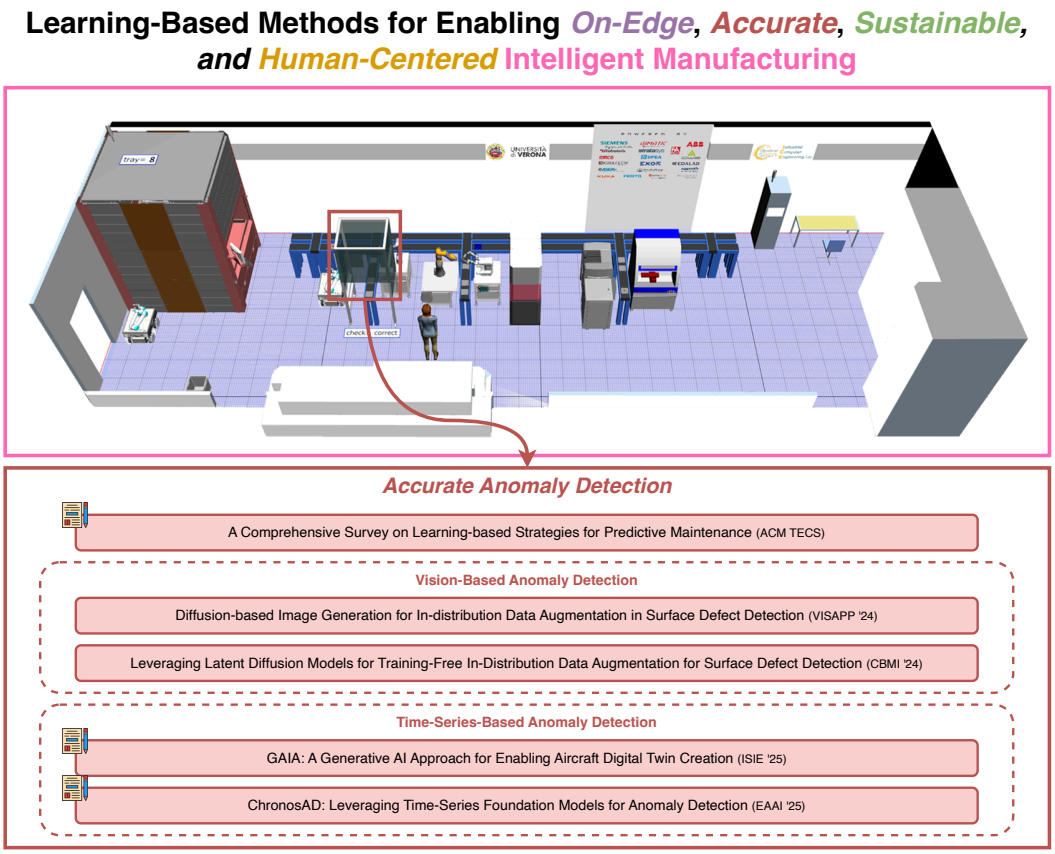


Figure 3.1 Overview of research contributions in accurate anomaly detection. The notebook icon with the pencil indicates that the contribution is currently under submission.

i.e., samples that do not conform to a prototypical texture. In many real-world applications, a human expert inspects every product and removes defective parts. Unfortunately, training human experts can be expensive. In addition, humans are relatively slow to perform this task, and their performances are subject to stress and fatigue.

Automated defect detection systems [79, 80] can easily overcome most of these issues by learning classifiers on defective and nominal training products. The main drawback is the data collection process required to train a model effectively. In fact, defective items (*i.e.*, positive samples) are relatively rare compared to nominal items (*i.e.*, negative samples). Thus, the user may need to collect massive amounts of data to have enough positive samples. Furthermore, with the rise of the Industry 4.0 paradigm and the transition to flexible manufacturing processes, there is an increasing demand for systems that can quickly adapt to new production setups [1],

i.e., customized products manufactured in small batches. Traditional automated systems cannot meet these demands, since data collection could easily involve the whole batch size.

Recent studies on SDD focused on limiting the impact of the labeling process by formulating the problem under the unsupervised learning paradigm [81, 82] or training exclusively on nominal samples [83], possibly using few-shot learning strategies [84]. In both cases, the goal is to generate an accurate model of the nominal sample distribution and classify everything with a low probability score as anomalies. However, due to the limited restoration capacity of these models, these approaches tend to generate many false positives, especially in datasets with complex structures or textures [85].

It is worth noting that, in industrial settings, anomalies are not generated by Gaussian processes but are the result of specific, often predictable, issues during the production process. Consequently, anomalous samples are not randomly distributed outside the nominal distribution; they can be modeled as a mixture of Gaussian distributions in the feature space [86]. Expert operators can easily define the main problems they can expect from the manufacturing process, such as which kinds of defects, in which locations, and how often they expect them to appear. Thus, Generative Artificial Intelligence (GenAI) can represent a powerful tool for SDD, with defect image generation emerging as a promising approach to improve detector performance.

Motivations for In&Out [9] (Section 3.2). As a result, in **In&Out**, we promote using Denoising Diffusion Probabilistic Model (DDPM) to produce fine-grained realistic defects, solving the above issue. Specifically, we can distinguish two different scenarios: *i)* when no defects are available (zero-shot data augmentation); *ii)* when some defects are available, which could be very few (few-shot, or N-shot with N small) or in a large number (full-shot or N-shot with N large).

Motivations for DIAG [10] (Section 3.3). The above work was later extended in another article, in which we propose an interactive learning protocol where a vision language model is used to generate realistic images starting from textual prompts. Specifically, we promote the use of DDPMs to produce fine-grained realistic defect images that can be used as positive samples to train an anomaly detection model. We

name our approach **DIAG**, a training-free Diffusion-based In-distribution Anomaly Generation pipeline for data augmentation in the SDD task. Using pre-trained DDPMs with multimodal conditioning, we can exploit the knowledge of domain experts to generate plausible anomalies without needing real positive data. When using these augmented images to train an anomaly detection model, we show a notable increase in detection performance compared to previous state-of-the-art augmentation pipelines.

Time-Series Domain

Due to their simplicity, MLPs were the initial models extensively investigated in the field of anomaly detection. For example, [87–90] utilizes MLPs to classify anomalous data from normal data. In addition to MLPs, other widely used architectures are RNNs. Since they excel in handling temporal data, they have produced numerous articles for PdM and anomaly detection. [91] utilizes an Long Short-Term Memory (LSTM) to predict the Remaining Useful Life (RUL) of a jet engine. Instead, in [92, 93], the authors used LSTM to predict the faults that may occur in the near future. The [94] utilizes a self-attention mechanism within an LSTM encoder and decoder architecture, where the encoder processes input data, extracts relevant information, and the decoder generates predictions or outputs based on that information, making it more effective for accurate prediction. In addition, ConvNets excels in identifying complex patterns and anomalies in time-series data [95, 96]. In fact, in [97–104], the authors use ConvNets as a feature extractor, and other models such as MLPs or LSTMs are used for the predictions. AEs demonstrated their utility in PdM due to their non-supervisory nature. For example, [105–107] utilizes AE for the anomaly detection task.

In this thesis, we explore the application of the learning architectures presented above to the aircraft domain, with a specific focus on analyzing Landing Gear System (LGS). This is because LGS is one of the most critical components of an aircraft, responsible for safe takeoff, landing, and taxiing operations [108]. As a result, the integrity of LGS is paramount because any failure can lead to catastrophic consequences, compromising the safety of the aircraft and the lives of its passengers and crew. The primary challenge in this domain is the limited availability of real-world fault data [109]. This is because collecting such data is not only expensive but also fraught with safety concerns, as deliberately inducing faults in operational

systems is not feasible. Consequently, the aviation industry has increasingly relied on simulation as a crucial way to generate data to support maintenance strategies. Simulations offer a controlled environment where deterministic fault scenarios can be engineered based on known physical principles and system behaviors, ensuring safety while providing valuable data for model development.

Although high-fidelity simulations can generate deterministic data following specific rules and conditions, they may not fully represent the variability and unpredictability of real-world operations [110]. In this context, Generative AI can be crucial in bridging this gap by generating more diverse and realistic data. Introducing realistic variations can narrow the gap between simulated environments and real-world conditions, enhancing the robustness and generalizability of learning models.

Motivations for GAIA [11] (Section 3.4). **GAIA** is the first comprehensive pipeline to allow the creation of DTs to support PdM in the aircraft domain. Specifically, **GAIA** introduces *i)* the first benchmark for LGS and *ii)* an effective method to enhance these data through GenAI, meeting rigorous standards of aerospace grade. In particular, DSLG D/R is a novel and accurate physics-driven dataset designed for LGS multiclass fault classification in collaboration with Leonardo S.p.A., a leading international aerospace company. Our commitment to providing reliable, high-quality datasets is directly aligned with industry demands. Furthermore, to enhance our multi-physics data, we design and train a GenAI model, specifically a Conditional Generative Adversarial Network (C-GAN), ensuring that the generated data accurately capture the variability observed in real-world simulations.

Motivations for ChronosAD [12] (Section 3.5). With **ChronosAD**, we propose a novel deep anomaly detection method for tabular data that leverages recently suggested time-series foundation models to capture both temporal and contextual dependencies. We utilize pre-trained embeddings from the Chronos time-series foundation model to encode temporal patterns, while a custom Temporal Block comprising Bidirectional LSTMs and multi-head attention refines these embeddings to extract contextual relationships. This design enables our method to adapt to diverse tabular datasets, effectively modeling intricate data patterns for robust anomaly detection. To the best of our knowledge, this is the first work that successfully

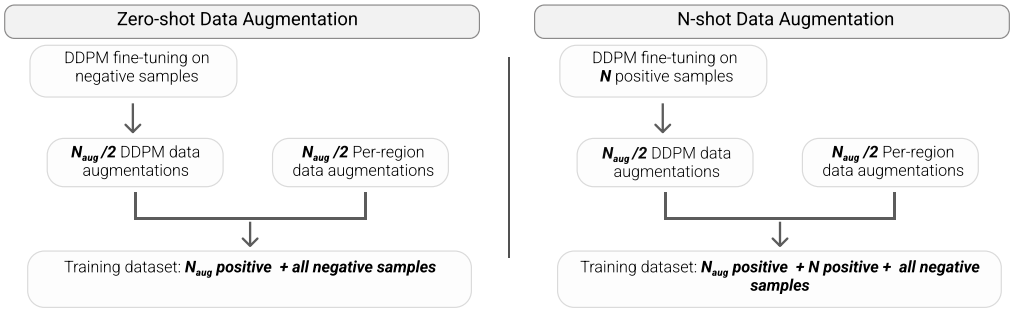


Figure 3.2 General schema of our **In&Out** method.

shows how time-series foundation models can be used for the downstream anomaly detection task on tabular datasets.

3.2 In&Out

The **In&Out** data augmentation proposal aims at producing N_{aug} additional positive images. The approach can be applied, with slightly different pipelines, in two scenarios: *i)* when there are no positive samples available (zero-shot data enhancement) and *ii)* when positive samples are available (N-shot data enhancement, where N can be small or large). In the following, the two pipelines are detailed; a graphical sketch is presented in Figure 3.2.

Section Organization

In the following, we first discuss the **In&Out** zero-shot data augmentation (Section 3.2.1). Next, we present the **In&Out** N-shot data augmentation (Section 3.2.2). Finally, we provide the experimental results in Section 3.2.3.

3.2.1 Zero-Shot Data Augmentation

In this scenario, we simulate that no positive samples are available in the training set. Thus, our aim is a zero-shot data augmentation procedure in which two steps are performed: fine-tuning and data augmentation.

Fine-tuning step. Dreambooth is adopted to perform fine-tuning on a DDPM. To reduce training time and lower computation requirements, we only train low-rank update matrices employing Low-Rank Adaptation (LoRA) [111]. These update matrices are then summed to the original weights, completing the fine-tuning procedure. Specifically, we control the weight of the LoRA update matrices during the merge with a parameter α: a value close to 0 results in no fine-tuning, while a value close to 1 results in the strongest fine-tuning.

In the zero-shot data augmentation, we perform fine-tuning with a portion of randomly chosen negative samples from the training set. The number of samples depends on the complexity of the data we want to manipulate: the larger the intra-class variance, the larger the number of elements to sample. In this preliminary study, we select the number of samples heuristically.

Data augmentation. In this step, we create the N_{aug} augmented images generating $N_{aug}/2$ in-distribution images and $N_{aug}/2$ out-of-distribution images. The $N_{aug}/2$ in-distribution images are obtained by exploiting the fine-tuned DDPM through natural language prompts, describing the desired anomalies. To define the types of defects in natural language and verify how well text expressions are suited to generate a genuine defect for the data at hand, it is reasonable to perform some human-in-the-loop cycles, exploiting the expert's domain knowledge to evaluate the augmentation quality. Specifically, the operator prompts textual expressions and evaluates the generated data (total of $N_{aug}/2$), certifying reasonable defects or revising expressions for improved generations. The $N_{aug}/2$ out-of-distribution images are obtained by increasing the data per region.

This ensures that half of the augmented data will be in-distribution, describing the visual appearance of the defects (the diffusion-based one), while the other half of the data will focus on specifying what is certainly not a perfect sample (the per-patch images). After augmentation, the final training dataset will be formed by positive images augmented with N_{aug} plus all the original negative samples.

3.2.2 N-Shot Data Augmentation

In this scenario, we assume that we have N images from the positive pool of dataset images on which we perform Dreambooth fine-tuning with LoRA. We refer to the

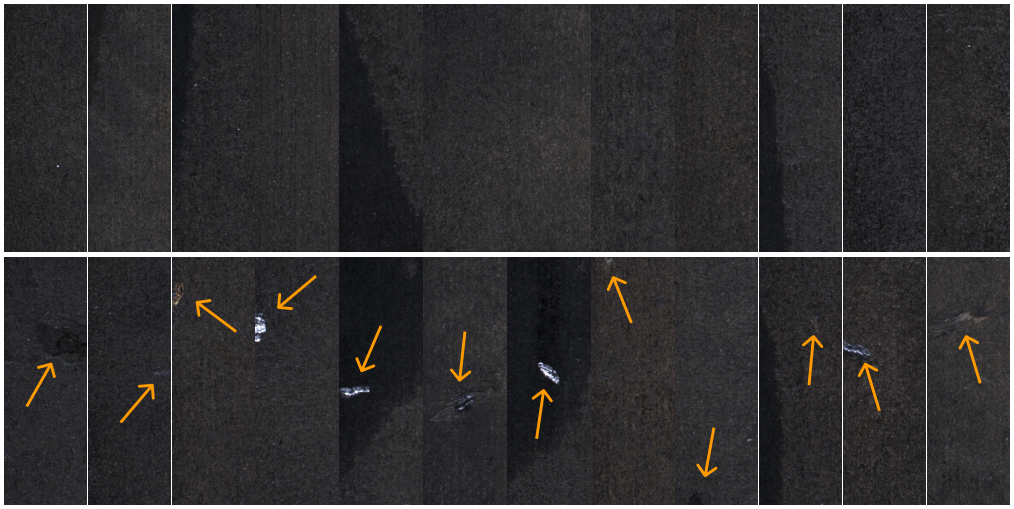


Figure 3.3 Normal (top row) and anomalous (bottom row) samples from the Kolektor Surface-Defect Dataset 2 (KSDD2) dataset. Note that some defects are very difficult to find.

cases where $N \sim 5$ is a few-shot data augmentation. After fine-tuning, positive $N_{aug}/2$ samples in distribution are generated. For the zero-shot data augmentation scenario, additional $N_{aug}/2$ out-of-distribution images are obtained by the per-region data augmentation.

After augmentation, the final training dataset will be formed by N_{aug} augmented positive images + N original positive images plus the negative samples.

3.2.3 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Dataset descriptions. The KSDD2 [23] contains RGB images of defective production items, provided and annotated by Kolektor Group d.o.o. The defects vary in shape, size, and color, from small scratches and minor spots to large surface imperfections.

Since the images are of different sizes, we standardize the resolution of the dataset by centering the image and resizing all the images to 200×600 pixels. The dataset is divided into training and testing subsets, with 2,085 negative and 246

Figure 3.4 Anomalous samples generated by DDPM. It is evident how it provides in-distribution positive samples.

positive samples in the training set and 894 negative and 110 positive samples in the test set. We show several normal and anomalous samples in Figure 3.3.

DDPM fine-tuning. In our experiments, we use Stable Diffusion [112] as DDPM. The fine-tuning process follows the Dreambooth procedure. We used the prompt "`skt background`", where "`skt`" is the identification token. The string "`skt`" has no semantic meaning and was selected to define an ID code for a new visual class. On the other hand, "`background`" is the subject class, identified as the most suited to obtain images with a homogeneous background. The regularization images have been generated using the prompt "`background`". The weight of the prior preservation loss is set to 1.0 as in the original article. For faster training time and lower computation requirements, we also employ the LoRA-c3Lier low-rank adaptation, a modified version of LoRA that also applies low-rank approximations to 3×3 convolutional kernels and linear layers.

The code is implemented in PyTorch [66]. We used AdamW8bit [113] as an optimizer, with a learning rate of $1e-5$. We kindly direct the reader's attention to our configuration file for a more comprehensive exploration of the various hyperparameters involved.

DDPM data augmentation. After training Stable Diffusion, we use it to generate $N_{aug}/2$ augmented images. In the zero-shot scenario, we use the prompts "`skt background cracked`" and "`skt background scratched`" to induce the generation of anomalous samples. These prompts have been chosen after a series of tests and result in images containing plausible anomalies like the ones shown in Figure 3.4. These generated images are then added to the training set, which will be used to train the anomaly detection model. We train and evaluate this model with four different seeds for each of our experiments, generating $N_{aug}/2$ new images each time to provide the most statistically relevant results.

ResNet-50 training and testing. We use the PyTorch implementation of ResNet-50 [56] as our anomaly detection model, in which we substitute the fully connected layers after the backbone to make it a binary classifier. The network is trained for 50 epochs with an SGD optimizer, a learning rate of 0.01, and a batch size of 5.

To maintain consistency with the training and evaluation procedures of KSDD2, we modify their official implementation to accommodate our ResNet-50 model. In particular, our setup is similar to the weakly supervised one presented in [23], where only images and ground-truth labels are used to train the model. For each scenario, *i.e.*, zero-shot data augmentation and N-shot data augmentation, we will train three versions of our ResNet-50 model: *i)* using only MemSeg [114] to generate N_{aug} images; *ii)* using only our DDPM to generate N_{aug} images; and *iii)* using **In&Out** as data augmentation, resulting in $N_{aug}/2$ images generated by MemSeg and $N_{aug}/2$ generated by our DDPM.

Zero-Shot Data Augmentation

In these experiments, we emulate a situation where ***no*** positive samples are available in the training set. With this premise, we train our diffusion model with only 50 randomly chosen negative samples from the training set. We chose this number empirically and deemed it sufficient to represent the intra-class variance of the negative samples. We train the DDPM for 5 epochs, using as a guiding prompt "`skt background`" and $\alpha = 0.60$.

Once the diffusion model is trained, we generate $N_{aug}/2$ augmented positive samples using specific requests from the data set. In our case, we used prompts such

Table 3.1 Results between MemSeg and DDPM when ***no*** anomalous samples are available.

$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
MemSeg 80	.514 (.026)	**.733** (.113)	.436 (.033)
MemSeg 100	.388 (.066)	.633 (.129)	.432 (.054)
MemSeg 120	.511 (.050)	.683 (.054)	.470 (.091)
Average	.471 (.047)	**.683** (.099)	.446 (.059)
$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
DDPM 80	**.547** (.086)	.427 (.301)	.695 (.194)
DDPM 100	.532 (.028)	.387 (.277)	**.714** (.286)
DDPM 120	.445 (.186)	.465 (.329)	.591 (.274)
Average	**.508** (.100)	.426 (.302)	**.667** (.251)

as "`skt background cracked`" and "`skt background scratched`", resulting in images similar to those shown in Figure 3.4. Therefore, we produce $N_{aug}/2$ images outside the distribution using MemSeg, obtaining N_{aug} of our **In&Out** approach. We also experimented with fully MemSeg and fully DDPM augmentation pipelines for comparison.

We train the ResNet-50 model on different values of N_{aug} and evaluate it on the original test set. For each number of data augmentation, four different seeds have been used to report the most statistically relevant results. We report the comparison between MemSeg and DDPM in Table 3.1, where the numbers outside the parentheses indicate the average results over the four seeds, while the numbers between parentheses indicate the standard deviation. As we can see, DDPM achieves the highest Average Precision (AP) (.547), recorded at 80 augmented images, while also resulting in an overall higher mean AP compared to the MemSeg pipeline (.508 vs. .471).

We want to highlight the difference between the precision and recall scores of MemSeg and DDPM. Although DDPM achieves a higher recall (.714), the MemSeg pipeline results in a higher precision (.733). This behavior is clearly shown in Figure 3.5 and Figure 3.6, where we plot the precision and recall values of the two methods for different N_{aug}.

When combined in the **In&Out** pipeline, where half of the augmented positive samples are provided by DDPM and the other half is provided by MemSeg, we obtain a huge performance boost in maximum (.626) and average (.573) AP, with balanced

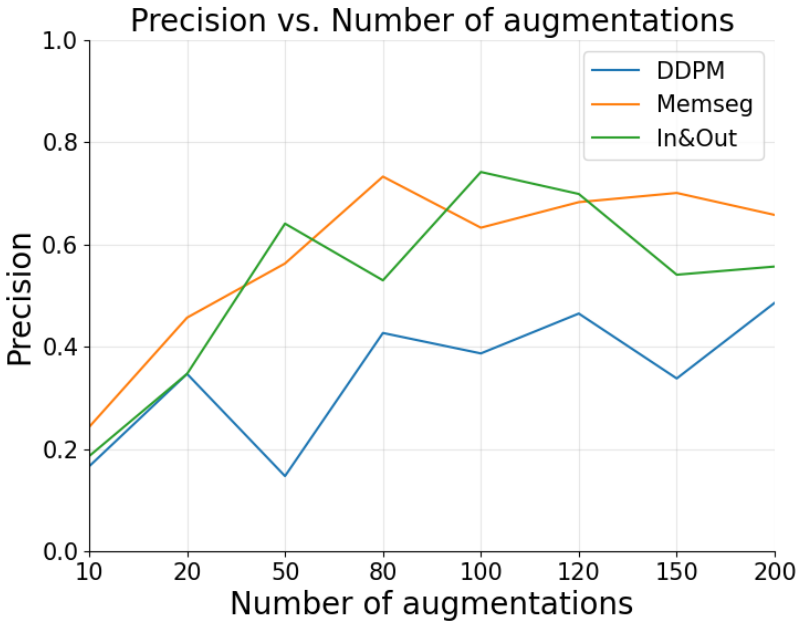


Figure 3.5 Precision of the methods as a function of the number of augmentations. Note that MemSeg has higher overall precision. **In&Out** balances this metric.

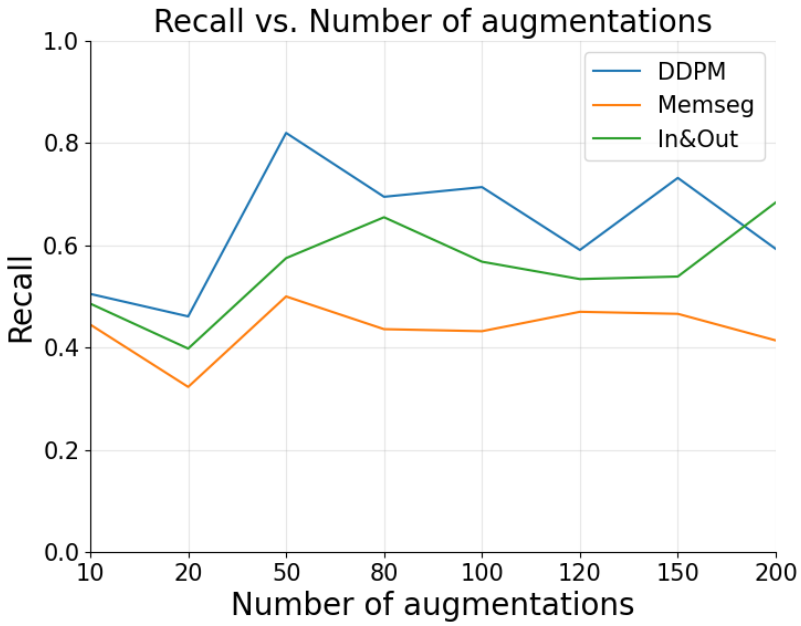


Figure 3.6 Recall of the methods as a function of the number of augmentations. Note that DDPM has a higher overall recall. **In&Out** balances this metric.

precision and recall metrics. These results, reported in Table 3.2, suggest how combining in-distribution (DDPM) and out-of-distribution (MemSeg) data improves

Table 3.2 Results when ***no*** anomalous samples are available using **In&Out**. Thus, $N_{aug}/2$ samples generated with DDPM and $N_{aug}/2$ with MemSeg.

$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
In&Out 80	.556 (.085)	.530 (.219)	**.655** (.065)
In&Out 100	**.626** (.059)	**.742** (.109)	.568 (.029)
In&Out 120	.536 (.023)	.699 (.085)	.534 (.086)
Average	**.573** (.056)	.657 (.138)	.586 (.060)

Table 3.3 Results between MemSeg and DDPM when ***few*** anomalous images are available. Each training set contains $N = 5$ anomalous samples, plus N_{aug} augmented images.

$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
MemSeg 80	.582 (.018)	**.836** (.101)	.466 (.049)
MemSeg 100	.511 (.086)	.686 (.082)	.527 (.069)
MemSeg 120	**.593** (.044)	.801 (.065)	.507 (.053)
Average	**.562** (.049)	**.774** (.083)	.500 (.057)
$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
DDPM 80	.580 (.045)	.542 (.270)	**.634** (.212)
DDPM 100	.526 (.075)	.610 (.063)	.477 (.081)
DDPM 120	.535 (.063)	.659 (.127)	.491 (.046)
Average	.547 (.061)	.604 (.153)	**.534** (.113)

precision and recall scores, helping the model better understand what an anomalous sample is.

N-Shot data augmentation, N small

Within manufacturing environments, organizations strive to minimize the occurrence of defects, resulting in a generally restricted number of anomalous samples. In this subsection, we place ourselves in the situation described above, *i.e.*, only a minimal number of ground-truth positive samples are available in the dataset.

To simulate this challenging setup, we randomly select only $N = 5$ anomalous samples from the KSDD2 training dataset and use them to fine-tune the DDPM for 49 epochs with $\alpha = 0.95$. Following the procedure introduced in Section 3.2.2, we generate several training sets induced by the different N_{aug} of new samples, plus the N images on which we trained the DDPM. For the classifier, we use the same

Table 3.4 Results when ***few*** anomalous images are available using **In&Out**. Each training set contains $N_{pos} = 5$ anomalous samples, plus N_{aug} augmented images, where half samples are generated by DDPM and half by MemSeg.

$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
In&Out 80	.531 (.041)	.507 (.220)	.655 (.126)
In&Out 100	**.578** (.041)	.450 (.343)	**.761** (.245)
In&Out 120	.575 (.025)	**.635** (.316)	.636 (.189)
Average	.561 (.036)	.531 (.293)	**.684** (.187)

ResNet-50 architecture. The findings of this experiment are documented in Table 3.3. As we can see, the MemSeg method slightly outperforms DDPM, resulting in an average AP of .562 and .547, respectively. Moreover, MemSeg produces a maximum AP of .593 at $N_{aug} = 120$, while DDPM records a maximum AP of .580 at $N_{aug} = 80$. Precision and recall behave similar to what was seen in Section 3.2.1, with DDPM having a higher recall (.634 vs. .527) and a lower precision (.659 vs. .836) w.r.t. MemSeg.

Interestingly enough, in Table 3.4, we can see that the **In&Out** pipeline does not seem to increase the performance, achieving an average AP on par with MemSeg (.561) while recording a slightly lower maximum AP (.578 vs. .593). We hypothesize that, in this setup, DDPM overfits the minimal number of anomalous images and cannot generalize the anomalous samples properly. This is a problem if the samples on which we fine-tune the model are a subset of all the anomalies and, thus, are not representative enough of the entire anomalous distribution.

N-Shot data augmentation, N large

Finally, to demonstrate **In&Out** as a general data enhancement technique, we explore the scenario with more positive samples in the training set. To this end, we made all 246 positive samples available to the anomaly detection model during training, in addition to the usual N_{aug} augmented anomalous images. Following the procedure in Section 3.2.2, we use all the $N = 246$ positive samples from the training set to fine-tune our diffusion model for 25 epochs with $\alpha = 0.80$. Finally, we define a baseline by training ResNet-50 with $N_{aug} = 0$ (**In&Out** 0), achieving an average AP of .747. The results are reported in Table 3.5.

Table 3.5 Results when ***all*** the anomalous samples are available using **In&Out**. Each training set contains all the anomalous KSDD2 samples, plus N_{aug} augmented images, where half of the samples are generated by DDPM and half by MemSeg. Additionally, **In&Out** 0 indicates the performance achieved without data augmentation. Note that MixedSegdec [23] indicates the results reported under the weakly supervised setting.

$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
MixSegdec	.733 (-)	- (-)	- (-)
In&Out 0	.747 (.055)	.826 (.081)	.723 (.058)
In&Out 80	.747 (.022)	.764 (.046)	**.734** (.032)
In&Out 100	.775 (.013)	.868 (.050)	.720 (.026)
In&Out 120	**.782** (.030)	**.906** (.064)	.689 (.030)
Average	**.768** (.022)	**.846** (.053)	.714 (.029)

Table 3.6 Results between MemSeg and DDPM when ***all*** the anomalous samples are available.

$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
MemSeg 80	.744 (.007)	.851 (.055)	.691 (.058)
MemSeg 100	**.774** (.016)	.814 (.038)	.752 (.028)
MemSeg 120	.734 (.032)	.772 (.107)	.707 (.031)
Average	.751 (.018)	.812 (.067)	.717 (.039)
$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
DDPM 80	.758 (.007)	.808 (.056)	**.768** (.043)
DDPM 100	.763 (.008)	.829 (.059)	.725 (.034)
DDPM 120	.772 (.034)	**.858** (.084)	.725 (.061)
Average	**.764** (.016)	**.832** (.066)	**.739** (.046)

The results of the two separate data augmentation procedures are reported in Table 3.6. In this scenario, the anomaly detection model trained with DDPM augmented images achieves a maximum AP of .772, outperforming both the baseline (.747) and resulting in a higher average AP than MemSeg (.764 vs. .751). As we can see in Table 3.5, **In&Out** achieves the highest average AP yet (.768) while balancing precision and recall metrics, confirming our intuition. In particular, with 120 augmented images, the maximum AP classification score is .782, beating the previous .733 [23] and setting the new state-of-the-art.

3.3 DIAG

In this section, we provide detailed explanations of **DIAG**, in which we investigate spatial control to allow the synthesis of defect samples that incorporate regional information and exhibit enhanced controllability of image generation through a human-in-the-loop pipeline, effectively using domain expertise to generate more plausible in-distribution anomalies.

Section Organization

In the following, we first explore multimodal diffusion-based image generation (Section 3.3.1). Then, we describe the **DIAG** pipeline in detail (Section 3.3.2) and discuss its application to the anomaly detection task (Section 3.3.3). Finally, we present the experimental results in Section 3.3.4.

3.3.1 Multimodal Diffusion-Based Image Generation

DDPMs [115, 116] are a class of deep latent variable models that work by modeling the joint distribution of the data over a Markovian inference process. This process consists of small perturbations of the data with a variance preserving property [117], such that the limit distribution after the diffusion process is approximately identical to a known prior distribution. Starting with samples from the prior, a reverse diffusion process is learned by gradually denoising the sample to resemble the initial data by the end of the procedure.

Formally, the data distribution $q(x_0)$ is modeled using a latent variable model $p_\theta(x_0)$:

$$p_\theta(x_0) = \int p_\theta(x_{0:T}) dx_{1:T} \,, \tag{3.1}$$

$$p_\theta(x_{0:T}) := p_\theta(x_T) \prod_{t=1}^{T} p_\theta^{(t)}(x_{t-1}|x_t) \,, \tag{3.2}$$

where $x_1, \ldots, x_T$ are latent variables of the same dimensionality as x_0.

The parameters θ are learned by maximizing an ELBO (evidence lower bound) of the log evidence, *i.e.*:

$$\max_{\theta} \mathbb{E}_{q(x_0)}[\log p_\theta(x_0)] \leq$$
$$\max_{\theta} \mathbb{E}_{q(x_0,x_1,\ldots,x_T)} \left[\log p_\theta(x_{0:T}) - \log q(x_{1:T}|x_0)\right] , \tag{3.3}$$

where $q(x_{1:T}|x_0)$ represents a fixed inference process defined as the following as a Markov chain:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) , \tag{3.4}$$

$$q(x_t|x_{t-1}) := \mathcal{N}\left(\sqrt{\frac{\alpha_t}{\alpha_{t-1}}}x_{t-1}, \left(1 - \frac{\alpha_t}{\alpha_{t-1}}\right) I\right) , \tag{3.5}$$

where $\alpha_{1:T} \in (0,1]^T$ is a predefined variance schedule and the covariance matrix is ensured to have positive terms on its diagonal. Specifically, this parametrization has the property:

$$q(x_t|x_0) = \int q(x_{1:t}|x_0) dx_{1:(t-1)} =$$
$$\mathcal{N}(x_t; \sqrt{\alpha_t}x_0, (1-\alpha_t)I) , \tag{3.6}$$

therefore we can write x_t as a linear combination of x_0 and a noise variable ε.

When we set α_T sufficiently close to 0, $q(x_T|x_0)$ converges to a standard Gaussian for all x_0, so it is natural to set $p_\theta(x_T) := \mathcal{N}(0, \mathbf{I})$. Given that all conditionals are modeled as Gaussian with fixed variance, the objective in Section 3.3.1 can be greatly simplified. In particular, [116] shows that the following (further simplified) lower bound provides optimal generative performance:

$$L(\varepsilon_\theta) := \sum_{t=1}^{T} \mathbb{E}_{x_0,\varepsilon_t} \left[\|\varepsilon_\theta^{(t)}(\sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\varepsilon_t) - \varepsilon_t\|_2^2\right] , \tag{3.7}$$

where $x_0 \sim q(x_0), \varepsilon_t \sim \mathcal{N}(0, I)$, $\varepsilon_\theta = \{\varepsilon_\theta^{(t)}\}_{t=1}^T$ is a set of T functions, with each $\varepsilon_\theta^{(t)} : X \to X$ having trainable parameters $\theta^{(t)}$.

In practice, these functions are approximated by a neural network conditioned on diffusion time t. After training the model, we can generate new samples by

first sampling x_T from the known prior $p_\theta(x_T)$, and then iteratively reversing the diffusion process, thereby sampling $\{x_{T-1} \dots x_0\}$.

In addition, we used the natural ability of DDPMs to incorporate multimodal conditioning into the generation process, taking inspiration from [118, 112, 119, 120]. Specifically, we will use prompts, *i.e.*, textual descriptions of the anomaly, and negative prompts, *i.e.*, prompts that guide the image generation "away" from its concepts. This results in high-quality images that comply with the descriptions given [121–123].

In particular, we opt to utilize an inpainting model, as demonstrated in [115, 112]. Given an image with a masked region, inpainting seamlessly fills it with content that harmonizes with the surrounding image. Although typically used to eliminate unwanted artifacts, the inpainting process ensures that the masked area incorporates the provided prompt, effectively merging textual and visual content.

3.3.2 The DIAG Pipeline

To generate an anomalous image i_a, the process starts by sampling a random negative image, a description of the anomaly, and a mask, which forms the triplet (i_n, d_a, m_a). Instead of directly operating on the image pixels using DDPM, we use a Latent Diffusion Model (LDM) to work in a lower-dimensional latent space [112]. Thus, the above information will be fed to a text conditioned LDM to perform inpainting on the image i_n using the mask m_a.

The anomaly description d_a guides the generation, filling the masked region of i_n with an anomaly that complies with the prompt. The domain knowledge of industrial experts is used to generate images reminiscent of real anomalous samples, providing textual descriptions of the type, shape, and spatial information of potential anomalies.

The LDM is then conditioned on this information to inpaint plausible anomalies in defect-free samples. Formally, given pictures of sample samples without defects (negative) I_n, domain experts will provide textual descriptions D_a of what different anomalies may look like. At the same time, regions where these anomalies may appear in the defect-free samples will be designated. We define this set of regions as a set of binary masks M_a of possible anomalies, shapes, and locations. The result of this operation is i_a, an anomalous version of i_n, where an anomaly has been inpainted

in the masked region m_a. Due to the stochastic nature of LDMs, this process can be repeated multiple times to generate an augmented set of anomalous sample images I_a. Finally, the set I_a can be used as a data augmentation for training anomaly detection models, as presented in the following section.

3.3.3 The Anomaly Detection Task

We approach the anomaly detection problem as a binary classification problem, where the objective is to predict whether a sample belongs to one of two classes. Specifically, we used a ResNet-50 [56] backbone trained with a binary cross-entropy loss function denoted as $\mathscr{L}_{\text{BCE}}$. The binary cross-entropy loss measures the dissimilarity between the predicted probability distribution and the actual distribution of the labels. Mathematically, it is defined as:

$$\mathscr{L}_{\text{BCE}}(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] , \tag{3.8}$$

where, y represents the ground truth labels, $\hat{y}$ represents the predicted probabilities, and N is the number of samples. In detail, y_i denotes the true label for sample i, which can be 0 or 1, while $\hat{y}_i$ signifies the predicted probability that sample i belongs to class 1.

3.3.4 Experimental Results

In this section, we show the efficacy of our data augmentation approach for defect detection from a quantitative and qualitative point of view.

Dataset description. We use the KSDD2 [23], one of the most recent, complex, and real-world surface defect detection datasets. This dataset comprises 246 positive and 2,085 negative images in the training set and 110 positive and 894 negative images in the testing set. Positive images are images with visible defects, such as scratches, spots, and surface imperfections.

Since the images have different dimensions, we standardize the resolution of the dataset, resizing all the images to 224×632 pixels while keeping the number of normal and anomalous samples unchanged.

Evaluation metrics. The anomaly detection performance was evaluated based on AP, Precision, and Recall, following the evaluation protocol defined in [9].

Additionally, to evaluate visual similarity between generated images and the original dataset images, we use the Fréchet Inception Distance (FID) [124], a popular metric in the field of image generation which computes the distance between the distribution of two sets of images. More specifically, the Fréchet distance calculates the distance $d(.,.)$ between a Gaussian with mean (m,C) obtained from $p(.)$ and a Gaussian with mean (m_w,C_w) obtained by $p_w(.)$, where $p_w(.)$ represents real-world data and $p(.)$ represents generated data. In practice, these distributions are two sets of data: the "world" data (*i.e.*, the images in a dataset) and the "generated" data (*i.e.*, the generated images). These sets are then fed to an Inception model pre-trained on ImageNet to extract deep features from each sample of the distributions. The resulting two sets of features represent the Gaussians with mean (m_w,C_w) and (m,C) for the "world" and "generated" data, respectively. Specifically, [124] shows that a lower FID score matches a human's higher perceived visual similarity (a lower perceptual distance), meaning that similar sets of images will have a lower FID than dissimilar sets. Formally, the FID score is:

$$d((m,C),(m_w,C_w)) = ||m-m_w||_2^2 + \mathrm{T}(C+C_w-2(CC_w)^{\frac{1}{2}}) \,, \tag{3.9}$$

where T refers to the trace linear algebra operation.

Inpainting via DIAG. We use the pre-trained implementation of SDXL [123] from Diffusers [125] as our text-conditioned LDM. In particular, SDXL shows drastically improved performance compared to previous versions of Stable Diffusion [112] and achieves results comparable to those of commercial state-of-the-art image generators.

Following the procedure outlined in Section 3.3.2, we use the negative images of KSDD2 as the set I_n. As the set of anomaly descriptions D_a, we used the prompts "`white marks on the wall`" and "`copper metal scratches`". Instead, "`smooth, plain, black, dark, shadow`" were used as negative prompts to further improve performance. These prompts were selected through a human-in-the-loop iterative pipeline until the resulting images resembled plausible anomalies. We used the segmentation masks of positive samples in the KSDD2 dataset to simulate the definition of plausible anomalous regions by domain experts.

Then, these data are fed to the pre-trained SDXL model to perform inpainting on the negative images in a training-free process, generating the set of augmented anomalous images I_a as described in Section 3.3.2.

Finally, the generated images I_a are added to the training set, which will be used to train the anomaly detection model.

ResNet-50 training and testing. For a fair comparison with [9], we use the same PyTorch implementation of the ResNet-50 [56] as our anomaly detection model, in which we substitute the fully connected layers after the backbone to make it a binary classifier. The network is trained for 50 epochs with Adam [60] as an optimizer, a learning rate of 0.0001, and a batch size of 32.

To maintain consistency with the training and evaluation procedures of KSDD2, our set-up is the same as that presented in [23, 9], where only the images and ground-truth labels are used to train the model. For our comparison, we use the official code of In&Out [9] to fine-tune a DDPM in the full-shot scenario (on all positive images of the KSDD2 dataset) and generate their augmented images. Likewise, we follow the procedure of MemSeg [114] to generate the "per-region" augmented images. Finally, we generate **DIAG** augmented images, following the inpainting methodology described in Section 3.3. The set of images used for training changes depending on the experiment and the pipelines being tested, but, in general, it can be seen as a combination of the original negative images I_n, an optional set I_p of the original positive images, and the set of generated positive images I_a.

Zero-Shot Data Augmentation

Here, we emulate the situation where ***no*** original positive samples are available in the training set. This scenario makes generating augmented positive samples necessary and restricts the users to augmentation procedures that do not rely on positive images. To do this, we build the set of anomalous augmented images I_a by generating N_{aug} positive augmented samples with different pipelines, *i.e.*, MemSeg, In&Out, and **DIAG**. Then, we train a ResNet-50 on a dataset that includes the original negative samples I_n and the augmented positive samples I_a. Finally, we evaluate the model on the original test set.

Table 3.7 Results between MemSeg, In&Out and **DIAG** when ***no*** anomalous samples are available. In **bold**, the best results. Underlined, the second best.

Model	N_{aug}	AP ↑	Precision ↑	Recall ↑
MemSeg [114]	80	.514	.733	.436
MemSeg [114]	100	.388	.633	.432
MemSeg [114]	120	.511	.683	.470
In&Out [9]	80	.556	.530	.655
In&Out [9]	100	.626	.742	.568
In&Out [9]	120	.536	.699	.534
DIAG (ours)	80	.769	.851	**.673**
DIAG (ours)	100	**.801**	.924	.664
DIAG (ours)	120	.739	**.944**	.609

Table 3.7 reports the comparison between the models trained with MemSeg, In&Out, and **DIAG** augmented data at different values of N_{aug}. As we can see, our proposed method achieves the highest AP (.801), recorded at 100 augmented images, while also resulting in a consistently higher AP compared to the MemSeg and In&Out pipelines. These impressive results highlight how, through domain expertise in the form of anomaly descriptions and segmentation masks, it is possible to generate in-distribution images able to meaningfully guide an anomaly detection network, even in a complicated scenario where no real anomalous data are available.

Surprisingly, the performance of **DIAG** with augmented images $N_{aug} = 120$ is lower than using a smaller number of augmented images. We hypothesize that this is due to the stochastic nature of the LDM image generation. Although it allows the generation of various images given the same guidance, it can also lower, in some cases, the predictability of the quality of the generated samples, which sometimes may not faithfully comply with the prompt. Future works will focus on studying the consistency of quality in the image generation pipeline.

Full-Shot Data Augmentation

To showcase **DIAG** as a general data augmentation technique, we also explore the scenario where real positive samples are available in the training set. To this end, we include all the 246 real positive samples I_p in the training set, together with the real negative images I_n and the N_{aug} augmented positive images I_a.

Table 3.8 Results between MemSeg, In&Out and **DIAG** when ***all*** the anomalous samples are available. In **bold**, the best results. Underlined, the second best.

Model	$\mathbf{N_{aug}}$	**AP** ↑	**Precision** ↑	**Recall** ↑
MemSeg [114]	80	.744	.851	.691
MemSeg [114]	100	.774	.814	.752
MemSeg [114]	120	.734	.772	.707
In&Out [9]	80	.747	.764	.734
In&Out [9]	100	.775	.868	.720
In&Out [9]	120	.782	.906	.689
DIAG (ours)	80	.869	<u>.912</u>	.755
DIAG (ours)	100	<u>.911</u>	**.978**	<u>.800</u>
DIAG (ours)	120	**.924**	.896	**.864**

As we can see from Table 3.8, **DIAG** achieves the highest average AP yet (.924), surpassing the .782 set by the previous state-of-the-art data augmentation pipeline [9]. When comparing these results with the ones obtained in the "zero-shot data augmentation" scenario, it is clear how more in-distribution images improve model performance during training. This is highlighted by the improvement in performance of all models when adding the real positive images I_p to the training set. At the same time, the inclusion of **DIAG** augmented images allows the model to further explore the anomaly distribution, resulting in the difference in performance between the different data augmentation pipelines.

DIAG Qualitative Results

The main goal of our data augmentation pipeline is to generate in-distribution synthetic positive images, meaning images that closely resemble the real ones. Figure 3.7 shows qualitative results. It is evident that the images produced by **DIAG** are markedly more realistic compared to those generated by MemSeg and In&Out.

In addition, we provide a numeric evaluation of the similarity between the generated images and the real ones by employing FID [124]. It is worth noting that due to the limited number of anomalous images in the original dataset, we are forced to calculate FID on a different network layer, precisely the second max pooling layer. This is a common procedure in cases where the number of images is low, as the calculation requires the number of samples (images) to be higher than the number of

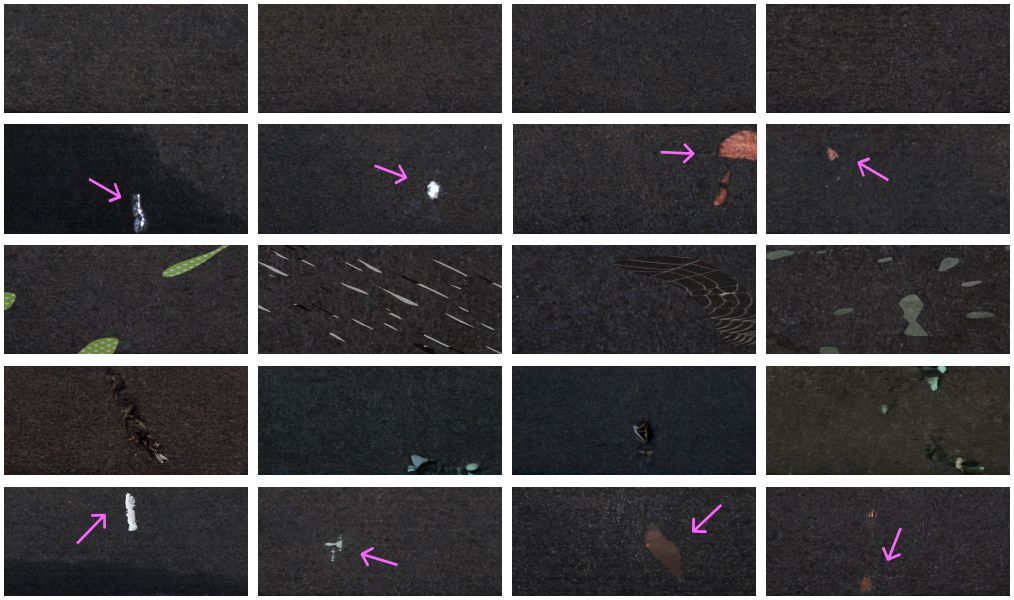


Figure 3.7 First row displays some negative samples from the KSDD2 dataset. Instead, the second row shows some images of positive samples from the same dataset. In the third row, we show the MemSeg-generated defect samples. The fourth row shows **In&Out** generated defect samples. Lastly, the final row showcases images generated with **DIAG**. Notably, the defect images that **DIAG** generated are more realistic and in-distribution.

Table 3.9 FID scores between the real positive images of KSDD2 and the images generated by MemSeg, **In&Out** and **DIAG**. The scores are calculated using the first and second max pooling layers of the Inception network, having 64 and 192 features, respectively. In **bold**, the best results.

Augmentation procedure	**FID 64** ↓	**FID 192** ↓
MemSeg [114]	0.834	4.376
DDPM [9]	0.334	1.520
DIAG (ours)	**0.096**	**0.411**

features. Note that this only changes the magnitude of the values obtained, not the general behavior of the metric. In the specific case of KSDD2, we choose the first and second max-pooling layers with 64 and 192 features, respectively. Specifically, we compare the images generated with MemSeg, In&Out, and **DIAG** with the ones available in the KSDD2 dataset and compute the FID scores between the positive images from the KSDD2 and the previously mentioned sets of augmented images.

The results, reported in Table 3.9, highlight how **DIAG** can generate images that are very similar to those originally present in the dataset, resulting in the lowest FID of all other methodologies.

3.4 GAIA

In this section, we report how we employed a *C-GAN* to address the challenge of generating more diverse and realistic data, allowing the creation of DTs to support PdM in the aircraft domain. In particular, C-GAN, an extension of Generative Adversarial Networks (GANs) [126], are a class of neural networks that generate new data samples conditioned on auxiliary information, which can be labels or other forms of data [127]. In this way, the C-GANs models allow the generation of synthetic data tailored to specific conditions, thus enhancing the diversity and representativeness of the training dataset.

Section Organization

In the following, we introduce the C-GAN architecture (Section 3.4.1). Next, we detail the **GAIA** data augmentation technique (Section 3.4.2) and present our DSLG D/R dataset (Section 3.4.3). Then, we provide an explanatory data analysis of the DSLG D/R dataset (Section 3.4.4). Finally, we present the experimental results in Section 3.4.5.

3.4.1 The C-GAN Architecture

Our architecture consists of two main components: the generator G and the discriminator D. The generator G aims to generate realistic data samples x given a condition y, while the discriminator D aims to distinguish between real data samples and those generated by G, conditioned on y.

Specifically, the generator G takes as input a random noise vector $\mathbf{z} \sim \mathcal{N}(0, I)$ and the condition $(\mathbf{y})$. It outputs a generated data sample $\hat{\mathbf{x}} = G(\mathbf{z}, \mathbf{y})$. Then, the discriminator D receives as input either a real data sample $\mathbf{x}$ paired with the condition $\mathbf{y}$ or a generated data sample $\hat{\mathbf{x}}$ paired with the same condition $\mathbf{y}$, and outputs a probability $D(\mathbf{x}, \mathbf{y})$ indicating whether the sample is real or fake.

In particular, the loss functions for G and D are defined as follows:

$$\mathscr{L}_D = -\mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(\mathbf{x},\mathbf{y})] - \mathbb{E}_{\mathbf{z},\mathbf{y}}[\log(1 - D(G(\mathbf{z},\mathbf{y}),\mathbf{y}))] , \tag{3.10}$$

$$\mathscr{L}_G = -\mathbb{E}_{\mathbf{z},\mathbf{y}}[\log D(G(\mathbf{z},\mathbf{y}),\mathbf{y})] , \tag{3.11}$$

where $\mathbb{E}$ is the expected value. Here, $\mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(\mathbf{x},\mathbf{y})]$ represents the expected value of $\log D(\mathbf{x},\mathbf{y})$ over the joint distribution of real data $\mathbf{x}$ and the condition $\mathbf{y}$. Instead, $\mathbb{E}_{\mathbf{z},\mathbf{y}}[\log(1 - D(G(\mathbf{z},\mathbf{y}),\mathbf{y}))]$ represents the expected value of $\log(1 - D(G(\mathbf{z},\mathbf{y}),\mathbf{y}))$ over the joint distribution of the noise vector $\mathbf{z}$ and the condition $\mathbf{y}$.

3.4.2 GAIA Data Augmentation

After training the C-GAN model, we utilized the trained generator G to augment our datasets. By sampling from the noise distribution $\mathbf{z} \sim \mathscr{N}(0, I)$ and conditioning on different $\mathbf{y}$ values, we generated additional synthetic data samples $\hat{\mathbf{x}} = G(\mathbf{z},\mathbf{y})$.

These synthetic samples were incorporated into the original dataset D, resulting in an augmented dataset D' used for future learning-based model training and evaluation.

Unlike traditional augmentation techniques such as Gaussian noise or magnitude warping, our proposed C-GAN can capture complex relationships and dependencies within the data. This behavior is critical in avionic time-series, where interactions between variables can be intricate and non-obvious.

3.4.3 The DSLG D/R Dataset

The primary objective of the DSLG D/R dataset is to support the development and evaluation of LGS fault detection and classification, allowing the analysis of normal and faulty behaviors. The system used to generate the dataset simulates the operation of an aircraft LGS, modeled in Simscape using SimMechanics and SimHydraulics. The system includes two primary actuators: the Main Actuator, responsible for deployment and retraction, and the Lock Actuator, responsible for locking and unlocking the LGS.

The actuators are controlled by a PID controller that manages both deployment/retraction and locking/unlocking, with transitions between these phases being handled by a supervisory state machine composed of four states: deploy, retract, lock, and unlock. The system operates under hydraulic supply pressure, which powers both actuators, accounting for realistic mechanical and hydraulic dynamics during normal and faulty operation modes. Sensors measure actuator positions, pressures, and velocities for each actuator.

The dataset consists of 448 multivariate time-series simulations, each spanning 40 seconds and sampled at 1 kHz, resulting in 40,001 time steps per simulation. Each simulation includes data from 7 channels that represent different measurements of the system. Specifically:

- **Time:** The time variable tracking the 40 seconds.
- **MainActuatorPos:** The position of the main actuator.
- **MainActuatorPressure:** The pressure within the main actuator.
- **MainActuatorVelocity:** The velocity of the main actuator.
- **LockActuatorPos:** The position of the lock actuator.
- **LockActuatorPressure:** The pressure within the lock actuator.
- **LockActuatorVelocity:** The velocity of the lock actuator.

The dataset is divided into 224 healthy simulations and 224 faulty simulations (see Figure 3.8), which are categorized into the following different types of failures:

- **Healthy (label 0):** Normal operation of the system.
- **Combined Fault (label 1):** Simultaneous faults in both the main and the lock mechanisms (71 simulations).
- **Main Friction Fault (label 2):** Faults in the main actuator (71 simulations).
- **Lock Friction Fault (label 3):** Faults in the lock actuator (71 simulations).
- **Hydraulic Supply Fault (label 4):** Faults in the hydraulic system that affect the supply pressure (11 simulations).

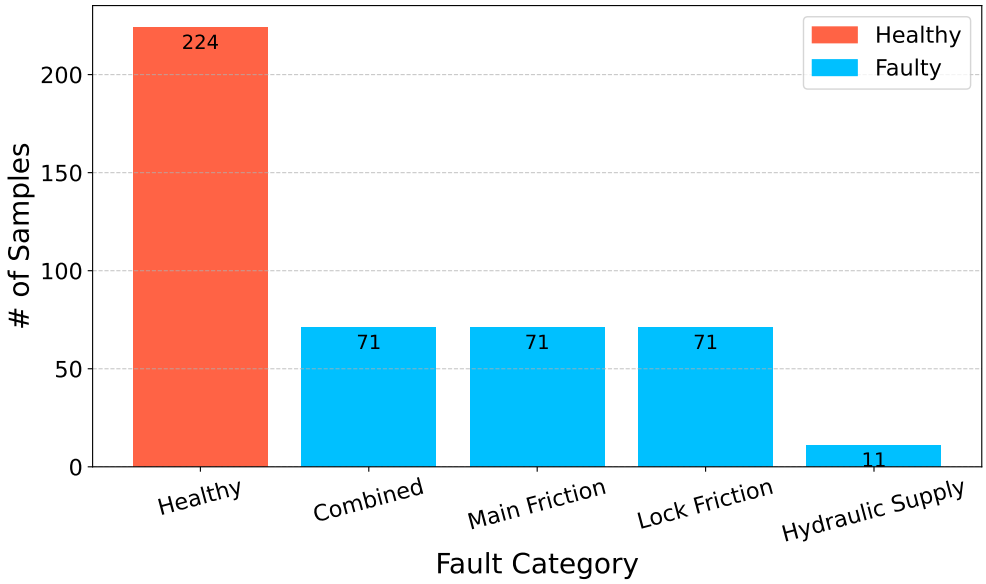


Figure 3.8 Distribution of the healthy and faulty simulations samples divided by class within the DSLG D/R dataset.

3.4.4 DSLG D/R Statistical Data Analysis

We performed an explanatory data analysis to provide more technical details regarding our DSLG D/R dataset and validate it.

First, we account for any null values or duplicates present at the same time t. Next, we visually and numerically analyze the $S(x,t)$ distribution. As the second step, we test the stationarity of the time-series using the Augmented Dickey-Fuller (ADF) test [128]. The results indicate that all time-series considered in this study are stationary, allowing us to proceed with the correlation analysis directly on the raw data without additional preprocessing steps (*e.g.*, detrending).

Given stationarity, we can perform a linear correlation analysis between the dataset's features, as the distribution is time-invariant, which supports standard statistical analysis and modeling. For this purpose, we employ Pearson's correlation coefficient ρ, defined as:

$$\rho = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}, \tag{3.12}$$

where X and Y are random variables, cov denotes the covariance, σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y. The correlation matrix reveals a strong positive correlation between the position, pressure, and velocity of the

main and locking actuators. These results suggest that the actuators' movements are highly synchronized, with changes in one actuator's position leading to corresponding adjustments in pressure and velocity within both actuators. In addition, the matrix indicates a moderate positive correlation between the main and lock actuator positions, suggesting coordinated movement. These findings highlight the interconnectedness of the components and their crucial role in the overall performance of the system, demonstrating that our multiphysics DSLG D / R data set best represents the mechanics and behavior of a LGS.

3.4.5 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Models details. In our experiments, we evaluate the performance of the various augmentation techniques using two widely recognized ML classification models: Support Vector Machine (SVM) [129], and k-Nearest Neighbour (k-NN) [130]. Furthermore, we also implement two common DNN architectures: a Convolutional-1D Neural Network (Conv-1D) and a MLP.

The Conv-1D model comprises two convolutional layers, followed by a ReLU activation function. Then, a pooling layer is applied and the result is flattened before being passed to the fully connected layer.

For MLP, the input is flattened and passed through five fully connected layers, each followed by batch normalization, ReLU activation, and dropout.

Instead, the C-GAN consists of the conditional generator and the discriminator. Specifically, the conditional generator inputs both a noise vector and class labels. The labels are then embedded into a continuous space using an embedding layer. These embeddings are then concatenated with the noise vector and pass through three fully connected layers. The first two layers apply ReLU activation functions, while the final layer applies a tanh activation function to produce the generated output. Like the generator, the conditional discriminator first embeds the class labels, which are then concatenated with the input data. The concatenated input passes through three fully connected layers. The first two layers use ReLU activations, while the

Table 3.10 Accuracy (A), Precision (P), Recall (R) and F1-score (F1) of the SVM, Conv-1D, and MLP models on the DSLG D/R dataset. We compare the results between the vanilla multi-physics, vanilla multi-physics + Gaussian noise, vanilla + magnitude time warping, and vanilla multi-physics + **GAIA** augmentation. In **bold**, the best results. Underlined, the second best.

	SVM				Conv-1D				MLP			
DSLG D/R	**A** ↑	**P** ↑	**R** ↑	**F1** ↑	**A** ↑	**P** ↑	**R** ↑	**F1** ↑	**A** ↑	**P** ↑	**R** ↑	**F1** ↑
Original	64.44	0.537	0.644	0.582	76.66	0.738	0.766	0.745	84.44	0.904	0.844	0.855
+ Gaussian	66.67	0.467	0.666	0.545	81.11	**0.813**	0.811	**0.766**	86.66	0.888	0.866	0.869
+ magnitude	65.56	0.547	0.655	0.591	76.66	0.766	0.766	0.765	65.55	0.678	0.655	0.651
GAIA (ours)	**71.11**	**0.715**	**0.711**	**0.698**	**82.22**	0.730	**0.822**	0.761	**93.33**	**0.940**	**0.933**	**0.931**

final layer uses a Sigmoid activation function to output a probability score indicating whether the input is real or generated.

Datasets descriptions. To effectively showcase the capabilities of **GAIA**, we begin our experiments with our DSLG D/R; see Section 3.4.3 for more details. We performed an 80-20 train-test split, resulting in 358 samples for training and 90 samples for testing.

To further validate the effectiveness of **GAIA**, we applied it to the challenging task of fault classification in electrical power system transmission lines, using the well-established Electrical Fault dataset [131]. In this dataset, the inputs include three voltages and three currents corresponding to the respective three phases, as well as ten different types of faults and no-fault conditions. The training set consists of 6,288 input samples, while the test set has 1,573 samples.

Training details. All the code is implemented in PyTorch [66]. We trained all our classification models for 100 epochs, with a learning rate of 1×10^{-3}, using Adam [60] as the optimizer, on an NVIDIA RTX 3090. Instead, we train our C-GAN model for 500 epochs, with a learning rate of 1×10^{-5}, betas of $(0.5, 0.999)$, always using Adam as an optimizer, on an NVIDIA RTX 3090.

Results on the DSLG D/R Dataset

To evaluate the effectiveness of our **GAIA** data augmentation solution, we compared the results on the vanilla multi-physics, vanilla multi-physics + Gaussian noise,

vanilla + magnitude time warping, and vanilla multi-physics augmented with the C-GAN model. The results are summarized in Table 3.10, showing the accuracy (A), precision (P), recall (R), and F1-scores.

As shown, the vanilla multi-physics dataset alone yields the lowest performance across all evaluation metrics. These results were expected, as relying solely on the original data limits the model's ability to generalize, especially when faced with the complexity of avionic complex real-world systems, and the models were not able to capture the complex dynamics of the LGS of the aircraft.

Adding standard augmentation techniques improves performance compared to the vanilla dataset. However, these techniques have inherent limitations. Although effective in certain scenarios, Gaussian noise and magnitude time warping may not always generate realistic or informative data samples, since they do not explicitly consider the specific characteristics of each of the features of the aircraft system, limiting their effectiveness in capturing intricate nuances. As a result, although performance improves, there is no guarantee that these methods will consistently produce better results.

In contrast, our **GAIA** data augmentation pipeline consistently outperforms the vanilla multi-physics and the vanilla multi-physics + standard augmentation approaches across all metrics. These results demonstrate the superiority of our data augmentation proposal in generating high-quality, realistic, and informative augmented data. By learning the underlying distribution of the original data and leveraging the generative capabilities of C-GAN, **GAIA** can generate data that more closely resemble the underlying distribution of the aircraft system's behavior, enhancing the generalization ability of the trained model, confirming our claims.

The slight decrease in precision and F1-score when using Conv-1D, despite higher accuracy and recall compared to the competitor, may be attributed to the sensitivity of the convolutional model to local patterns in the data. Specifically, Conv-1D excels in capturing temporal dependencies, which can lead to more robust overall detection (as reflected in accuracy and recall). However, this may occasionally result in more false positives, slightly lowering precision and the F1-score. Despite this, the performance drop in precision and F1-score is not critical, as overall accuracy and recall improvements demonstrate that Conv-1D effectively identify relevant patterns. Furthermore, this behavior is not observed with SVM or MLP, suggesting that the trade-off is more related to the specialized architecture of Conv-1D.

Table 3.11 Accuracy (A), Precision (P), Recall (R) and F1-score (F1) of the SVM, k-NN, and MLP models on the Electrical Fault dataset. We compare the results between the vanilla multi-physics, vanilla multi-physics + Gaussian noise, vanilla + magnitude time warping, and vanilla multi-physics + **GAIA** augmentation. In **bold**, the best results. Underlined, the second best.

	SVM				k-NN				MLP			
Elect. Fault	**A** ↑	**P** ↑	**R** ↑	**F1** ↑	**A** ↑	**P** ↑	**R** ↑	**F1** ↑	**A** ↑	**P** ↑	**R** ↑	**F1** ↑
Original	74.70	0.716	0.747	0.721	79.72	0.790	0.797	0.793	79.97	0.714	0.799	0.747
+ Gaussian	75.37	0.726	0.753	0.730	78.70	0.785	0.787	0.785	80.92	**0.852**	0.809	0.776
+ magnitude	73.30	0.696	0.733	0.706	79.40	0.787	0.794	0.789	76.66	0.777	0.766	0.727
+ GAIA (ours)	**76.48**	**0.737**	**0.764**	**0.741**	**82.01**	**0.812**	**0.820**	**0.815**	**81.05**	0.798	**0.810**	**0.802**

Results on the Electrical Fault Dataset

Similarly to the previous experiments, we also compared the performance of our **GAIA** on the Electrical Fault dataset. The results are reported in Table 3.11.

As expected, the vanilla dataset exhibited suboptimal performance. The other key takeaway is that standard augmentation techniques, such as Gaussian noise and magnitude time warping, do not consistently improve performance across all metrics. At the same time, our method consistently leads to better results. For example, when using the SVM model, our method achieves the highest accuracy (76.48%), precision (0.737), recall (0.764), and F1-score (0.741), demonstrating that **GAIA** augmentation provides a significant advantage over traditional techniques. Similarly, for k-NN and MLP, **GAIA** produces superior results in most metrics, confirming its effectiveness across different models.

Although our method slightly loses precision with respect to MLP (0.798 compared to 0.852 for Gaussian noise), this minor drop in precision is not worrisome. The overall gains in other metrics, such as accuracy (81.05%) and recall (0.810), indicate that our method still leads to a more balanced and robust model, emphasizing its superiority for general performance rather than focusing on precision alone.

These findings suggest that **GAIA** augmentation can better capture the complex patterns in the data, leading to improved results where standard augmentation does not do so consistently, confirming our claims also in this challenging scenario.

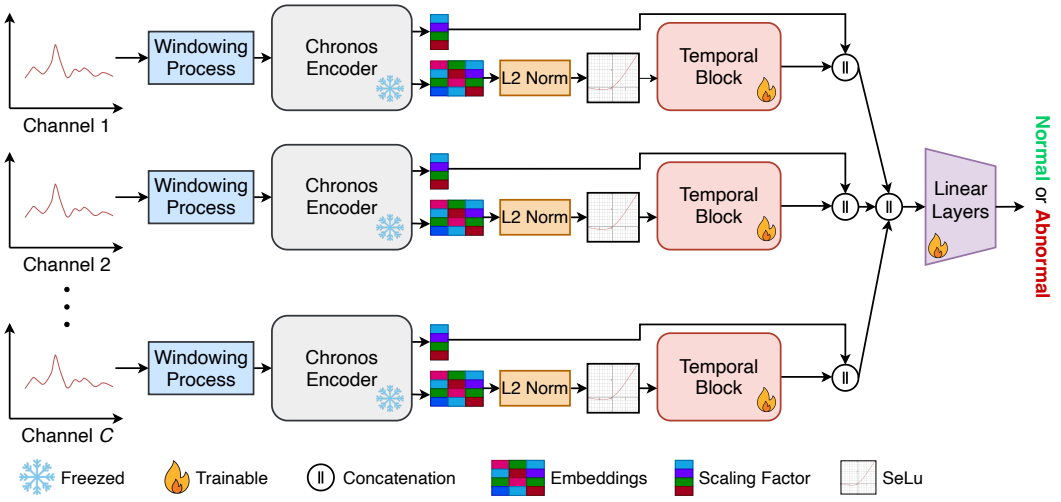


Figure 3.9 The figure illustrates our **ChronosAD** architecture. As a data preparation step, all input time-series data for each channel undergo a windowing process to segment it into smaller chunks. These segments are then passed into the Chronos Encoder. The embeddings are then standardized using L2 normalization to ensure consistent magnitudes. The normalized embeddings are then processed through the Temporal Block, which employs Bidirectional LSTM networks to capture temporal dependencies and a multi-head attention mechanism to enhance contextual understanding. After processing through the Temporal Block, the embeddings from all channels are concatenated into a single feature vector. This concatenated vector is passed through a series of linear layers to produce the final classification, determining whether the input sequence is labeled as normal or abnormal.

3.5 ChronosAD

In this section, we introduce **ChronosAD**, the first framework based on time-series foundation models designed for anomaly detection on time-series data. The overall architecture is illustrated in Figure 3.9.

Section Organization

In the following, we first formalize the problem (Section 3.5.1). Next, we detail the **ChronosAD** training procedure (Section 3.5.2). Finally, we present the experimental results in Section 3.5.3.

3.5.1 Problem Statement

Let the characteristics of the time-series be indicated by $\mathbf{X} \in \mathbb{R}^N$, where N represents the total length of the time-series. The class of time-series on a particular time stamp t is represented by $c_t \in 0,1$, where 0 and 1 correspond to the normal and abnormal classes, respectively.

The time-series is divided into non-overlapping segments, each of length l, referred to as the context length. The duration of the context determines the temporal window through which the model observes patterns to detect anomalies. This segmentation results in $N_l = \lceil N/l \rceil$ segments, each containing l consecutive timestamps.

For each segment, let $\mathbf{C}_l \subseteq c_t : t \in [t, t+l)$ represent the set of classes within that segment. To ensure consistency, segmentation is performed so that all timestamps within a given segment belong to the same class, *i.e.*, $\mathbf{C}_l$ contains only identical class labels $\mathbf{C}_l = c$.

A segment of length l is classified as abnormal if and only if at least one timestamp t within the segment is assigned the abnormal class $(c_t = 1)$. In contrast, if all timestamps in the segment are assigned the normal class $(c_t = 0)$, the segment is classified as normal. To formalize this, the model uses a threshold λ to classify the anomaly score calculated from the features of the segment $\mathbf{x} \in \mathbb{R}^l$.

As a result, the anomaly classification can be expressed as a decision function $f : \mathbb{R}^l \to 0,1$ defined as follows:

$$y = \begin{cases} 1, & \text{if } f(\mathbf{x}) \geq \lambda \\ 0, & \text{otherwise} \end{cases}, \tag{3.13}$$

where $y \in 0,1$ represents the output class, with $y = 0$ indicating a normal segment and $y = 1$ indicating an abnormal segment.

Let $X \in \mathbb{R}^{N \times T \times C}$ represent the input of the time-series, where N is the batch size, T is the sequence length, and C is the number of channels. For each channel c, the input is processed through the following pipeline.

3.5.2 ChronosAD Training Strategy

The Chronos Encoder

Each window channel's time-series $X_c \in \mathbb{R}^T$ is input to the pre-trained Chronos foundation model [132], which outputs a fixed-dimensional embedding vector:

$$\mathbf{E}_c = Encoder(X_c) \ , \quad \mathbf{E}_c \in \mathbb{R}^d \ , \tag{3.14}$$

where d is the embedding dimension.

Furthermore, the Chronos encoder provides a scaling factor s, represented as a vector of size N, corresponding to the batch size. For each instance in the batch, the scaling factor is calculated as the mean of the absolute values of the time-series for a single channel X_c in a context window of length l, weighted by the attention mask A, which represents valid time steps within the window. Formally, the scaling factor is defined as:

$$s = \frac{1}{\sum_{i=t}^{t+l} A_i} \sum_{i=t}^{t+l} (|X_{c,i}| - m) \ , \quad \text{with } s > 0 \ , \tag{3.15}$$

where $X_{c,i}$ is the value of the time-series for channel c at index i, A_i is a binary indicator denoting whether the i-th time step is valid, and m is the mean of the time-series values within the same context window.

The scaling factor plays a crucial role in describing the distribution of the time-series within the specified context length. Specifically, it reflects the natural scale of the data, enabling the model to adapt its processing to varying magnitudes across different time-series, thereby improving its ability to generalize across datasets with diverse characteristics.

The Temporal Block

Before feeding the embeddings into the Temporal Block, L2 normalization is applied to standardize the embedding magnitudes and mitigate the impact of extreme values. This normalization step ensures numerical stability and promotes balanced input representations.

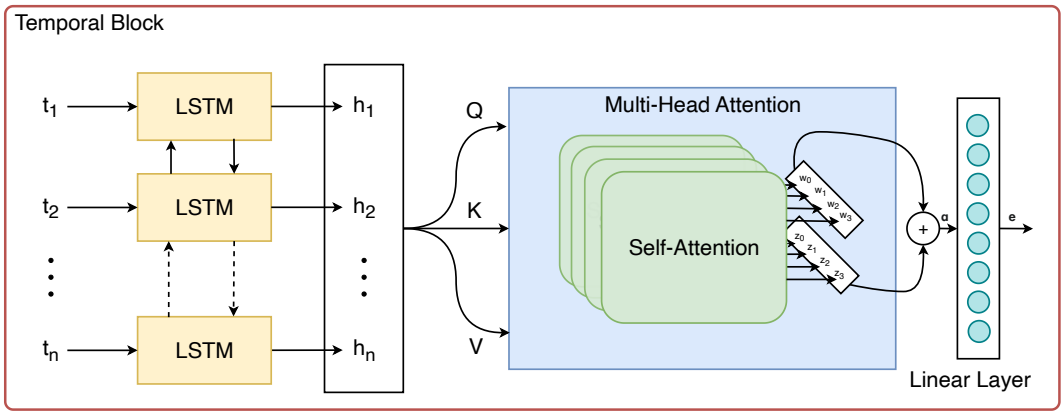


Figure 3.10 This figure illustrates the Temporal Block used in **ChronosAD** to capture temporal dependencies in sequence data. It comprises a Bidirectional LSTM network for encoding forward and backward temporal features, followed by a multi-head attention mechanism to refine the representation and focus on relevant information within the sequence.

Then, the Scaled Exponential Linear Unit (SeLU) activation function is employed to non-linearly transform the embeddings, effectively clamping excessively negative values and shifting them toward a more neutral range. This transformation helps to preserve gradient flow during backpropagation, thus mitigating the problem of vanishing gradients and enabling more efficient learning.

The embeddings are then passed into the Temporal Block (depicted in Figure 3.10), which processes the sequence data to capture temporal dependencies. The Temporal Block consists of two primary components: *i)* a set of Bidirectional LSTM (BiLSTM) networks and *ii)* a multi-head attention mechanism.

Let the input to the Temporal Block be denoted as $\mathbf{E} \in \mathbb{R}^{N \times T \times d}$ where N is the batch size, T is the sequence length, and d is the embedding dimension. Each time step t of the sequence is processed through a BiLSTM to encode forward and backward temporal features. Formally, for a sequence $\mathbf{e}_t$ at time t:

$$\mathbf{h}_t = \text{BiLSTM}(\mathbf{e}_t) \,, \quad \mathbf{h}_t \in \mathbb{R}^{2h} \,, \tag{3.16}$$

where $\mathbf{h}_t$ represents the hidden state for time t, and h is the hidden dimension of the LSTM in each direction. The resulting sequence of hidden states $\mathbf{H} \in \mathbb{R}^{T \times 2h}$ captures bidirectional temporal information.

These encoded temporal features are further refined using the multi-head attention mechanism, which computes the attention-weighted representation of the sequence.

For each head, the queries (Q), keys (K), and values (V) are derived from $\mathbf{H}$ as:

$$Q = \mathbf{H}W_Q \ , \quad K = \mathbf{H}W_K \ , \quad V = \mathbf{H}W_V \ , \tag{3.17}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{2h \times d_a}$ are learnable weight matrices, and d_a is the dimensionality of each attention head.

The attention weights are computed using the scaled dot-product mechanism:

$$\alpha = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_a}}\right) \ , \tag{3.18}$$

and the output of the attention head is given by:

$$\mathbf{z} = \alpha V \ . \tag{3.19}$$

By concatenating the outputs of multiple attention heads, the final attention-enhanced representation $\mathbf{Z}$ is obtained:

$$\mathbf{Z} = \text{Concat}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_h)W_O \ , \tag{3.20}$$

where h is the number of attention heads, and W_O is a learnable projection matrix.

Finally, a Linear Layer transforms $\mathbf{Z}$ into a fixed-dimensional representation for each time step:

$$\mathbf{e} = \text{Linear}(\mathbf{Z}) \ , \tag{3.21}$$

where $\mathbf{e}$ is the output embedding that captures both temporal dependencies and attention-enhanced contextual information.

The Loss Function and Anomaly Score

After processing the embeddings through the Temporal Block, the output $\mathbf{e}$ for each channel is concatenated along the feature dimension. Specifically, for C channels, the concatenated vector $\mathbf{E}_{concat}$ is obtained as:

$$\mathbf{E}_{concat} = \text{Concat}(\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N) \in \mathbb{R}^{N \times d_e} \ , \tag{3.22}$$

where $\mathbf{e}_i$ represents the output embedding for the i-th channel, and d_e is the dimension of the embedding vector.

The concatenated embeddings $\mathbf{E}_{concat}$ are then passed through a Fully Connected Network (FCN) consisting of a series of fully connected layers. The FCN maps the concatenated features to a final prediction, which classifies the sequence as normal or abnormal. Formally, the output of the FCN is given by:

$$\hat{y} = \text{FCN}(\mathbf{E}_{concat}) \,, \tag{3.23}$$

where $\hat{y}$ is the predicted class label, with 0 indicating a normal sequence and 1 indicating an abnormal sequence.

Specifically, the output of the fully connected network, $\hat{y}$, is then used to calculate the loss of Binary Cross-Entropy (BCE) for training. The BCE loss is employed to measure the discrepancy between the predicted probability $\hat{y}$ and the true label y, where $y \in \{0,1\}$ represents the ground truth class (normal or abnormal). Formally, it is defined as:

$$\mathscr{L}_{\text{BCE}}(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] \,, \tag{3.24}$$

where y is the ground truth label, $\hat{y}$ is the predicted probability that the sequence is abnormal, and N is the number of samples. The model is trained by minimizing this loss function to improve the accuracy of the anomaly detection task, effectively adjusting the weights of the Temporal Block and fully connected layers through backpropagation.

3.5.3 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Evaluation metrics. To assess the effectiveness of **ChronosAD**, we consider two key evaluation metrics: the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) and the AP.

The AUC-ROC measures the ability of a model to distinguish between normal and anomalous instances by evaluating the trade-off between the true positive rate and the false positive rate across different decision thresholds. A higher AUC value indicates better discrimination capability, with a perfect classifier achieving an AUC of 1. However, AUC-ROC does not consider the actual confidence scores of the predictions beyond their order.

To complement this, we use AP, which is derived from the Precision-Recall (PR) curve and provides a more informative measure when dealing with imbalanced datasets, as is often the case in anomaly detection. AP is computed as the area under the PR curve, summarizing precision at different recall levels and placing greater emphasis on correctly identifying the minority class. Together, AUC-ROC and AP provide a comprehensive evaluation of the performance of the model in distinguishing anomalies from normal patterns.

Datasets descriptions. For our experiments, we used a diverse collection of datasets, including seven time-series datasets from the UCR Archive [133], a real-world bearing dataset (CWRU) [145], an ECG arrhythmia data set (MIT-BIH) [146], a water treatment dataset (SWaT) [147], and a simulated dataset (Waveform) [148]. These datasets were chosen to evaluate the performance of the proposed method in varying characteristics such as the number of samples, dimensions, classes, and sampling rates (Hz). In the following, we provide detailed descriptions of each dataset.

PPOC [133]: This dataset contains temporal representations of the proximal finger bone outlines. The data is categorized into accurately and inaccurately described bone shapes by mapping the distances between the bone center and its outline.

TwoLeadECG [133]: Derived from PhysioNet, this dataset consists of electrocardiogram (ECG) signals from two leads, divided into two distinct categories of cardiac activity.

Strawberry [133]: This dataset represents spectral data for food, with two categories differentiating strawberry from non-strawberry samples.

TwoPatterns [133]: A synthetic dataset that simulates trend changes over time, with four categories: “down-down”, “up-down”, “down-up”, and “up-up”.

Table 3.12 Anomaly Detection Performance. All values are percentages (%) and averaged with stddevs over five seeds. In **bold**, the best results. Underlined the second best. In parentheses, there is a change in performance over the second-best.

Dataset	**PPOC** [133]		**TwoLeadECG** [133]		**Strawberry** [133]		**TwoPatterns** [133]	
	AUC ↑	**AP** ↑	**AUC** ↑	**AP** ↑	**AUC** ↑	**AP** ↑	**AUC** ↑	**AP** ↑
AnoGAN [134]	52.30±1.27	71.88±12.28	52.04±7.86	73.76±5.90	61.97±11.63	74.36±7.5	51.46±1.12	88.89±0.41
ALAD [135]	50.39±14.85	67.07±23.26	51.64±10.76	72.80±9.02	52.93±8.85	69.69±19.99	50.50±4.65	88.23±2.18
Deep-SVDD [136]	65.73±4.53	82.08±7.36	85.56±1.52	89.82±1.43	72.80±13.66	73.74±22.31	83.61±8.30	97.03±1.47
BeatGAN [137]	86.02±2.92	74.40±13.65	78.42±10.79	74.62±8.70	91.38±0.27	89.66±7.94	75.85±1.95	88.60±0.7
GOAD [138]	66.82±8.32	71.59±10.66	52.97±11.68	54.90±10.10	55.69±27.79	61.32±24.42	74.07±3.07	77.39±2.79
USAD [139]	74.09±5.20	84.49±5.93	67.37±15.60	80.55±10.28	63.45±27.84	68.73±28.89	85.50±4.15	97.61±0.85
TLKF [140]	75.05±8.24	84.61±4.43	71.49±11.44	82.55±9.14	72.88±8.61	79.84±10.67	75.19±9.93	85.22±5.47
GTA [141]	80.31±6.11	84.30±5.32	72.60±0.08	80.05±2.81	81.00±8.12	85.87±6.15	84.91±5.11	96.48±1.11
NSIBF [142]	75.47±8.43	82.23±4.28	88.09±1.48	90.78±2.20	76.44±21.82	72.24±21.24	88.71±0.32	92.74±1.94
ECOD [143]	81.86±4.13	92.58±4.49	82.68±8.94	88.19±8.10	82.59±4.91	88.78±7.11	90.91±2.47	95.85±1.23
KalmanAE [144]	94.53±0.14	96.37±1.08	91.14±4.42	94.12±3.54	93.92±0.92	92.45±2.54	94.28±1.43	99.02±0.26
ChronosAD (ours)	**96.05** (+1.52)	**98.05** (+1.68)	**100** (+8.86)	**100** (+5.88)	**98.18** (+4.26)	**99.05** (+6.60)	**100** (+5.72)	**100** (+0.98)

Dataset	**UWaveGestureLibraryY** [133]		**FordA** [133]		**SmallKitchenAppliances** [133]		**CWRU** [145]	
	AUC ↑	**AP** ↑	**AUC** ↑	**AP** ↑	**AUC** ↑	**AP** ↑	**AUC** ↑	**AP** ↑
AnoGAN [134]	82.74±7.41	98.52±0.84	50.27±4.45	71.02±4.05	48.76±14.60	83.48±5.76	72.47±8.97	87.13±1.62
ALAD [135]	52.64±20.09	94.55±3.52	50.22±1.00	71.68±4.10	51.94±2.69	84.40±1.54	47.94±0.35	91.84±0.13
Deep-SVDD [136]	65.68±4.45	96.38±0.64	64.54±8.26	80.48±7.96	74.17±10.65	91.68±4.36	92.09±1.31	99.02±0.17
BeatGAN [137]	91.46±4.65	97.73±1.70	73.35±10.22	84.00±7.71	83.40±5.11	82.11±0.33	97.91±0.52	95.84±1.46
GOAD [138]	87.28±5.93	96.87±0.85	54.65±0.13	73.66±0.75	48.89±2.44	52.39±2.21	53.20±2.11	58.17±1.32
USAD [139]	90.06±5.10	99.30±0.39	63.81±1.21	79.83±2.42	62.31±5.10	85.96±1.76	85.86±7.37	97.88±2.53
TLKF [140]	79.09±6.87	88.32±3.87	62.54±7.43	71.64±6.53	67.91±8.14	76.73±8.65	86.43±5.84	95.61±1.75
GTA [141]	89.43±5.28	92.67±5.32	74.28±10.8	84.90±8.45	76.05±8.64	87.96±7.98	95.57±2.54	96.78±2.65
NSIBF [142]	92.63±0.85	97.58±0.88	73.29±5.99	82.72±1.92	84.66±6.54	91.95±5.93	97.18±2.51	98.08±1.15
ECOD [143]	93.50±4.07	97.00±1.23	75.28±6.47	84.80±6.33	83.78±6.74	91.18±4.32	97.50±3.46	98.57±0.25
KalmanAE [144]	95.67±1.49	99.59±0.38	79.30±3.14	87.64±7.43	89.48±3.14	93.32±2.58	99.02±0.07	99.68±0.03
ChronosAD (ours)	**98.38** (+2.71)	**99.75** (+0.16)	**97.43** (+18.13)	**97.29** (+9.65)	**89.79** (+0.31)	**94.08** (+0.76)	**100** (+0.98)	**100** (+0.32)

Dataset	**MIT-BIH** [146]		**SWaT** [147]		**Waveform** [148]		**Average**	
	AUC ↑	**AP** ↑	**AUC** ↑	**AP** ↑	**AUC** ↑	**AP** ↑	**AUC** ↑	**AP** ↑
AnoGAN [134]	75.33±1.46	58.75±1.67	61.78±1.20	77.43±0.77	52.91±4.48	34.89±0.35	60.18±5.85	74.55±3.74
ALAD [135]	71.66±0.33	62.07±0.29	77.52±4.79	87.02±3.45	68.35±1.43	35.50±0.14	56.88±6.34	74.98±6.14
Deep-SVDD [136]	85.76±0.81	66.64±0.91	87.04±0.77	88.65±0.34	84.68±2.09	45.12±7.27	85.33±1.28	82.78±4.92
BeatGAN [137]	92.99±0.87	89.87±1.23	89.98±0.02	89.99±0.01	83.19±2.97	44.27±4.85	88.36±1.81	82.62±4.38
GOAD [138]	72.70±0.52	87.61±0.23	63.16±5.11	70.49±3.67	77.08±1.00	37.49±1.65	64.21±5.23	67.44±5.33
USAD [139]	88.91±3.02	76.00±3.59	87.43±2.61	92.62±1.83	85.56±2.40	42.41±2.30	87.32±1.66	82.30±5.52
TLKF [140]	80.34±2.67	72.85±7.63	72.49±1.37	79.38±2.85	64.13±2.35	43.11±1.76	73.41±6.62	78.16±5.70
GTA [141]	72.96±4.63	62.78±7.01	91.32±0.29	94.12±0.13	63.85±2.63	39.12±1.66	81.11±4.93	82.27±4.41
NSIBF [142]	84.53±2.20	72.88±1.44	95.76±0.63	96.47±0.80	62.25±4.93	45.20±6.83	83.54±5.06	83.89±4.41
ECOD [143]	80.44±0.33	70.39±0.62	82.73±0.07	79.43±0.07	74.88±9.21	40.09±1.36	84.61±4.09	82.46±3.19
KalmanAE [144]	95.06±0.71	91.48±1.73	97.32±0.06	98.63±0.09	87.04±2.69	46.15±1.30	92.43±1.65	90.76±1.90
ChronosAD (ours)	**97.59** (+2.53)	**96.95** (+5.47)	**98.84** (+1.52)	**98.81** (+0.16)	**92.41** (+5.37)	**86.93** (+40.78)	**97.15** (+4.72)	**97.36** (+6.59)

UWaveGestureLibraryY [133]: A dataset focused on gesture recognition based on accelerometer data, containing eight categories corresponding to different hand gestures.

FordA [133]: This automotive dataset is used for the diagnosis of faults in vehicle subsystems, with two categories indicating the presence or absence of faults.

SmallKitchenAppliances [133]: A dataset that profiles the power consumption of kitchen appliances, categorized into three classes: kettles, microwaves, and toaster ovens.

CWRU [145]: A bearing fault diagnosis dataset with four categories: drive and fan bearing faults recorded at 12 kHz, drive-side bearing faults recorded at 48 kHz, and normal data.

MIT-BIH [146]: This arrhythmia dataset contains 48 half-hour recordings of dual-channel ambulatory ECG data from 47 individuals, sampled at 360 Hz. It is annotated with normal beats and abnormal patterns, including bundle branch block, atrial premature beats, and ventricular premature beats.

SWaT [147]: A dataset for the water treatment control system for safety and attack detection, consisting of data from 51 sensors and actuators, classified into normal and attack states.

Waveform [148]: A simulated dataset generated by the Waveform Database Generator that comprises three types of waveform. Category 0 is designated as an anomaly, while the remaining categories are considered normal.

Competitors descriptions. To assess the performance of **ChronosAD**, we compared it with eleven state-of-the-art methods. The specifics of these baseline models are outlined below.

AnoGAN [134]: A generative adversarial network that takes advantage of normal samples to learn a multiple distribution in the latent space. Identifies anomalies by measuring the reconstruction error generated by the model.

ALAD [135]: A GAN-based anomaly detection framework that employs cyclic consistency loss and multiple discriminators to stabilize adversarial training. Detects anomalies using reconstruction errors in both the input and latent spaces.

Deep-SVDD [136]: Learns a neural network-based support vector data description of normal data. It determines anomalies by calculating the distance of samples from the center of a hypersphere enclosing the feature space.

BeatGAN [137]: A GAN specifically designed for time-series data. Processes ECG signals by splitting them into fixed segments and utilizes an encoder to capture latent representations for anomaly detection.

GOAD [138]: A classification-based anomaly detection model that integrates one-class classification with transformation-based methods. Detects anomalies by evaluating the probability that transformed samples are in their designated subspaces.

USAD [139]: A two-stage adversarially trained autoencoder comprising an encoder and two decoders. It offers a robust, unsupervised approach with high stability during adversarial training.

TLKF [140]: Integrates transformers and LSTM networks to improve the state estimation capabilities of Kalman filter-based systems.

GTA [141]: A comprehensive framework to detect anomalies in multivariate time-series. It models inter-channel relationships and temporal dependencies using propagation convolution and multi-branch attention mechanisms.

NSIBF [142]: A hybrid anomaly detection model for CPSs. The architecture combines LSTM networks for dynamic system modeling with Bayesian filters to pinpoint anomalies within a dynamic state space.

ECOD [143]: Employs a non-parametric strategy for the detection of anomalies by estimating the underlying data distribution. Anomalies are identified as rare instances that occur in the tails of the distribution.

KalmanAE [144]: A deep-implementation-optimized Kalman filter for unsupervised time-series anomaly detection, where the state of the system of a normal time-series can be fit using the embedding-optimized Kalman filter in an unsupervised manner, and anomalies can be detected from data points that deviate from the normal system state.

ChronosAD Quantitative Results

As we can see from 3.12, **ChronosAD** consistently outperforms all competitors presented in all scenarios, often achieving near-perfect results.

In UCR datasets [133] such as TwoPatterns, UWaveGestureLibraryY, and FordA, **ChronosAD** achieves AUC values close to or above 100, setting a new benchmark for performance in time-series anomaly detection. For example, its flawless performance on TwoPatterns and near-optimal results on UWaveGestureLibraryY emphasize its precision in detecting nuanced deviations across synthetic and gesture-based time-series data. Similarly, its results on FordA, a data set geared toward automotive fault detection, further underscore its relevance for applied diagnostic tasks.

Real-world data sets such as CWRU [145] and SWaT [147] further validate the robustness of **ChronosAD**. Specifically, on CWRU, the model attains perfect scores,

reflecting its ability to precisely identify faults in industrial bearing systems. For SWaT, a challenging multivariate dataset representing CPSs, **ChronosAD** achieves unmatched precision, highlighting its ability to model complex interdependencies across multiple data channels. Likewise, MIT-BIH [146] excels in detecting anomalies in high-resolution ECG recordings, a critical requirement in medical diagnostics.

In summary, **ChronosAD**'s consistent performance across benchmarks, including the simulated Waveform dataset, reaffirms its generalizability. Its averaged AUC of 97.36 and AP of 96.59 places it ahead of established methods such as KalmanAE [144], indicating not just incremental but substantial advances in anomaly detection. These results reflect **ChronosAD**'s ability to balance sensitivity and specificity while navigating diverse data modalities, sampling rates, and dimensional complexities.

Chapter 4

Sustainable New Products Performance Forecasting

The fast fashion industry represents the second most polluting industry in the world, responsible for 79 trillion liters of water consumed and 92 million tonnes of waste produced per year [33], contributing 8% of all carbon emissions and 20% of all global wastewater [34]. Having the ability to predict sales volumes for an unreleased product with greater precision could represent a significant step toward making this market more efficient, reducing the use of production resources, and especially minimizing unsold inventory [13]. Although forecasting time-series with a known historical past has been extensively analyzed [149, 150], very little attention has been paid to a much more practical and challenging scenario: forecasting new products that the market has not seen before.

Specifically, this problem, known as NFPPF [151], is far from trivial, as unreleased products do not have sales data available. Thus, it is necessary to retrieve valuable information from the available data, which may be technical specifications of the product (such as color, type, material), release period, or interest shown for similar products in the past [151].

Due to the rapidly changing nature of fashion trends, determining what is considered fashionable or outdated can be challenging. This makes it difficult to accurately predict the market performance of a specific item and identify the key factors that influence its popularity. Traditional deterministic forecasting models have shown reasonable performance in specific situations. However, they are limited by their

assumption that the characteristics of past-season products are directly applicable to new items, which often exhibit distinct features. This leads to inaccurate predictions as a result of the change in the characteristics of the input data.

Recently, DDPMs [116], or Diffusion Models in general, have impressed with their realistic image and video generation capabilities. Moreover, DDPMs have shown promising results even in the context of time-series analysis [152]. Specifically, DDPMs implicitly learn the probability distributions of data, such as images [9, 10], or fashion sales, as we demonstrate in this research. As a result, they are not affected by the issue of the feature domain shift due to the nature of the diffusion process [115]. In fact, a DDPM is gradually trained to denoise Gaussian noise; therefore, it does not use explicit features extracted from a specific sample as input.

In particular, when a DDPM comes across a sample with features that are not in its training data (a common occurrence in the fast-fashion industry), its denoising diffusion process naturally keeps its predictions within the real sales distribution. Furthermore, even when DDPM is conditioned on extracted features, the model's predictions do not diverge significantly [153]. In contrast, deterministic approaches do not have this kind of reliability because they directly link input features to predicted sales, which can result in wrong predictions for new feature combinations. This characteristic is critically important, making DDPMs an ideal tool for NFPPF.

In Figure 4.1, we present all publications related to sustainable new products performance forecasting. In the following section, we outline the relevant literature and the motivation behind each work. Then, we provide a detailed explanation of each work, encompassing both methodologies and the achieved results.

Motivations for MDiFF [14] (Section 4.2). In this chapter, we present **MDiFF**, a two-stage architecture specifically crafted to tackle NFPPF. We first train a multimodal score-based diffusion model that learns to generate samples from the true sales distribution. A second refinement model, based on an MLP, is then used to refine the prediction DDPM. DDPMs, by nature, generates output from Gaussian noise, guaranteeing a fundamental property, which is the generation of non-deterministic samples. In order to better control this behavior and have more stable results, what we do is generate multiple sales signals for the same object. Specifically, 50 samples are generated and given as input to the refinement model, which subsequently gener-

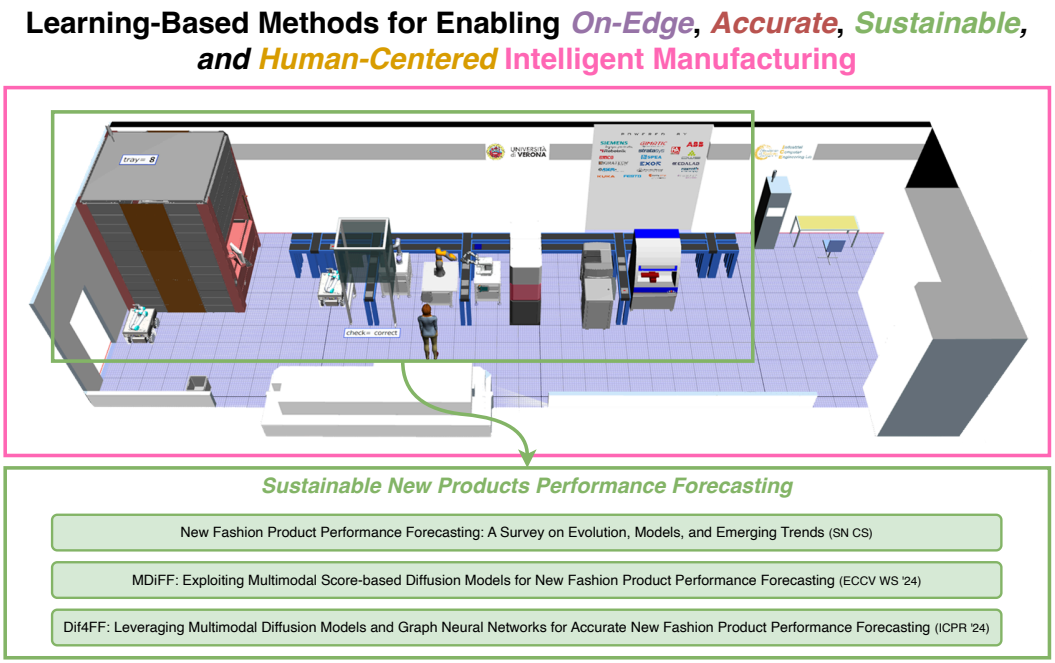


Figure 4.1 Overview of research contributions on sustainable new products performance forecasting.

ates the final sales prediction. This strategy ensures that MLP receives data in the distribution that matches more closely the actual sales data distribution, leading to improved pipeline reliability.

Motivations for Dif4FF [15] (Section 4.3). The previous work was later extended in **Dif4FF**, in which we conditioned the multimodal score-based diffusion model on more data, and the refining module was replaced by a Graph Convolutional Network (GCN). Specifically, our GCN builds two types of graph from the input. One graph focuses on the time dimension, highlighting important connections among weak sales. Then, we create another graph based on prediction space, pinpointing strong connections among model-predicted samples. Finally, we use three Conv1D layers to compress the graph network's output and generate the prediction vector. This allowed us to process the expanded multimodal input more effectively.

4.1 Related Works

The existing literature on NFPPF with deep learning tools is limited but growing. One of the first articles to investigate the sales forecasting problem is [154]. Following, [155] attempts to emulate an expert's decision-making process, training a ConvNet-based model to extract visual features from the image and then, through the k-Nearest Neighbors (k-NN) algorithm, confront the features with other elements already seen to produce the final prediction of the sales.

With [156, 157], there was a first attempt at tackling this task by exploring various algorithms. In particular, [156] compares several ML algorithms, such as gradient boosting and random forest, discussing which is better for NFPPF. Moreover, the authors also tried two deep learning approaches, *i.e.*, Feed-Forward Networks (FFNs) and LSTMs, both fed with multimodal signals. Those signals were obtained from static attributes of the items, such as category, color, fabric, and variable information, such as discounts and promotions. Similarly, [157] uses an architecture based on RNNs, including more signals such as past sales, images, textual embeddings, and discounts. The model also operates a soft-attention mechanism to understand which information is more relevant to produce the predicted sales signal. However, their autoregressive model produced the same prediction between products of different seasons. Unfortunately, the authors' code and dataset are proprietary and not publicly available.

Building on the value of exogenous signals in fashion forecasting, [151] propose an encoder-decoder Transformer-based architecture that incorporates as input of the model all the multimodal data offered by the dataset [158]. The encoder is fed with Google Trend signals, while the decoder receives an ensemble of features extracted from images, textual descriptions, and the item's temporal information (release date). This approach effectively extends previous work by using a more powerful architecture to extract insights from exogenous signals, showing the effectiveness of Google Trends information in relation to NFPPF.

In contrast to previous approaches, this research introduces the first diffusion model-based implementation to solve the NFPPF task. In this way, we solve the problem common to all the previous methods: unrealistic predictions due to the shift in the input feature domain.

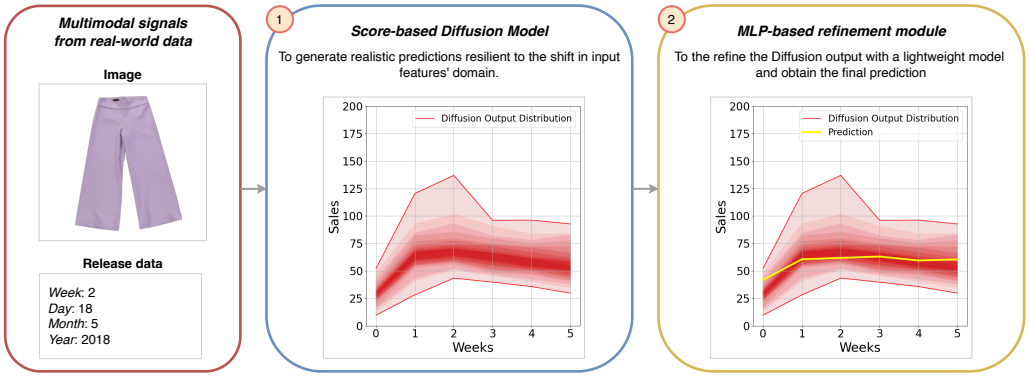


Figure 4.2 The **MDiFF** two-stage pipeline for NFPPF. Starting from multiple signals of a single fashion product, we build a multimodal score-based diffusion model to generate an initial prediction of the sales, addressing potential objects with features beyond the training distribution. Then, we refine the diffusion output using a lightweight MLP to obtain the final prediction.

Datasets for NFPPF. Publicly available datasets for fashion forecasting, such as [159], take into account applications that are different from NFPPF. They have usually been used to forecast fashion styles, which are aggregates of products of multiple brands in terms of popularity based on social networks, such as, for example, Instagram. In our case, the task is different since we focus only on single products and not on groups of products, so we have fewer data to reason on. In addition, we are considering genuine sales data and not popularity trends. As a result, in our research, we use the VISUELLE dataset [158, 151], the only dataset available in the literature for NFPPF, and thus the de facto standard for this task. Due to its nature, our research is also impactful from an industrial level.

4.2 MDiFF

In this section, we introduce **MDiFF**, a two-stage architecture shown in Figure 4.2 specifically crafted to tackle NFPPF.

Section Organization

In the following, we introduce the formalization of the problem (Section 4.2.1), our score-based diffusion models (Section 4.2.2), the methodology used to guide sample

generation (Section 4.2.3), the MLP-based refinement stage (Section 4.2.4), and the experimental results (Section 4.2.5).

4.2.1 Problem Formalization

Given a new product j, we want to predict $y \in \mathbb{R}^W$ expressed as the performance vector in terms of sales in an interval of W weeks since its release date.

For every j, a set of 2 attributes is given: an image of the product $i_j \in \mathbb{R}^{w \times h \times 3}$ with $w = h = 256$ and the release date $t_j \in \mathbb{R}^4$ composed of four digits representing the day, week, month and year of release.

4.2.2 Our Score-Based Diffusion Model

Score-based diffusion models [117] generalize DDPMs [116] generative models trained to reverse a discrete-time diffusion process. A Gaussian noise diffusion process, also known as the *forward process*, can be summarized as a chain of steps in which Gaussian noise is progressively added to the initial distribution, as described by the following equations:

$$q(x^1, ..., x^T | x^0 = y) = \prod_{t=1}^{T} q(x^t | x^{t-1}) \,, \tag{4.1}$$

$$q(x^t | x^{t-1}) := \mathcal{N}(\sqrt{1-\beta_t} x_{t-1}, \beta_t I) \,, \tag{4.2}$$

where $q(x^T) \approx \mathcal{N}(0,1)$, $y = q(x^0)$ is the true data distribution, β_t is the variance of the additive noise, and $t \in [0,T]$ represents the number of noising steps defined a prior.

A model p_θ is then trained to reverse the diffusion process by gradually removing noise, also known as *backward process*, to restore the initial distribution. Specifically, the backward process is formalized as follows:

$$\begin{aligned} p_\theta(x^{t-1} | x^t, c) = \mathcal{N}(x^{t-1}; \mu_\theta(x^t, t) \\ + s\sigma_t^2 \nabla_{x^t} \log p(x^t | x^0), \sigma_t^2 I) \,, \end{aligned} \tag{4.3}$$

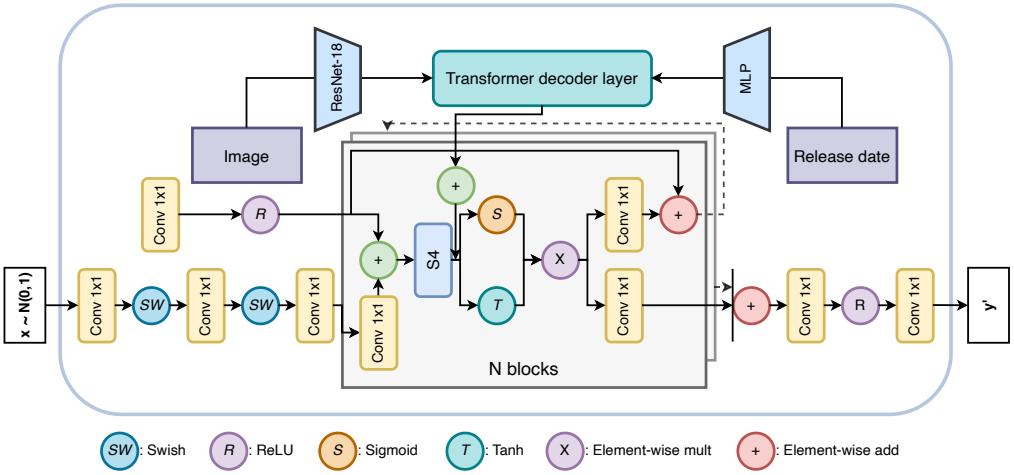


Figure 4.3 An overview of our multimodal score-based diffusion model. The diffusion basic block is taken from TS-Diff [21] (grey square), modified to be injected with the output of the transformer decoder layer, a module responsible for producing an embedding representing the two modalities of input related to the item. Each block contains two outputs: one for the subsequent block and another for a skip connection. The summation of all skip connections forms the model's final output. The primary component of each block is typically an S4 block [22], chosen by the authors of [21] for its efficiency when it comes to time-series and structured data. The input of the **MDiFF** is noisy data, and the output is the denoised sample.

where σ is the variance for each timestep, and s is the parameter that controls the strength of the conditioning.

Specifically, Figure 4.3 shows the architecture of our multimodal score-based diffusion model. The network is a stack of multiple N blocks. Every block has two outputs, one for the next block and one for a skip connection. The sum of all the skip connections represents the actual output of the model. Every block is mainly made up of an S4 block [22]. The *multimodal conditioning* c_j is added directly to the output of the S4 block, and the cross-attention is implemented using a Transformer decoder layer [160].

4.2.3 Multimodal Conditioning

To guide the generation of future sales of the j-th product by the diffusion model, we use multimodal data composed of images i_j of the product and the date of release t_j, as described in Section 4.2.1.

We train two different encoders I_θ, T_θ to extract features from the images and release dates, respectively. Then, these features are used to produce the conditional embedding through a cross-attention mechanism defined as:

$$c_j = \text{Softmax}\left(\frac{Q_j K_j^T}{\sqrt{d_k}}\right) V_j \ , \tag{4.4}$$

where $K_j = V_j = I_\theta(i_j)$, $Q_j = T_\theta(t_j)$ and $\sqrt{d_k}$ is the dimensionality of the number of features of the embeddings.

We designed the conditioning module of **MDiFF** to weight, through the attention mechanism, the embedding of the image $I_\theta(i_j)$ with the features $T_\theta(t_j)$ obtained from the temporal information given by the release date. The idea behind this architectural choice came from the fact that every visual feature of the item must be considered with respect to the fashionable concept of the current season to effectively guide the reverse diffusion process. To effectively and efficiently use the cross-attention mechanism, we used a Transformer decoder layer to serve this scope. More details on the implementation of the various encoders are reported in Section 4.2.5

4.2.4 MLP-Based Diffusion Outputs Refinement

We approach the refinement stage as a regression task to predict a continuous output value. The model is designed to reduce the dimensionality of all 50 predictions of the diffusion model, analyzing both the feature and the temporal dimensions. Given $x \in \mathbb{R}^{WxN}$ the set of predictions of the diffusion model, y the true sales signal and $\hat{y}$ the output of the model with $y, \hat{y} \in \mathbb{R}^{1xW}$, the model is defined as follows:

$$\begin{aligned} x_t &= \phi_t(x) \ , \\ \hat{y} &= \phi_n(x_t') \ , \end{aligned} \tag{4.5}$$

where $N = 50$ is the number of generated samples of the diffusion model, $W = 6$ is the number of weeks of prediction, $\phi_t(x)$ the temporal MLP defined as $\phi_t(x) = W_t x + B_t$ and $\phi_n(x) = W_n x + B_n$ the MLP that regress the final sales signal.

Specifically, for $\phi_t(x)$, $W_t \in \mathbb{R}^{WxW}$ and $B_t \in \mathbb{R}^{WxN}$; on the other hand, for $\phi_n(x)$ we have $W_n \in \mathbb{R}^{1xN}$ and $B_n \in \mathbb{R}^{1xW}$. In this case, the dimensionality W is the number of weeks to predict, while W_t, W_n are the weight matrices of the two MLPs.

Specifically, we used an MLP network trained with a Mean Squared Error (MSE) loss function denoted as $\mathscr{L}$MSE. The mean squared error loss measures the dissimilarity between the predicted output and the ground truth. Mathematically, it is defined as:

$$\mathscr{L}_{\text{MSE}}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \,, \tag{4.6}$$

In detail, y_i denotes the true output value for sample i, while $\hat{y}_i$ signifies the predicted output value for sample i. More details about the structure of the refinement module are explained in Section 4.2.5.

4.2.5 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Implementation details. The backbone of our model is a TS-Diff [21] architecture, a multi-purpose score-based diffusion model developed for predicting, reconstructing, and refining univariate time-series data. TS-Diff consists of a series of $M = 4$ S4 Blocks [22] layers, connected via skip connections. We have extended the model to be conditioned as described in Section 4.2.3, using a Transformer decoder layer [160] to implement the multi-head cross-attention used to ensembling the two embeddings.

Image encoder. We used as image encoder I_θ a ResNet-18 [56] pre-trained on ImageNet-1K [73]. We substituted the last two layers of the model with a Conv1D and a Linear to reduce the dimensionality of the features extracted, obtaining a tensor $I_\theta(i_j) \in \mathbb{R}^{C\times W}$, with C channels equal to 64 and W forecasting horizon of six weeks.

Temporal encoder. The temporal encoder T_θ comprises four different MLPs that expand the dimension from 1 to C. The output of this model is then concatenated along the channel dimension and fed into another MLP that reduces the number of features from $4C$ to C, resulting in $T_\theta(t_j) \in \mathbb{R}^C$.

MLP-based refinement. The refinement network is based on two MLPs working on different dimensions of the input tensor. The first part is a stack of five linear layers working on the temporal dimension of the input (*i.e.*, the six weeks of prediction), expanding and compressing the feature space to match the same dimensionality of

the input. The second part comprises three other linear layers, operating on the sample's dimensionality. These layers gradually reduce and compress the $N = 50$ predictions to achieve the actual prediction. We used the ReLU as an activation function, with a skip connection between the input tensor and the output of the first MLP.

Dataset description. We used the VISUELLE fast-fashion dataset [158, 151] to test our proposal. The dataset provides a comprehensive collection of fashion products and consumer behavior data. It encompasses three main components: product information, customer data, and market trends.

Product information includes detailed descriptions of individual items. This involves visual representations in the form of high-resolution images that showcase the product on a plain background. In addition, textual attributes such as product category, color, fabric, and release date are provided.

Customer data offers meaningful insights into consumer preferences and purchasing habits. It contains anonymized information on a large number of customers, including their purchase history, specific items purchased, purchase dates, and stores where purchases were made.

Finally, market trend data are incorporated into the Google Trends time-series. This information tracks the popularity of product attributes, such as color, category, and fabric, over time, providing valuable information on consumer interest and demand fluctuations.

Specifically, the VISUELLE dataset provides purchase information from 667K users, containing data on 5,577 products exposed in 100 stores of Nunalie, an Italian fast-fashion company. The dataset contains 5,080 samples for the training set and 497 samples for the testing set, and does not serve as a suitable validation set to evaluate during the training procedure model. Therefore, our model was evaluated during the training on the test set.

Since we aim to put ourselves in a more challenging scenario, we rely only on the image and the release date as multimodal conditioning for each product, skipping the Google Trends and description ground truth available. Specifically, the fact that **MDiFF** is conditioned only with image and release date is a pro, since these data types are extremely easy to find, minimizing the need for annotations by object

operators and much more applicable on a large scale automatically in the fast fashion market.

Paired with every item, the dataset gives the sum of sales in all 100 stores of the specific product in the 12 weeks after the release date. Following the evaluation protocol of [37, 151], we only predict the first six values of the available interval.

Evaluation metrics. The Mean Average Error (MAE) and Weighted Absolute Percentage Error (WAPE), *i.e.*, the two main metrics representing the quality of the forecasting, are used to evaluate **MDiFF**. Formally, they are defined as:

$$\text{MAE} = \frac{\sum_{t=0}^{T} |y_t - \hat{y}_t|}{T} , \tag{4.7}$$

$$\text{WAPE} = \frac{\sum_{t=0}^{T} |y_t - \hat{y}_t|}{\sum_{t=0}^{T} y_t} , \tag{4.8}$$

where y represents the actual values of the time-series, $\hat{y}$ represents the forecasted values, and T represents the total number of observations in the time-series.

Training details. All the code is implemented in PyTorch [66]. For the multimodal score-based diffusion model, we train the network for 500 epochs, with a learning rate of 1×10^{-3}, a weight-decay of 5×10^{-4}, using AdamW [67] as an optimizer, on a NVIDIA RTX 4090. On the other hand, for the MLP networks, a Bayesian algorithm was used to search for the best training hyperparameters of the refinement MLP network.

MDiFF Quantitative Results

This section discusses the quantitative results obtained with **MDiFF**. As we can see from Table 4.1, **MDiFF** outperforms all other state-of-the-art methods without using the information from Google Trends and the textual description of the various samples. This clearly works in our favor. Specifically, using Google Trends could worsen the model's performance in our diffusion-based architecture. We hypothesize that the information from Google searches may be noisy in some cases and that, with certain samples, this worsens the performance. However, with respect to the description of the model's features, the diffusion model probably extracts the

Table 4.1 Quantitative results of **MDiFF** expressed in terms of WAPE and MAE, described in Equation (4.8) and Equation (4.7), respectively. In **bold**, the best results. Underlined, the second best.

Method	IMAGE	RELEASE	DESCR.	GOOGLE T.	WAPE ↓	MAE ↓
Attribute k-NN [157]			✓		59.8	32.7
Image k-NN [157]	✓				62.2	34.0
Attr + Image k-NN [157]	✓		✓		61.3	33.5
GBoosting [161]	✓	✓			64.1	35.0
GBoosting+G [161]	✓	✓		✓	63.5	34.7
Cat-MM-RNN [157]		✓	✓	✓	63.3	34.0
X-Att-RNN [157]	✓	✓	✓		59.5	32.3
GTM-Transformer [151]	✓	✓	✓	✓	<u>55.2</u>	<u>30.2</u>
MDiFF (ours)	✓	✓			**54.7**	**30.1**

information directly from the image features more effectively than obtaining it from the description. For more information on that, see Section 4.2.5.

In particular, we report comparisons between eight other existing models. [151] is the most similar in terms of performance to **MDiFF**. The most noticeable improvement is in WAPE, which dropped from 55.2 to 54.7, with a slight improvement in MAE as well. These performance enhancements offer significant benefits that might not be immediately apparent. Primarily, as already introduced in Section 4.2, a diffusion-based model is always preferable for this task, since it should better maintain performance with out-of-distribution objects, ensuring greater stability in practice when used in real-world usage contexts. Secondly, given our specific architecture, we only need less information to achieve better results.

MDiFF Qualitative Results

In this section, we explain the role of the MLP in refining the multimodal score-based diffusion model output forecasting sales. Starting with the visualization of a few samples in Figure 4.4, it can be seen that the model itself gives as output a distribution of predictions that follows the ground truth very closely. Therefore, the role of the MLP refinement may seem obsolete, but there are several factors to consider that, instead, make it crucial.

Firstly, the model output consists of N predictions; then it is necessary to use some technique to obtain a single final prediction. Examples might be simply

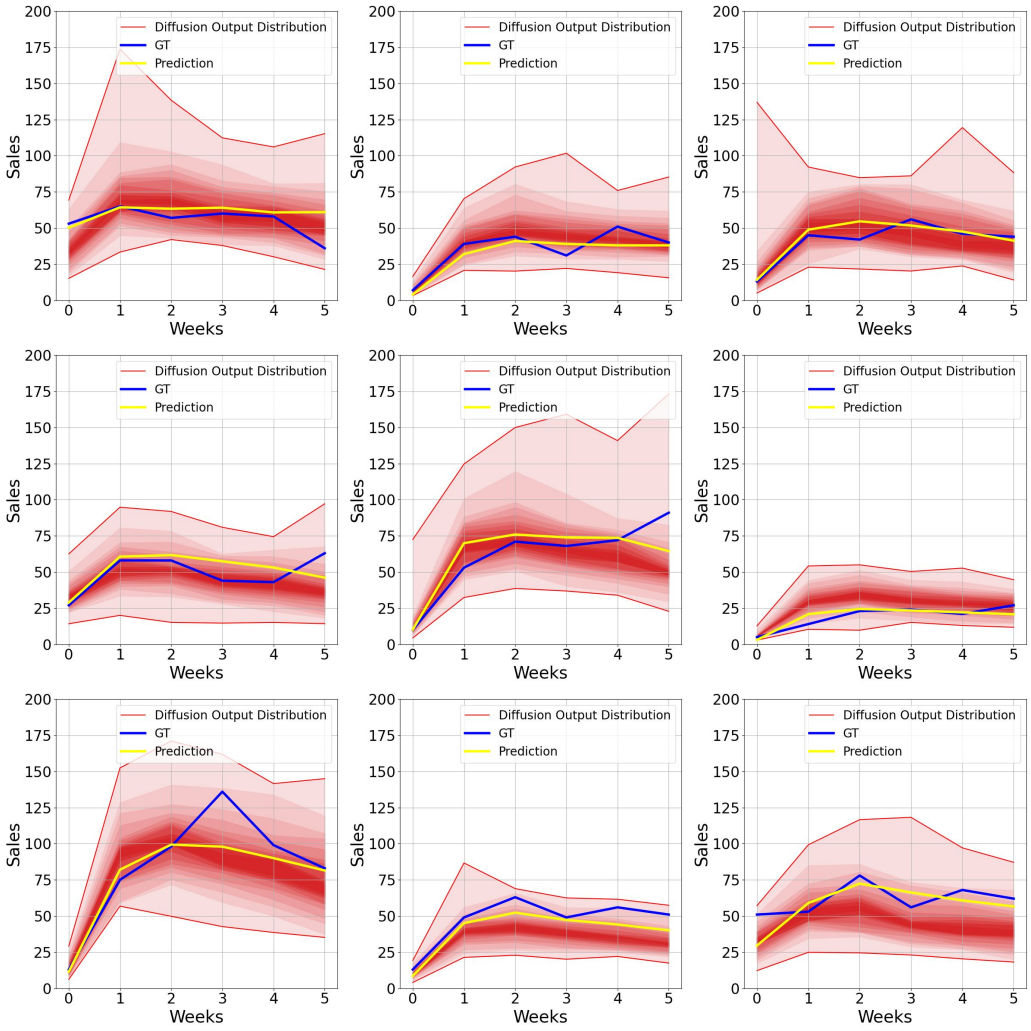


Figure 4.4 In the figures above are presented some visual representations of the multimodal score-based diffusion model output. In particular, the red region represents the output distribution of the diffusion model given a certain sample. The red area is obtained by computing the weekly quantiles among the 50 outputs. The Prediction line, on the other hand, is the output of the refinement MLP, *i.e.*, the final prediction. The forecasting period is for six weeks from the date of release. The y-axis shows the number of units sold of a specific garment in the chain's various shops.

taking the prediction mean or median. However, this would result in performance degradation, as the ground truth often differs from the distribution's mean or median.

As a result, we implemented a lightweight MLP network trained to refine the diffusion model output, as described in Section 4.2.5. As we can see from Figure 4.4, specifically in the second and final image, the ground truth does not reside in the

Table 4.2 Table representing the different tests made with the same multimodal score-based diffusion model. We tested our model first without the temporal condition and then without images.

Model	**WAPE** ↓	**MAE** ↓
MDiFF (ours) without the temporal condition	56.4	31.1
MDiFF (ours) without the images	56.8	31.6
MDiFF (ours)	**54.7**	**30.1**

densest region of the distribution, and the refinement network very effectively follows its movement away from the median of the diffusion output.

In order to understand how much the refinement model actually helps to improve the performance (and not just predict the mean/median) of the diffusion distribution output, we conducted additional ablative studies that show that the use of the refinement module is a winning strategy, improving performance considerably without excessively increasing the model complexity. We did not explore how performance might improve with more complex models, since our simple MLP already yielded good results. We plan to delve into this matter and conduct a more thorough analysis in the future.

Ablation Studies

We conducted ablative studies to test how well the model performs using different conditioning setups. This helps us to understand which configuration best achieves optimal performance with the diffusion model. It should be noted that the error values for each test performed were obtained by running the entire **MDiFF** pipeline and not just the diffusion model. The results are reported in Table 4.2.

Looking at the results, it is clear that conditioning the model with just the images is insufficient. This is because the features of the dress without information on the season and period in which it is sold are insufficient to predict an accurate sales value. Indeed, it is not difficult to think that, since the fashion market is a sector strongly influenced by trends, a certain garment may be very fashionable in one season but remain completely unsold in the next.

However, it is quite straightforward to understand why just the temporal information without any further detail on the item's color, fabric, or shape is insufficient to determine an accurate prediction of the sales.

It is important to note that for other models such as [151], the importance of the various multimodal data types may differ. In **MDiFF**, unlike [151], conditioning is not processed directly to obtain a prediction, but is only used to guide the reverse diffusion process. The impact that one type of conditioning can have on different architectures is, therefore, very different from another.

4.3 Dif4FF

Section Organization

This section first discusses how we guide the generation process using multimodal information (Section 4.3.1). Then, we explore the proposed neural network-based graph refinement (Section 4.3.2) and show the experimental results (Section 4.3.3).

To review the theoretical background of diffusion-based generation models, look at the equations explained in Section 3.3.1. Instead, for a review of the theoretical background of multimodal score-based diffusion models, look at Section 4.2.2.

4.3.1 Our Improved Multimodal Conditioning

We used a combination of data sources to predict future sales for each product. These data will include images of the products (represented as i_j), the Google Trend signal g_j, and their release dates t_j.

Specifically, Figure 4.5 shows the architecture of our multimodal score-based diffusion model. The network is a stack of multiple N blocks. Every block has two outputs, one for the next block and one for a skip connection. The sum of all the skip connections represents the actual output of the model. Every block is mainly made up of an S4 block [22]. With respect to MDiFF [14], our multimodal conditioning also includes Google Trend signals, so it is different in its composition.

We train three different encoders I_θ, T_θ and G_θ to extract features from input information. Then, these features are used to produce the conditional embedding

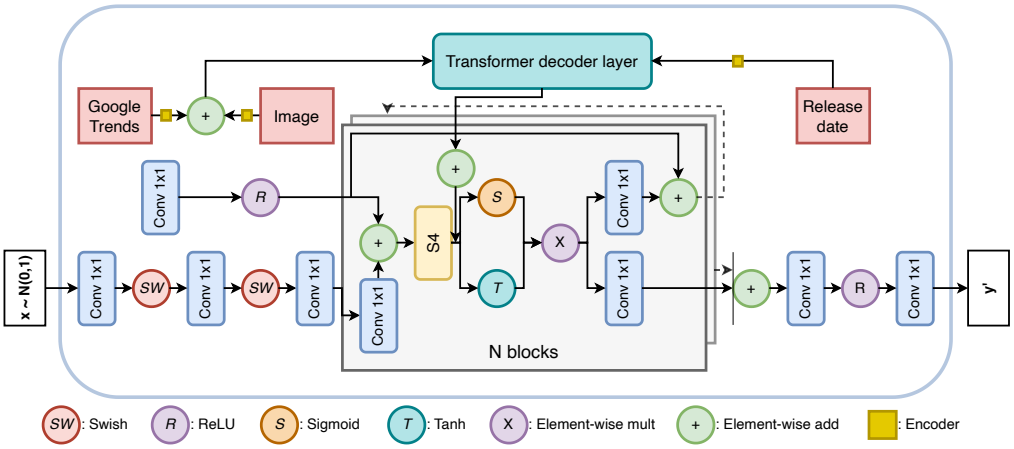


Figure 4.5 An overview of our improved multimodal score-based diffusion model. Each block contains two outputs: one for the subsequent block and another for a skip connection. The summation of all skip connections forms the model's final output. The primary component of each block is typically an S4 block [22]. With respect to MDiFF [14], our multimodal conditioning also includes Google Trend signals, so it is different in its composition.

through a cross-attention mechanism defined as:

$$c_j = \text{Softmax}\left(\frac{Q_j K_j^T}{\sqrt{d_k}}\right) V_j \,, \tag{4.9}$$

where $K_j = V_j = G_\theta(g_j) + T_\theta(i_j)$, $Q_j = I_\theta(i_j)$ and $\sqrt{d_k}$ is the dimensionality of the number of features of the embeddings.

The idea behind this choice is to start from the Google trend signal and use the release date embedding as positional encoding. Lastly, the Google Trend signal is weighted on the visual features (*i.e.*, shape, color, fabric, etc.) extracted from the image. The *multimodal conditioning* c_j is added directly to the output of the S4 block, and the cross-attention is implemented using a Transformer decoder layer [160].

4.3.2 GCN-Based Diffusion Outputs Refinement

The refinement module comprises two main modules: the first is based on ST-GCN [162]. The GCN block operates on two different dimensions by constructing two graphs.

The first graph operates on the dimension of the space of the predictions made by the diffusion model. The mathematical formulation is the following:

$$X_s = \phi_s(A_s X) \,, \tag{4.10}$$

where $A_s \in \mathbb{R}^{S\times S}$ is the adjacency matrix, X output of the diffusion model and ϕ_s an MLP.

The second block works on the time dimension, learning a graph that weighs the connections between the various prediction weeks. Formally, it is defined as:

$$X_t = \phi_t(A_t X_s) \,, \tag{4.11}$$

where $A_t \in \mathbb{R}^{W\times W}$ is the adjacency matrix, X_t output of the first GCN block and ϕ_t an MLP.

These blocks are designed to compress the N predicted samples and regress the final prediction. The network is then trained with a Mean Squared Error (MAE) loss function denoted as $\mathscr{L}_{\text{MAE}}$, mathematically defined as:

$$\mathscr{L}_{\text{MAE}}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|;, \tag{4.12}$$

where $\hat{y}$ represent the ground truth value and $\hat{y}_i$ is the value predicted by the model.

4.3.3 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results. See Section 4.2.5 for dataset and evaluation metric details.

Implementation details. Our model is based on TS-Diff [21]. This architecture is a score-based diffusion model that is particularly useful for predicting, reconstructing, and refining time-series tasks. TS-Diff uses a series of building blocks called “S4 Blocks” [22], stacked together four times with connections that allow information to flow directly (skip connections).

We have made some modifications to TS-Diff to incorporate the additional information from the images and the release dates. As described in the subsection on multimodal conditioning (Section 4.3.2), we have added a layer inspired by Transformers [160]. This layer helps to combine the features extracted from the images i_j, the release dates t_j, and the Google Trends signals g_j to properly guide the diffusion model.

Image encoder. We used as image ecoder I_θ a ResNet-18 [56] pre-trained on ImageNet-1K [73]. We substituted the last two layers of the model with a Conv1D and a Linear to reduce the dimensionality of the features extracted, obtaining a tensor $I_\theta(i_j) \in \mathbb{R}^{C\times W}$, with C channels equal to 64 and W forecasting horizon of six weeks.

Temporal encoder. The temporal encoder T_θ comprises four different MLPs that expand the dimension from 1 to C. The output of this model is then concatenated along the channel dimension and fed into another MLP that reduces the number of features from $4C$ to C, resulting in $T_\theta(t_j) \in \mathbb{R}^C$.

Google Trends encoder. Lastly, as G_θ, we adopted a Transformer encoder layer [160] that performs a self-attention operation on Google Trends. The encoder layer also reduces the dimensionality of the encoding, resulting in $G_\theta(g_j) \in \mathbb{R}^{C\times W}$.

GCN-based refinement. As described in Section 4.3.3, the architecture used for the refinement network is based on ST-GCN [162]. The first GCN-based module comprises two ST-GCN blocks, with an expansion and subsequent reduction of the channels to create a first embedding. On the other hand, the second module is composed of 1D convolutions, reducing the sample dimension from 50 to 1 (to obtain an actual final prediction). Specifically, this compression is done by three layers of 1D convolutions with Parametric ReLU (PReLUs) as activation functions.

Training details. All the code is implemented in PyTorch [66]. For the multimodal Score-based diffusion model, we train the network for 500 epochs, with a learning rate of 1×10^{-3}, a weight-decay of 5×10^{-4}, using AdamW [67] as an optimizer, on a NVIDIA RTX 4090.

Given the nature of diffusion models, different seeds produce different predictions. As a result, we ensured that our score-based diffusion model was executed in the most deterministic setup possible by setting the seed value to 32 across all libraries

Table 4.3 Quantitative results of **Dif4FF** expressed in terms of WAPE and MAE on VISUELLE, described in Equation (4.8) and Equation (4.7), respectively. In **bold**, the best results. Underlined, the second best.

Method	IMAGE	RELEASE	DESCR.	GOOGLE T.	WAPE ↓	MAE ↓
Mean predictor					60.1	32.8
Median predictor					50.3	31.8
Attribute k-NN [157]			✓		59.8	32.7
Image k-NN [157]	✓				62.2	34.0
Attr+Image k-NN [157]	✓		✓		61.3	33.5
GBoosting [161]	✓	✓			64.1	35.0
GBoosting+G [161]	✓	✓		✓	63.5	34.7
Cat-MM-RNN [157]		✓	✓	✓	63.3	34.4
X-Att-RNN [157]	✓	✓	✓		59.5	32.3
GTM-Transformer [151]	✓	✓	✓	✓	55.2	30.2
MDiFF [14]	✓	✓			<u>54.7</u>	<u>30.1</u>
Dif4FF (ours)	✓	✓		✓	**54.6**	**30.0**

used. A Bayesian algorithm was used for the GCN networks to search for the best training hyperparameters.

Dif4FF Quantitative Results

Here, we discuss the quantitative results obtained with **Dif4FF**. As we can see from Table 4.3, **Dif4FF** outperforms all other state-of-the-art methods. To verify the statistical significance of the results, we ran 10 instances of the **Dif4FF** pipeline. The mean MAE was 30.2 with a variance of 0.04, and the mean WAPE was 54.8 with a variance of 0.08.

In **Dif4FF**, we do not use the information from the textual description of the various samples since these could worsen the model performance in our diffusion-based architecture. Furthermore, the diffusion model probably retrieves the color, fabric, and color information directly from the image features. Furthermore, an objective product description is notoriously difficult to obtain in a real-world scenario since the description of fashion garments involves a complex interplay of abstract concepts and stylistic elements, making objective descriptions for each garment in the dataset challenging [163]. For more information on that, see Section 4.3.3.

In particular, we report comparisons between eight other existing models. [151] is the most similar in terms of performance to **Dif4FF**. The most noticeable improve-

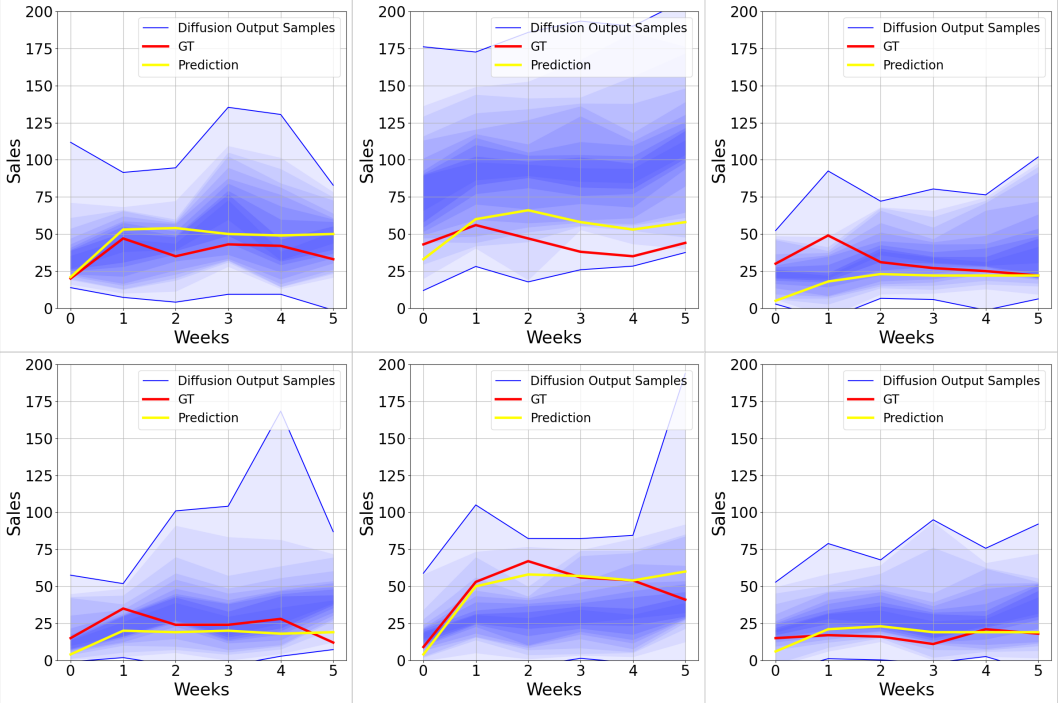


Figure 4.6 In the figures above are presented some visual representations of the multimodal score-based diffusion model outputs. In particular, the blue region represents the output distribution of the diffusion model given a certain sample. Specifically, the blue area is obtained by computing the weekly quantiles among the 50 outputs. The Prediction line, on the other hand, is the output of the refinement GCN, *i.e.*, the final prediction. The forecasting period is six weeks from the release date, depicted on the x-axis. On the y-axis, the number of units sold of a specific garment in the various shops is shown.

ment can be seen in WAPE, which has a sharp drop from 55.2 to 54.7, with a slight improvement in MAE as well. These performance enhancements offer significant benefits that might not be immediately apparent. Primarily, as already introduced in Section 4.3, a diffusion-based model is always preferable for this task, since it should better maintain performance with out-of-distribution objects, ensuring greater stability in practice when used in real-world contexts. Secondly, given our specific architecture, we need less information to achieve better results with respect to the current state-of-the-art.

Dif4FF Qualitative Results

In this section, we explain the role of the GCN in refining the multimodal score-based diffusion model output forecasting sales. Starting with the visualization of a few

Table 4.4 Table representing the different tests made with the same multimodal score-based diffusion model. We tested our model first without the temporal condition and then without images.

Model	WAPE ↓	MAE ↓
Dif4FF (ours) without the temporal condition	56.6	31.5
Dif4FF (ours) without the images	56.2	31.4
Dif4FF (ours) without the Google Trends	55.6	30.9
Dif4FF (ours)	**54.6**	**30.0**

samples in Figure 4.6, even in this case, the role of the GCN refinement may also seem obsolete, but that is not the case.

As we said in Section 4.2.5, since the model output consists of N predictions, it is necessary to use some technique to obtain a final single prediction. Examples such as taking the prediction mean or median would result in performance degradation, as the ground truth often differs from the distribution's mean or median.

As a result, we implemented a powerful GCN network trained to refine the diffusion model output, as described in Section 4.3.3. This allowed us to process the expanded multimodal input more effectively with respect than using a simple MLP. As we can see from Figure 4.6, specifically in the second and fifth images, the ground truth does not reside in the densest region of the distribution, and the refinement network very effectively follows its movement away from the median of the diffusion model outputs.

In order to understand how much the refinement model actually helps to improve the performance (and not just predict the mean/median) of the diffusion distribution output, the errors in terms of MAE and WAPE of both static measurements were calculated. Using the median to extract a final prediction, we obtained a result of 34.8 of MAE and 59.3 of WAPE. On the other hand, for the mean, an MAE error of 34.5 and WAPE of 58.7 were obtained. Thus, adopting a refinement network is a winning strategy that significantly improves performance without excessively increasing the complexity of the model.

Table 4.5 Table representing the results obtained in the domain-shift example. It is clear that our diffusion model is more resilient to the domain shift due to the different years it has been used compared to the second-best performing method.

Model	**WAPE** ↓	**MAE** ↓
GTM-Transformer [151]	56.9	32.8
Dif4FF (ours)	**55.9**	**31.4**

Ablation Studies

The first ablation study tests the model performance using different conditioning setups. It should be noted that the error values for each test were obtained by running the entire **Dif4FF** pipeline. As we can see from Table 4.4 it is clear that images and temporal information are crucial for the model to predict sales accurately. This is because the item's features without information on the season and the release period are insufficient to predict an accurate sales value. Since the fashion market is a sector strongly influenced by trends, a certain garment may be very fashionable in one season but remain completely unsold in the next. However, it is quite straightforward to understand why temporal information and Google Trends, without any further details on the item's color, fabric, or shape, are insufficient to determine an accurate prediction of sales. Google Trends represents information on the public appreciation of a certain item, and the impact on performance is lower when removed.

The second ablation study is related to the domain shift, a well-known issue in the fashion domain. To check the resiliency of the models to this phenomenon, we trained both GTM-Transformer and **Dif4FF**, removing from the train set every garment related to 2018, leaving all of them just in the test set. The VISUELLE test set is already composed of only 2018 garments. As shown in Table 4.5, our method significantly reduces the error when tested on garments completely outside the training distribution. This highlights that while existing methods may perform adequately on training data, their real-world performance suffers significantly under domain change. As a result, **Dif4FF** becomes a killer application for real-world scenarios where domain shift is a common challenge.

Finally, since it is well recognized that a common issue with diffusion models is their tendency to converge on the mean of the data distribution, we investigated the performance of **Dif4FF** compared to a simple predictor that uses only the mean and variance of the training data for the test set. The results are reported in Table 4.3

and reveal that based on the mean and variance, this naive predictor significantly underperforms compared to our **Dif4FF**. This finding reinforces the effectiveness of our proposed approach for the NFPPF task.

Chapter 5

Human-Centered Systems Design

The evolution from Industry 4.0 to Industry 5.0 represents a significant shift in industrial practices. The former focuses on automation, efficiency, and integration of CPSs [24], whereas the latter places humans at the center of these systems [27]. This new paradigm emphasizes human-machine collaboration, where technology improves productivity and ensures adaptability, resilience, and sustainability in industrial operations. The objective is not only to optimize machine performance but also to ensure human operators' safety, well-being, and active participation in these highly automated environments. This evolution presents unique challenges, particularly in integrating the human element into a digital and data-driven industrial ecosystem.

One of the key technologies that allows this integration is the concept of DT, which is a virtual replica of a physical asset [164]. By mirroring the real-time state of its physical counterpart, the DT enables continuous data collection, analysis, and simulation. Typically, this model is used to optimize production performance or predict future behaviors based on current and historical data. The ultimate goal of using DTs is to improve decision-making by providing a deeper insight into the behavior of the system under various conditions, allowing more informed strategies to optimize overall performance.

In Figure 5.1, we present all publications related to human-centered design. In the following section, we outline the relevant literature and the motivation behind each work. Then, we provide a detailed explanation of each work, encompassing both methodologies and the achieved results.

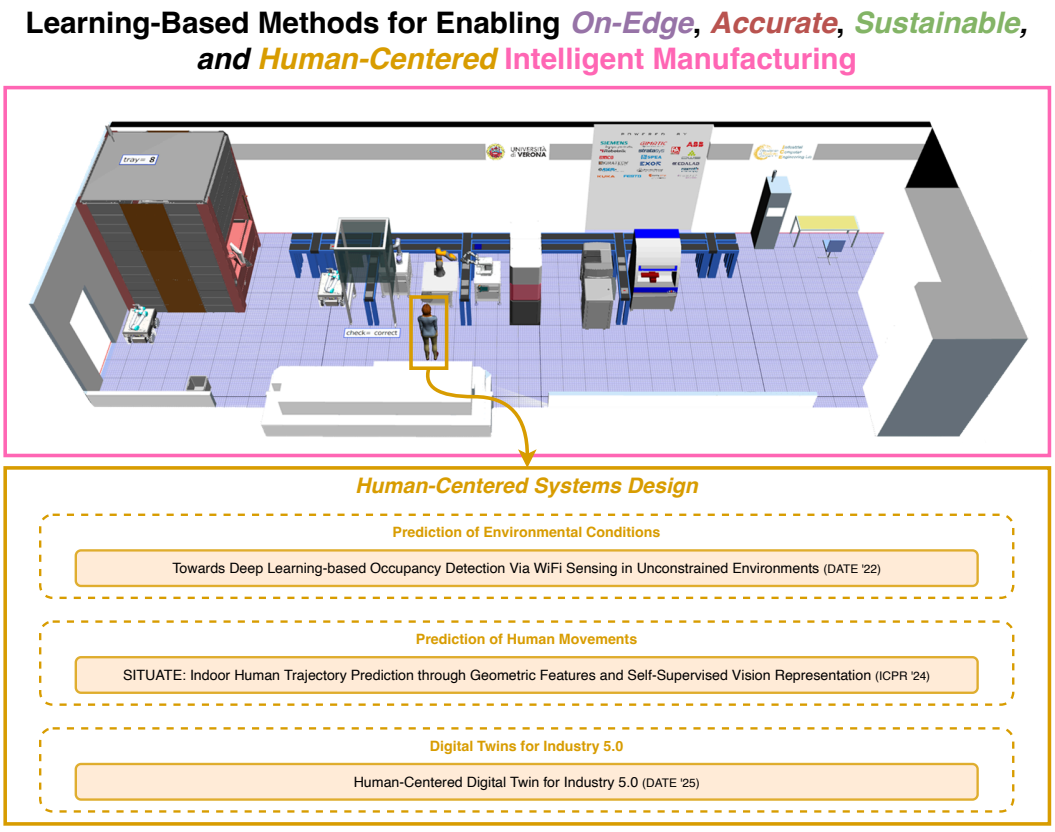


Figure 5.1 Overview of research contributions on human-centered design.

Motivations for WiFi-Based Detection [16] (Section 5.2). In the context of intelligent manufacturing, the design of remote monitoring and control systems is becoming of paramount interest [165]. Using dedicated ML techniques, these systems allow one to observe the status of the environment (*e.g.*, the presence and number of occupants) [166], as well as recognize the human activities carried out in it (*e.g.*, walking, standing, sitting) [167]. This kind of knowledge has practical uses in several contexts. For example, it allows one to automatically turn on/off a lighting and air conditioning system if a room is occupied or is found empty, thus optimizing energy consumption and building security [168]. Considering safety aspects, it is exploited in industrial plants to detect and possibly prevent accidents at work [169, 31]. In particular, current systems use wearables, smartphones, dedicated sensors (*e.g.*, passive infrared sensor sensors), ambient sensors (*e.g.*, temperature, humidity, brightness, pressure, CO_2 level) or video cameras to implement the mentioned recognition scenarios [170]. However, these approaches require special hardware to be installed/worn and, in some cases, have high computational requirements.

Beyond the approaches listed above, WiFi sensing techniques have become a prominent solution, thanks to the ubiquity of WiFi, both indoor and outdoor, and the relatively low deployment costs of such commodity off-the-shell WiFi routers [171]. Although WiFi sensing may seem a panacea for many sensing scenarios, its practical application is not straightforward for real-world cases. In fact, the propagation of the WiFi signal is strongly affected by environmental conditions such as *i)* changes in furniture layout, *ii)* presence of people and their activities, and *iii)* humidity and temperature [172]. These aspects limit the applications of WiFi-based monitoring systems, and in the current state-of-the-art, this is solved by reducing the possibility of environmental variations.

As a result, with **WiFi-Based Detection**, we go beyond the limitations mentioned above for the specific occupancy detection scenario. It presents a deep learning network capable of detecting the presence of people in an indoor environment, regardless of their positions, the activities they perform, or the humidity level and temperature. The network is lightweight, permitting the deployment on resource-constrained devices (*e.g.*, Nucleo-L432KC) for real-time utilization.

Motivations for SITUATE [17] (Section 5.3). Despite the significant volume of research over the past decade devoted to outdoor trajectory prediction [173–179], there has been a notable scarcity of studies that exploited user trajectory data in indoor settings [180–186], also considering the crucial role these predictions play nowadays in the development of location-based services within indoor spaces. In indoor spaces, users can navigate through different interconnected rooms, corridors, doors, and elevators, often having the freedom to deviate from straightforward paths and choose alternative routes. Indoor spaces also have a higher density of structural elements and potential obstacles, such as furniture, walls, and partitions.

Although outdoor trajectory prediction has received significant attention, indoor prediction is still an underexplored research area. As a result, we present **SITUATE**, the first model designed specifically to cope with indoor trajectory forecasting by using equivariant and invariant geometric feature learning and self-supervised vision representation.

Motivations for HC-DT [18] (Section 5.4). While extensive research exists on creating DTs of machines and factories [187], incorporating a human replica into

these digital worlds remains a challenge. A potential solution to this problem is the concept of *Industrial Metaverse*, *i.e.*, a digital space where human-machine interactions occur in real time [188]. However, Industrial Metaverse is still a new concept in the context of Industry 5.0 [189]. Most current developments focus on machines and automation, ignoring how humans fit into these systems. As a result, there is no clear methodology yet for developing human-centered DTs within the Industrial Metaverse.

As a result, in **HC-DT**, we propose a pipeline that utilizes Unreal Engine's MetaHuman technology [190] and AI to create a human-centered DTs for real-time monitoring and decision-making in industry 5.0 environments. This solution aims to improve safety, adaptability, and performance in Industry 5.0 scenarios by connecting the physical human with its virtual counterpart. In this work, we focus on the wakefulness and drowsiness detection task, and to the best of our knowledge, this is the first real proposal that shows how to create and integrate humans DTs in the context of Industry 5.0, addressing the critical gap in human-machine collaboration. We also demonstrate how human-centered DTs created through our pipeline can be seamlessly integrated into the DTs of a real factory: the ICE Laboratory at the University of Verona.

5.1 Related Works

This section explains the evolving role of DTs from Industry 4.0 to 5.0, highlighting the growing importance of human-centered systems. Then, we introduce the concept of the Industrial Metaverse and examine state-of-the-art simulation tools supporting this emerging paradigm.

The Role of Digital Twins in Industry 4.0 & Industry 5.0

The transition from Industry 4.0 to 5.0 represents a shift from machine-centered automation to human-centered systems [36, 35]. DTs has become a key technology for creating virtual replicas of physical assets to monitor, simulate, and predict machine behavior in real time [164]. Machine-centered DTs models have proven to be valuable for predictive maintenance, reducing downtime, and optimizing industrial operations. Industry 5.0 instead aims to reintroduce humans back into the loop, prior-

itizing adaptability, resilience, and sustainability [191]. This shift requires integrating human elements into DT frameworks to enable real-time monitoring and interaction between humans and machines. However, most existing DT implementations remain machine-centered, with human behavior and safety in industrial environments largely unexplored [27].

This thesis distinguishes between three key concepts: the *virtual replica*, the *digital shadow*, and the *DT*. The virtual replica is a visual representation of a human, created using detailed models such as MetaHuman. This visual representation of the operator enables realistic interaction within virtual environments. The digital shadow extends this concept by incorporating real-time data, dynamically updating the virtual replica based on the movements, expressions, and states of the operator [192]. Finally, DT builds on the virtual replica and digital shadow by adding predictive capabilities and enabling bidirectional interaction between virtual and physical counterparts.

A wide range of definitions for DT exist in various domains [193]. For this work, we adopt the definition provided by the National Academies of Sciences, Engineering, and Medicine, which states: "A digital twin is a set of virtual information constructs that mimics the structure, context, and behavior of a natural, engineered, or social system, is dynamically updated with data from its physical twin, has predictive capacity, and informs decisions that realize value. The bidirectional interaction between virtual and physical is central to the digital twin" [194]. This definition highlights the importance of dynamic, real-time data exchange between physical and virtual entities. This exchange enables DT to generate actionable insights and support more informed decision-making.

AI, especially deep learning models, play a crucial role in this transition by enabling real-time decision-making and predictive analysis in Industry 5.0 environments. AI's ability to analyze large volumes of data and identify patterns can significantly improve both machine and human safety, particularly in scenarios that require immediate response. Although advanced models have been successfully applied to tasks such as object detection and activity monitoring, their application in human-centered industrial contexts remains in its infancy [195].

Integrating DTs and AI in Industry 5.0 goes beyond simple machine optimization. It enables the development of dynamic and adaptive systems capable of predicting and reacting to machine failures and monitoring human conditions, enhancing safety and performance through real-time insights [196]. This advancement addresses the

challenge of integrating humans and machines in industrial settings. This thesis explores this issue and introduces the first human-centered DT solution, incorporating advanced deep learning models to facilitate machine integration within an industrial scenario.

The Industrial Metaverse and Simulation Tools

Industrial metaverse. The concept of the Industrial Metaverse is still in its early stages, especially within the context of Industry 5.0 [188]. As industries strive to integrate humans more deeply into smart systems, Metaverse offers a promising platform to create immersive digital environments where humans, machines, and DTs can interact in real-time. The platform enables simulations beyond machine-focused processes, incorporating human factors such as safety, ergonomics, and decision-making into complex, often hazardous, industrial environments [197].

Simulation tools. State-of-the-art simulation tools are critical for building the Industrial Metaverse. Widely adopted tools such as Microsoft's Azure DT [198] and Siemens' Plant Simulation [199] excel in creating highly detailed and machine-centric DTs for real-time monitoring in Industry 4.0. However, these frameworks often overlook the human element, which is instead central to Industry 5.0.

NVIDIA Omniverse [200] represents a significant advancement, offering a collaborative platform for building DTs that incorporates human elements into industrial simulations. Unlike traditional tools, Omniverse allows the integration of human factors within the DT during simulations. However, it functions primarily as a simulation environment rather than a real-time DT solution capable of continuous monitoring. Once DT is deployed, it lacks the capability for continuous human operator tracking or real-time decision-making.

In contrast, game engines such as Unreal Engine [190], traditionally used in entertainment, are being repurposed for industrial applications due to their high-fidelity graphics and real-time simulation capabilities. Unreal Engine's MetaHuman technology allows the creation of lifelike digital human models that serve as virtual representations of workers. This offers unprecedented opportunities to simulate human behavior in hazardous or high-stakes industrial environments.

Real-time synchronization between the physical operator and their virtual counterpart is achieved using the LiveLink plugin [201]. Alternatives like Rokoko Studio [202], Xsens [203], and Perception Neuron Motion Capture [204] require expensive motion capture suits, with starting costs exceeding $2,000, and are not easily interchangeable between operators, making them less practical for industrial use. Given these factors, LiveLink was selected for its affordability, ease of use, and suitability to track facial expressions without additional hardware, supporting our goal of creating a scalable, worker-friendly solution.

Although platforms like Unreal Engine can excel in creating human-centered environments DTs, their application in real-time industrial settings remains a novel approach. In this thesis, we demonstrate how industries can leverage the Unreal Engine's simulation and MetaHuman capabilities to integrate human elements seamlessly into industrial environments, making a critical step toward realizing the vision of Industry 5.0.

5.2 WiFi-Based Detection

This research presents the device setup used to collect Channel State Information (CSI) and environmental data in an unconstrained environment, and the ML model to deal with the occupancy detection task, equipped with a strategy for model interpretability.

Section Organization

In the following, we first describe the data collection setup in detail (Section 5.2.1). Next, we introduce the proposed deep network (Section 5.2.2). Finally, we present the experimental results in Section 5.2.3.

5.2.1 Data Collection Setup

Our data collection setup is shown in Figure 5.2. A single large office is taken into account: it is $12 \times 6 \times 3$ meters large, with three (2×1.8 meters) windows and one entrance door. The internal walls are made of plasterboard (12 centimeters thick), and the external walls are made of reinforced concrete (55 centimeters thick). The

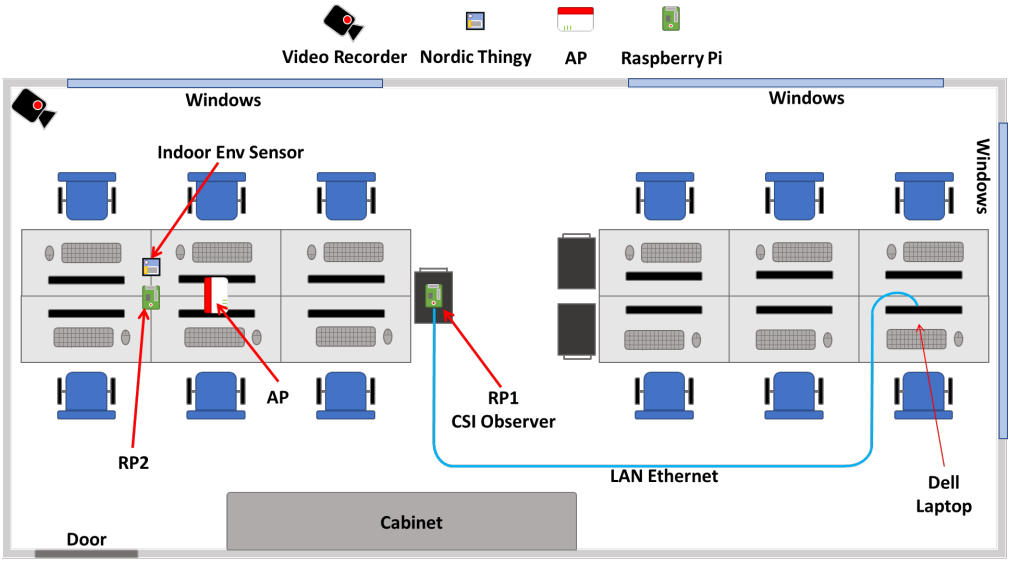


Figure 5.2 Overview of the data collection environment.

Table 5.1 Format of the collected data.

Timestamp	CSI Amplitude $a0$	...	$a63$	Environment Temperature	Humidity	Occupancy status
15:38:45.550	0.027	...	1	21.97	43	1
15:38:45.600	0.027	...	1	21.82	43	1
15:38:45.650	0.027	...	1	21.82	42	1
...	...	...	...	...	...	...
15:38:45.700	0.027	...	1	23.97	36	0

occupants were instructed to carry out their office activities (*e.g.*, walking, standing, sitting, going out, and getting in) without any restrictions.

Four devices are used for data collection: two Raspberry Pi (RP1 and RP2 in Figure 5.2), one Access Point (AP), and one Nordic Thingy 52. The AP exposes a WiFi network (over the 2.4 GHz band) observed by the RP1. Every Raspberry Pi device is patched using the Nexmon framework and extracts CSI amplitude data at 20 Hz. The AP and RP1 are placed 2 meters apart at a height of 140 cm, and the occupants cannot move between them. The RP2 device communicates over Bluetooth with a Nordic Thingy device, which serves as a ground-truth sensor for environmental information (*i.e.*, humidity, and temperature). Both Raspberry's are connected to a Dell Inspiron 7559 laptop that synchronizes them and stores the CSI amplitude together with humidity and temperature data. An external observer

manually annotated the presence of humans[1] based on recorded video data. A semi-automatic annotation tool considerably simplified the process by avoiding the need to explicitly annotate every single timestamp. An overview of the final dataset format is shown in Table 5.1. At each timestamp, we have the CSI amplitude of the 64 subcarriers (*i.e.*, $a0$ to $a63$), temperature ($^{\circ}C$), humidity (%), and the occupancy status.

5.2.2 The Proposed Deep Network

Given a subcarrier x of a CSI signal, we refer to the time-series of CSI amplitudes as $S(x,t)$, where t refers to the t-th time of the day and $x \in \{0,\ldots,63\}$. The time-series of humidity h and temperature e are defined as $S(h,t)$ and $S(e,t)$, where $h,e \in R$ represent the values of humidity and temperature at time t.

The ultimate goal of our approach is to take advantage of the expressiveness of the CSI signal for the occupancy detection task. At the same time, we want a technique that is interpretable, which means that its decisions can be motivated in simple terms, even by a non-expert. Finally, we are interested in presenting a lightweight system that will pave the way for real-time applications.

For these reasons, we are proposing here a memory- and computation-efficient deep learning-based solution. In detail, we implement a lightweight MLP composed of four layers, each except the last activated by the rectified linear activation unit (ReLU) function. Specifically, the first layer has 8.320 neurons, the second 33.024, the third 32.846, and the latest 129 neurons, for a total of 77.881 trainable parameters, with a forward/backward pass size of 0.01 MB. These size parameters have been chosen considering similar problems we encountered previously, with special care in keeping the number of parameters bounded. The MLP network is trained via adaptive mini-batch gradient descent, with a weight decay strategy [205], in order to prevent the network from overfitting the training data as well as the exploding gradient problem.

Since we cast an occupancy binary problem, given the input feature set $F = S(x,t) \cup S(e,t) \cup S(h,t)$ at timestep t, the model has to predict a binary label $p_t \in \{0,1\}$ corresponding to an empty and non-empty (*i.e.*, occupied) office, respectively. The discriminant function for such a task can be learned by minimizing Binary

[1]Label 0 if the environment is empty, and label 1 if there is at least one person in the environment.

Cross-Entropy (BCE), defined as follows:

$$\mathscr{L}_{\text{BCE}}(y,p) = -\frac{1}{T}\sum_{t=1}^{T} y_t \cdot \log(p_t) + (1-y_t)\cdot \log(1-p_t)\,. \tag{5.1}$$

Here, y_t is the target value (either 0 or 1). The prediction p_t can be any value between 0 and 1, indicating that the confidence in that sample is positive. Predictions are often normalized to match a probability via the 1-bounded logistic function, often known as the sigmoid function.

This model is capable of learning complex, non-linear relationships, it comes with powerful interpretability tools, and is very fast, as we will see later. Importantly, the model can be modified to feed a subset of signals, for example, the CSI signal $S(x,t)$ only. This will allow us to perform comparative experiments and compare the relative importance of the different features.

Interpretability of the model. As an additional contribution of our work, we propose the use of an Explainable AI (XAI) technique named Grad-CAM [41], which relies on the gradients of the loss function to compute importance weights associated with the input features (*i.e.*, CSI subcarriers, humidity, or temperature). This allows us to understand what input features the network pays the most attention to when making a decision for a specific class.

Intuitively, this method is a post hoc attention mechanism since it works on the already trained neural network. In particular, the Grad-CAM approach has been shown to pass the "sanity check" for saliency-based interpretability approaches [42], which means that it is guaranteed to express the relationships between input features and outputs present in the data on the specific deep learning model that has been adopted, in our case the MLP.

Grad-CAM essentially computes the importance weights as the average of the gradients flowing down from a class label throughout the different layers until the input features. To this end, we start by computing the hidden importance coefficient of Grad-CAM α_k^c as follows:

$$\alpha_k^c = \frac{1}{N}\sum_d \frac{\partial y^c}{\partial A_d^{(k)}}\,, \tag{5.2}$$

Table 5.2 Distribution of simultaneous presence of subjects in terms of data samples.

Occupancy	Empty = 0	Occupied = 1			
Occupants	Zero	One	Two	Three	Four
# Samples	3389840	986180	569480	332440	84400
(%)	(63.2%)	(18.4%)	(10.6%)	(6.2%)	(1.6%)
5362340	3389840 (63.2%)	1972500 (36.8%)			

where $A_d^{(k)}$ is the hidden feature map at layer k (*i.e.*, the output of the neurons of the layer k), d represents the number of hidden neurons at the layer k and c the class. Specifically, the weight α_k^c represents the partial linearization of the deep network downstream of A and captures the value of the feature map of the k-th layer for a target class c. Finally, we perform a weighted sum between the value just calculated and the feature maps $A_d^{(k)}$, followed by a ReLU to reset the negative values of the gradient to zero:

$$L_{GradCAM}^c = ReLU(\sum_k \alpha_k^c A_d^{(k)}) \, . \tag{5.3}$$

The last operation is adopted for numerical stability.

Dimensionality and speed of the model. Our network is designed using the PyTorch Lightning framework, resulting in a model size of 15.18 KiB, with a RAM occupancy of 23.04 KiB that is easily deployable over a resource-constraint device such as Nucleo-L432KC. Once trained, the model is capable of an inference time of 10.781 ms for each sample with the full amount of input features (CSI, temperature, humidity).

5.2.3 Experimental Results

This section describes the experimental trials that have been carried out to validate our claims, along with their implementation details and results.

Data Profiling

From January 04, 2022, at 15:08:40 h to January 07, 2022, at 17:38:40 h, for a total of 74 hours (*i.e.*, 268117 seconds with 32174040×66 samples), we collected a 10 GB dataset whose format is shown in Table 5.1. As features, here we focus on timestamp,

CSI amplitude (*i.e.*, 64 subcarriers), humidity, and temperature measurements. In terms of occupancy detection, six subjects (two women and four men) entered and used the office. They were informed of the data monitoring system and told to carry out their office activities as usual. In particular, the subjects worked freely in the room, moving chairs, raising/lowering curtains, and moving without a predefined pattern. Table 5.2 presents the distribution of the presence (simultaneously) of subjects in the environment. Overall, 63.2% of the dataset represents the empty environment; the remaining 36.8% represents the environment with at least one subject in it.

The data collected undergoes a time-series analysis composed of the following steps. Initially, we control for null values or duplicates present in the same t. Afterward, we analyze the data distribution of $S(x,t)$, $S(h,t)$, and $S(e,t)$ both visually and numerically. As a second step, we test the stationarity of the time-series using a statistical unit root test, namely the Augmented Dickey-Fuller (ADF) test [128]. The result shows that all time-series treated in this problem are stationary, so the correlation analysis can be performed on the raw data, without any other preprocessing (*e.g.*, detrendization) [206]. To do this, we rely on Pearson's ρ coefficient, defined as:

$$\rho = \frac{cov(X,Y)}{\sigma_x \sigma_y}\,, \tag{5.4}$$

where, given a pair of random variables (X,Y), cov is the covariance, σ_x is the standard deviation of X, and σ_y is the standard deviation of Y. In particular, we report that temperature and humidity have a positive correlation of 0.45. The temperature has a correlation of 0.44 with respect to the binary occupancy variable, while the humidity is correlated at 0.35. In terms of subcarriers, they are mostly correlated with neighboring subcarriers, and mid- to high-band carriers (*i.e.*, $a15$ to $a28$ and $a48$ to $a64$) are somewhat correlated with temperature and humidity ($\sim$ 0.20 to 0.30). An obvious point is that time as a feature is strongly correlated (*i.e.*, 0.77) with environmental data. This is because temperature and humidity strictly depend on the heating system and the presence of humans. The former, since the office presents a heating system that automatically activates and deactivates, and the latter, since humans modify the environment (*e.g.*, modify the setting of the heating system, open/close windows or doors, and there is also the effect of body temperature and breathing).

Table 5.3 Start time, end time, number of samples, min/max temperature, and humidity for the training (0) and testing (1 to 5) folds.

Fold	Start	End	Empty	Occupied	T	H
0	04/01 15:08	06/01 19:16	2348151	1405500	18.72/40.09	16/49
1	06/01 19:16	06/01 23:44	321742	0	20.36/23.90	20/45
2	06/01 23:44	07/01 04:12	321742	0	18.86/21.80	25/42
3	07/01 04:12	07/01 08:41	321742	0	18.68/20.80	25/43
4	07/01 08:41	07/01 13:09	56223	265519	18.38/22.10	22/43
5	07/01 13:09	07/01 19:16	0	321741	20.19/31.60	20/38

Table 5.4 Occupancy detection accuracy (in %) over the 5 testing folds comparing three different ML models.

	Logistic Regressor			Random Forest			MLP		
Fold	CSI	Env	C+E	CSI	Env	C+E	CSI	Env	C+E
1	68	99	76	99	100	99	100	99	92
2	71	100	72	100	100	100	100	100	99
3	77	100	86	99	100	100	100	100	100
4	94	18	86	88	75	88	83	54	65
5	96	31	91	100	100	100	100	99	99
Avg.	**81**	**70**	**82**	**97**	**95**	**97**	**97**	**90**	**91**

The Occupancy Detection Task

To evaluate the performance of our model, the dataset is divided into train and test sets, taking particular care in the division of the test set. In temporal order, the train set represents 70% of the collected data, and the test set the remaining 30%. The test set is further divided into five folds, representing different scenarios over time. This allows us to evaluate the model for general performance and test generalization abilities on unseen days separately. This is to stress the fact that when evaluating over different folds, the train set never changes, and the models are never re-trained. Table 5.3 presents the train/test split of the collected data.

Our model is trained for 10 epochs with a learning rate of $5e^{-3}$ on an NVIDIA GeForce RTX 3090 GPU. In addition to our model, we used two well-known ML models, Logistic Regression and Random Forest (RF), implemented using the Scikit-learn framework as a baseline to compare with the results of the MLP model. The results of the occupancy detection task are reported in Table 5.4. The tested models are trained on three different subsets of the collected dataset: *i)* only CSI data,

ii) only environment data (Env) (*i.e.*, humidity and temperature), and *iii)* CSI and environment data (C + E). In particular, the Logistic Regressor is a linear classifier whose results demonstrate that it is not easy to describe the intricate relationships of data in a linear manner.

Instead, the RF is a non-linear ensemble model based on decision trees, famous for its ability to resist overfitting, which achieves excellent performance. The proposed MLP model also achieves remarkable performance. These results show how the non-linear models (RF and MLP) are capable of properly using the CSI data, thereby reaching higher classification scores. However, we specify that the use of the MLP model is preferable (compared to RF) because:

- RF is computationally and space-intensive (*i.e.*, does not allow real-time operation and deployment on embedded boards).
- An MLP model can be trained continuously. There is no need to use the whole dataset again, and only new data can arrive in real-time, thus doing online training.
- An MLP can be easily extended by someone who wants to work on this problem.

In turn, this shows that using only environmental data is insufficient to optimally solve occupancy detection, while the CSI data contain sufficient information to agglomerate the possible impact of temperature and humidity. In addition, we specify that if we use only time as a feature for our analysis, the performance in terms of accuracy does not present good results (*i.e.*, 89.3%) compared with those of the MLP in Table 5.4.

Explain Model Outputs Using Grad-CAM

The plot in Figure 5.3 shows the importance of each feature (*i.e.*, CSI subcarriers on the yellow background, environment on the red) given to the MLP classifier for the prediction of occupancy, as explained in Section 5.2.2.

Interestingly, we found that temperature and humidity are not important (values close to 0, if not negative in the plot), while the highest importance values are located between low frequencies (subcarriers $a9$ to $a17$) and high frequencies (subcarriers

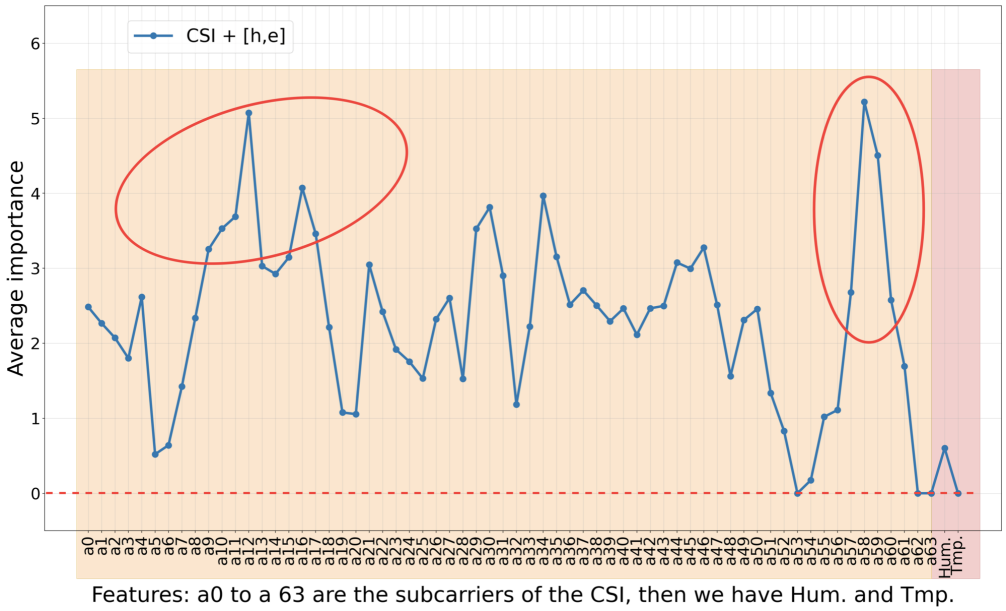


Figure 5.3 Results of the XAI method, depicting the importance of all the features using CSI, humidity (h), and temperature (e).

*a*57 to *a*60), confirming that paying attention to humidity and temperature does not help the task while paying attention to CSI does.

Humidity and Temperature Prediction

Inspired by the previous results, we performed multiple regression analyzes to see if we can estimate humidity and temperature from the CSI data. To do this, we fit a least-squares solution, both using linear regression (ordinary least squares) and non-linear regression (minimization of a squared error objective), implemented with our neural network model.

The goal of this analysis is to show that sufficient information regarding environmental factors is contained in the set of features of CSI signals, which would, therefore, allow us to better understand the environment, given only the amplitude values of CSI. We confirm such a result in Table 5.5.

The fact that MLP (which is a non-linear method) performs definitely better than the linear regressor shows that the variation of temperature and humidity inside the room is mostly reflected by CSI data in a non-linear fashion.

Table 5.5 MAE/MAPE results of linear and neural network regression models on the humidity (H) and temperature (T) prediction.

	Linear Regressor		Neural Network	
Fold	MAE (T/H)	MAPE (T/H)	MAE (T/H)	MAPE (T/H)
1	2.72/2.47	12.65/7.11	1.04/3.74	4.18/11.26
2	1.87/1.65	9.24/4.86	0.56/7.30	2.82/21.98
3	3.57/2.84	18.17/8.25	0.73/6.08	3.72/18.55
4	6.04/6.92	29.38/20.51	3.88/3.44	18.59/10.46
5	8.08/7.51	35.94/25.89	3.81/2.55	16.94/9.54
Avg.	**4.46/4.28**	**21.08/13.32**	**2.39/4.62**	**9.25/14.35**

This result also allows an understanding of why the non-linear classification models do not perform better when using the CSI + Env data: the latter represents a redundant feature, and redundancy is widely known to affect any classifier's performance negatively.

5.3 SITUATE

In this section, we present **SITUATE**, our novel approach for indoor human trajectory forecasting based on equivariant and invariant geometric feature learning modules and self-supervised vision representation.

Mathematical background. Given a set of transformations $T_x : X \to X$, a function $F : X \to Y$ is called Equivariant if exists a transformation $T_y : Y \to Y$ equivalent to T_x, on the Euclidean space such that:

$$F(T_x(X)) = T_y(F(X)) \ . \tag{5.5}$$

Moreover, we also want the model to have the invariance property. Given the same set of transformations, a function $F : X \to Y$ is called Invariant on the Euclidean space if it exists a transformation $T_y : Y \to Y$ such that:

$$F(X) = F(T_x(X)) \ . \tag{5.6}$$

Specifically, this work addresses the problem of multi-person trajectory forecasting by considering the input trajectories as a graph. As proven by [207], during the message passing of a Graph Neural Network (GNN), the property of equivariance can be ensured by enriching the features of the neighbor nodes with the distance $L2$ between nodes. Let $G = \{V, E\}$ be an input graph representing the input trajectory with nodes $v_i \in V$ and edges $e_{ij} \in E$. For every node v_i, a feature vector $h \in \mathbf{R}^h$ and an absolute position $x_i \in \mathbf{R}^3$ are given. To preserve equivariance among different layers of the model, we update the position as follows:

$$m_{ij} = \phi_e\left(h_i^l, h_j^l, \left\|x_i^l - x_j^l\right\|^2\right) , \tag{5.7}$$

$$x_i^{l+1} = x_i^l + C\sum_{j\neq i}\left(x_i^l - x_j^l\right)\phi_x\left(m_{ij}\right) , \tag{5.8}$$

where C is equal to $1/(M-1)$ with M number of nodes, ϕ_e and ϕ_x are learnable MLPs, defined as $\phi_e(\cdot) = W_e \cdot + B_e$, l indicates the layer and m_{ij} represents the information passed between two nodes during the message passing.

As reported in [207], ϕ_x has to be a scoring function $\phi_x : X \to S$, with $S \in \mathbf{R}^1$. With this procedure, the update of V is consistent, allowing the model to learn without being affected by SO(2) transformations, with SO(2) being the group of all rotations in the plane around the origin that preserve the Euclidean norm, mathematically described by 2×2 matrices. Furthermore, the features learned across layers must be consistent and invariant to graph transformations. To do so, the following procedure governs the final message-passing operations and the update of the features carried out by the $i-th$ layer:

$$m_i = \sum_{j\neq i} m_{ij} , \tag{5.9}$$

$$h_i^{l+1} = \phi_h\left(h_i^l, m_i\right) , \tag{5.10}$$

with ϕ_h, an MLP also designed as $\phi_h(X) = W_h X + B_h$, responsible for the invariant feature learning. Mixing these two components allows us to build an Equivariant and Invariant GNN using Euclidean SO(2) transformations.

Motion prediction. Here, we introduce the general formulation of the motion prediction problem. We have a multi-agent system with m agents. Each agent is

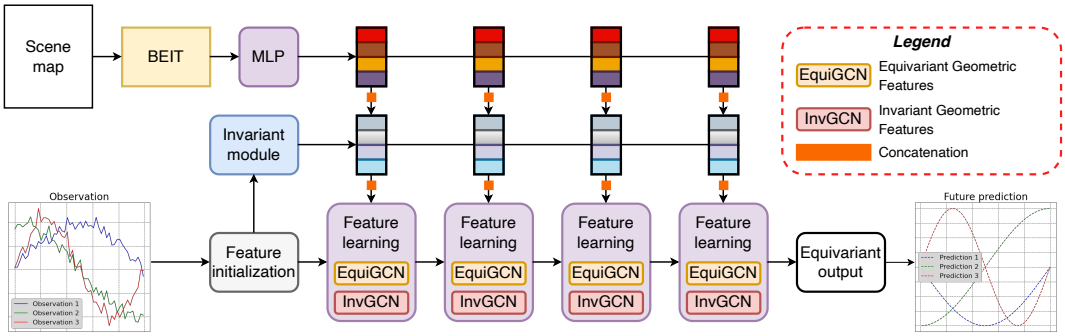


Figure 5.4 In **SITUATE**, we first produce a feature vector regarding the scene using the self-supervised vision representation module. Then, a feature initialization layer is used to initialize geometric and pattern features. Successively, we update the geometric and pattern features by the equivariant geometric feature learning and invariant pattern feature learning layers, obtaining expressive feature representation. We further use an invariant reasoning module to infer an interaction graph used in equivariant geometric feature learning. Finally, we use an equivariant output layer to obtain the final prediction.

represented as A_i, where $i = 1, 2, \ldots, m$. The goal is to predict the future motions of these agents based on their historical observations. For each agent A_i, we can denote historical observations by X_i. These observations typically include positions and can be represented as $X_i = \{x_0^i, x_1^i, \ldots, x_t^i\}$, where x_t^i represents the position of agent A_i at time step t. We also add velocity $S_i = \{s_{t+1}^i, s_{t+2}^i, \ldots, s_{t+f}^i\}$ as input information of the model. The velocity of an agent is a natural invariant feature because it is not affected by any translation or SO(2) transformation. We use the velocity to compute the initial feature vector of a specific agent A_i. More details in Section 5.3.2. Specifically, for each agent A_i, our aim is to predict its future f positions $Y_i = \{y_{t+1}^i, y_{t+2}^i, \ldots, y_{t+f}^i\}$.

Section Organization

In the following, we first introduce the **SITUATE** prediction network (Section 5.3.1). Next, we detail the feature initialization process (Section 5.3.2). Finally, we present the experimental results in Section 5.3.3.

5.3.1 The SITUATE Prediction Network

In this section, we present **SITUATE**, our motion prediction network that explicitly uses equivariant and invariant geometric features and a self-supervised scene representation module to tackle the indoor trajectory prediction problem. The model architecture is shown in Figure 5.4.

The first module we present is in charge of producing the scene-representation encoding. As anticipated, the subjects' motion characteristics differ greatly from those of the outdoor case when considering indoor trajectory forecasting. In fact, motion is strongly characterized and limited by objects and obstacles in the scene. Knowing the available space that limits the viable paths in the scene can, for every X_i, strongly reduce the cardinality of all the possible outcomes of the model. Starting from the assumption that all the objects and structures of the scene are available in the form of a scene layout or a camera image, BEiT [208] is first used to output visual tokens T_s, the so-called scene-representation encodings. These tokens T_s are fed into a learnable MLP defined as $\phi_t : T_s \rightarrow T_e$ and then concatenated into the input.

The input concatenated with T_e is then fed into two modules: Equivariant block (*EquiGCN*) and Invariant block (*InvGCN*). Following [209], these two blocks are both based on the implementation of the message passing described in Equation 5.7, modified to accept also T_e:

$$m_{ij} = \phi_e \left(h_i^l, h_j^l, \left\| x_i^l - x_j^l \right\|^2, T_e, a_{ij} \right) , \tag{5.11}$$

where a_{ij} is the edge attribute (or weight), which can be derived from the adjacency matrix.

Specifically, the *EquiGCN* block is responsible for updating the node's coordinates x, and represents the implementation of the update function described in Equation 5.8. On the other hand, *InvGCN* is the implementation of the update function of the node's features h in Equation 5.9.

The possible pathways are learned by the module ϕ_t, starting from the token produced by the pre-trained BeIT model. The outputs of *EquiGCN* and *InvGCN* are computed as reported respectively in Equation (5.8) and Equation (5.9), updating h_i^{l+1} and x_i^{l+1}.

To understand the contribution of these two modules in less formal and more practical terms, imagine a navigation system. It can get from point A to point B but might struggle with tricky situations. In **SITUATE**, *EquiGCN* injects a sense of direction as a compass you wear on your hat. Specifically, it ensures that the network understands the layout of the environment, regardless of where it starts "looking". *InvGCN*, on the other hand, acts like a map you hold – it helps the network to account for different starting points and body orientations, making the predictions more robust.

5.3.2 Feature Initialization

The input given to our model is a set of trajectories of different agents. The first step is to define a node for every position x_t for every agent A_i. Every node x_t is connected to x_{t-1} and x_{t+1} if it is related to the same agent A_i. Since only trajectories (and thus positions x_t) are given as starting data, it is necessary to define for each trajectory a vector of initial features h_i^0 to be used as input together with the positions x_i^0.

As stated in [210], having an invariant feature vector h_i^0 is necessary to guarantee equivariance. Given that as input data, we only have position X_i, we follow the procedure in [209] to use velocities to create h_i^0 as follows:

$$\hat{x}_i = \phi_X(X_i + \overline{\mathbb{H}}) + \overline{\mathbb{H}}\,, \tag{5.12}$$

$$\rho_i^t = \|v_i^t\|_2\,, \tag{5.13}$$

$$\theta_i^t = \text{angle}(v_i^t, v_i^{t-1})\,, \tag{5.14}$$

$$h_i^0 = \phi_{h_0}(\rho_i, \theta_i)\,, \tag{5.15}$$

where h_i^0 is the initial features vector of the $i-th$ agent. v_i^t represent the velocity of the agent and is defined as $\triangle\hat{x}_i^t$, where $\triangle$ is the finite difference operator, $\overline{\mathbb{H}}$ is the centroid of the observed trajectories of all agents in the scene. ϕ_X and ϕ_{h_0} are two fully connected layers responsible for encoding and producing the initial graph and the initial features of the trajectory.

To compute h_i^0, two different types of velocities (thus, information invariant to rotation and translation) are needed: $\triangle x_i^t$ effectively represents the Euclidean velocity of the agent and θ_i^t represents the angular velocity at a certain time step t.

Note that both ϕ_{x_0} and ϕ_{h_0}, and in general all the operations described, are linear transformations: this is necessary to preserve both the equivariance and the invariance properties of the remaining part of the model.

5.3.3 Experimental Results

Our experimental evaluation is based on two objectives. Firstly, we show the superiority of **SITUATE** in the two most well-known indoor datasets, defining the new state-of-the-art in indoor scenarios. Secondly, we prove that **SITUATE** can also achieve results comparable to those of other competitors in outdoor datasets. Finally, we report some ablation studies.

Datasets descriptions. We evaluate **SITUATE** on state-of-the-art indoor datasets and the most well-known outdoor human trajectory prediction dataset.

THÖR. The THÖR dataset [181] includes human motion trajectory and gaze data collected in an indoor environment with accurate ground truth for the position of participants. It comprises 395K frames at 100 Hz, 2531K people detections, and more than 600 individual and group trajectories between multiple resting points. The map was taken from the dataset's official website.

Supermarket. The Supermarket dataset [211] comprises 4 different scenario: German1, German2, German3, and German4, *i.e.*, four different supermarkets. The dataset collection involved attaching devices on shopping carts/baskets and recording their movements during customer usage. Each subset includes a file with a map of the supermarket.

ETH-UCY. The ETH [212] and UCY [213] dataset group consists of five different scenes: ETH & HOTEL (from ETH) and UNIV, ZARA1, & ZARA2 (from UCY). The scenes are captured in unconstrained outdoor environments with few objects blocking the pedestrian paths. In this case, images of the scene were used.

Evaluation metrics. We use standard metrics for the trajectory prediction task, *i.e.*, minimum Average Displacement Error (ADE), and minimum Final Displacement Error (FDE). In particular, ADE measures the average $L2$ difference between the

prediction in all time steps and the ground truth. On the other hand, FDE measures the difference between the predicted endpoint and the ground truth.

Prediction mode. Following the evaluation protocol of [209], **SITUATE** is used in two prediction modes: deterministic and multi-prediction. Deterministic means that the model only outputs a single prediction for each input motion observation, while multi-prediction means that the model has 20 predictions for each input motion observation. Under multi-prediction, ADE and FDE will be calculated using the best-predicted trajectory. To adapt to multi-prediction, we modify **SITUATE** to repeat the last feature updating layer and the output layer 20 times in parallel to have a multi-head prediction.

Implementation details. As a backbone for our model, we used the structure of [209]. The model architecture has four layers of geometric feature learning. We use the SiLU activation function and dropout with a probability of 0.5 within all MLPs. The visual embeddings of the image are derived from the last layer of the BEiT model. The model is provided with past trajectory information that spans eight discrete time steps, and the model's task is to predict 12 steps into the future. In addition to the dropout mentioned above, we apply the Discrete Cosine Transform (DCT) to the input data as a regularization technique. Specifically, by representing the data in the frequency domain, it becomes easier to distinguish between signal and noise components, resulting in a cleaner signal. We train our models with a batch size of 64 for 60 epochs, using AdamW [67] as an optimizer within the PyTorch Lightning framework on an NVIDIA RTX 3090.

Indoor Human Trajectory Prediction Results

We conducted comparative experiments to assess the robustness of our approach against existing trajectory prediction methods. The methods include deterministic evaluation models (TransF [176], MemoNet [178]), as well as multi-prediction evaluation models (PECNet [214], GP-Graph [179]). Our evaluation for both prediction modes also includes EqMotion [209], the state-of-the-art method with invariant end equivariant interaction reasoning. Table 5.6 and Table 5.7 show the results.

Table 5.6 Deterministic prediction performance (ADE (m)/FDE (m)) on the THÖR and the Supermarket datasets. The **bold**/underlined font denotes the **best**/second-best result.

	Performance (ADE (m) ↓ / FDE (m) ↓)		
Deterministic Evaluation	**THÖR**	**Supermarket**	**Average**
TransF [176]	2.62/4.81	2.56/2.90	2.59/3.85
MemoNet [178]	0.78/5.05	1.79/2.94	1.28/3.99
EqMotion [209]	0.56/0.94	1.71/2.65	1.13/1.79
SITUATE (ours)	**0.45/0.93**	**1.21/1.84**	**0.83/1.38**

Table 5.7 Multi-prediction performance (ADE (m)/FDE (m)) on the THÖR and the Supermarket datasets. The **bold**/underlined font denotes the **best**/second-best result.

	Performance (ADE (m) ↓ / FDE (m) ↓)		
Multi-prediction Evaluation	**THÖR**	**Supermarket**	**Average**
PECNet [214]	–	1.57/3.45	–
GP-Graph [179]	2.80/3.92	3.19/4.57	2.99/4.24
EqMotion [209]	1.32/1.03	1.29/1.77	2.61/1.40
SITUATE (ours)	**0.50/0.86**	**0.53/0.65**	**0.51/0.75**

The results show that the proposed **SITUATE** consistently outperforms all baseline methods in all cases. On the THÖR dataset, **SITUATE** achieves an ADE of 0.45 and an FDE of 0.93, showcasing its superiority over other models. In particular, compared to EqMotion, the second-best model **SITUATE** exhibits a substantial 22% reduction in ADE and a 1% reduction in FDE.

Similarly, on the Supermarket dataset, **SITUATE** continues to demonstrate its effectiveness with an ADE of 1.21 and an FDE of 1.84. Compared to EqMotion, again the closest competitor, **SITUATE** achieves a 29% reduction in ADE and a 31% in FDE, reinforcing its dominance.

The consistently good performance of **SITUATE** in both data sets underscores its robustness and efficacy in trajectory prediction tasks. The results further suggest that **SITUATE** is accurate and that the use of scene information when tackling indoor prediction scenarios offers a key advantage compared to the available approaches.

Table 5.8 Deterministic prediction performance (ADE (m)/FDE (m)) on the state-of-the-art ETH-UCY benchmark. The **bold**/underlined font denotes the **best**/second-best result.

	Performance (ADE (m) ↓ / FDE (m) ↓)					
Deterministic	**ETH**	**HOTEL**	**UNIV**	**ZARA1**	**ZARA2**	**Average**
S-LSTM [173]	1.09/2.35	0.79/1.76	0.67/1.40	0.47/1.00	0.56/1.17	0.72/1.54
SGAN-ind [174]	1.13/2.21	1.01/2.18	0.60/1.28	0.42/0.91	0.52/1.11	0.74/1.54
Traj++ [175]	1.02/2.00	0.33/0.62	0.53/1.19	0.44/0.99	0.32/0.73	0.53/1.11
TransF [176]	1.03/2.10	0.36/0.71	0.53/1.32	0.44/1.00	0.34/0.76	0.54/1.17
MemoNet [178]	1.00/2.08	0.35/0.67	0.55/1.19	0.46/1.00	0.37/0.82	0.55/1.15
EqMotion [209]	0.96/1.92	**0.30**/0.58	**0.50/1.10**	**0.39/0.86**	**0.30/0.68**	**0.49/1.03**
SITUATE (ours)	**0.94/1.90**	**0.30/0.57**	**0.50/1.10**	0.41/0.89	0.32/0.70	**0.49/1.03**

Table 5.9 Multi-prediction performance (ADE (m)/FDE (m)) on the state-of-the-art ETH-UCY benchmark. The **bold**/underlined font denotes the **best**/second-best result.

	Performance (ADE (m) ↓ / FDE (m) ↓)					
Multi-prediction	**ETH**	**HOTEL**	**UNIV**	**ZARA1**	**ZARA2**	**Average**
SGAN [174]	0.87/1.62	0.67/1.37	0.76/0.52	0.35/0.68	0.42/0.84	0.61/1.21
STGAT [215]	0.65/1.12	0.35/0.66	0.34/0.69	0.29/0.60	0.52/1.10	0.43/0.83
STAR [216]	0.36/0.65	0.17/0.36	0.31/0.62	0.29/0.52	0.22/0.46	0.26/0.53
NMMP [217]	0.61/1.08	0.33/0.63	0.52/1.11	0.32/0.66	0.43/0.85	0.41/0.82
Traj++ [175]	0.61/1.02	0.19/0.28	0.30/0.54	0.24/0.42	0.18/0.31	0.30/0.51
PECNet [214]	0.54/0.87	0.18/0.24	0.35/0.60	0.22/0.39	0.17/0.30	0.29/0.48
Agentformer [218]	0.45/0.75	0.14/0.22	0.25/0.45	0.18/**0.30**	0.14/0.24	0.23/0.39
GroupNet [219]	0.46/0.73	0.15/0.25	0.26/0.49	0.21/0.39	0.17/0.33	0.25/0.44
MID [177]	**0.39**/0.66	0.13/0.22	**0.22**/0.45	**0.17/0.30**	**0.13**/0.27	**0.21**/0.38
GP-Graph [179]	0.43/0.63	0.18/0.30	0.24/**0.42**	**0.17**/0.31	0.15/0.29	0.23/0.39
EqMotion [209]	0.40/**0.61**	**0.12/0.18**	0.23/0.43	0.18/0.32	**0.13/0.23**	**0.21/0.35**
SITUATE (ours)	0.41/0.64	0.13/0.20	0.23/0.43	0.22/0.35	0.14/0.26	0.22/0.37

Outdoor Human Trajectory Prediction Results

We also evaluate the performance of **SITUATE** with the deterministic and multi-prediction modalities with outdoor scenarios. Here, we show that **SITUATE** achieves competitive results, showing that indoor-oriented forecasting models tend to generalize better than outdoor-oriented ones. Table 5.8 and Table 5.9 present the quantitative results.

Specifically, when considering the deterministic prediction case, **SITUATE** shows a performance improvement by obtaining state-of-the-art results in both ADE and FDE across the ETH (0.94/1.90), HOTEL (0.30/0.57), and UNIV (0.50/1.10)

Table 5.10 Ablation results (ADE (*m*)/FDE (*m*)) of **SITUATE**. We assess the contribution of the scene representation module and regularization methods in the deterministic prediction case.

		Performance (ADE (*m*) ↓ / FDE (*m*) ↓)		
Scene Representation	**Regularization**	**THÖR**	**Supermarket**	**Average**
✗	✗	0.50/1.02	1.92/1.55	1.21/1.29
✗	✓	0.56/0.74	1.79/2.94	1.18/1.84
✓	✗	0.57/0.96	1.29/1.89	0.93/1.43
✓	✓	**0.45/0.93**	**1.21/1.84**	**0.83/1.38**

scenes. It ranks second in the ZARA1 and ZARA2 scenes, while performing on par with EqMotion [209] when considering average performance. In the context of multi-prediction modality, **SITUATE** secures the second rank in terms of ADE and FDE across nearly 75% of the scenes within the ETH-UCY dataset while maintaining an overall second place in average performance.

Designed for indoor scenarios and their peculiar conformations, **SITUATE** remarkably demonstrates robust capabilities, even when tested on outdoor datasets. In contrast, this is not always true for architectures tailored for outdoor instances, as we can observe in Table 5.6 and Table 5.7, which often struggle when confronted with scenes that differ from those for which they were designed.

Ablation Studies

We quantitatively evaluate the impact of the scene representation module and regularization methods considering the deterministic indoor prediction scenario. The results are summarized in Table 5.10. It is observed that both contributions play a crucial role in enhancing the overall performance of **SITUATE**. In particular, the scene representation module effectively encodes semantic information from the visual scene maps, facilitating an accurate understanding of the environment. On the other hand, regularization methods ensure robustness and generalization of the model by mitigating overfitting and improving its ability to generalize to unseen data. Therefore, we state that both contributions are indispensable for achieving the desired outcomes and validating the efficacy of our approach.

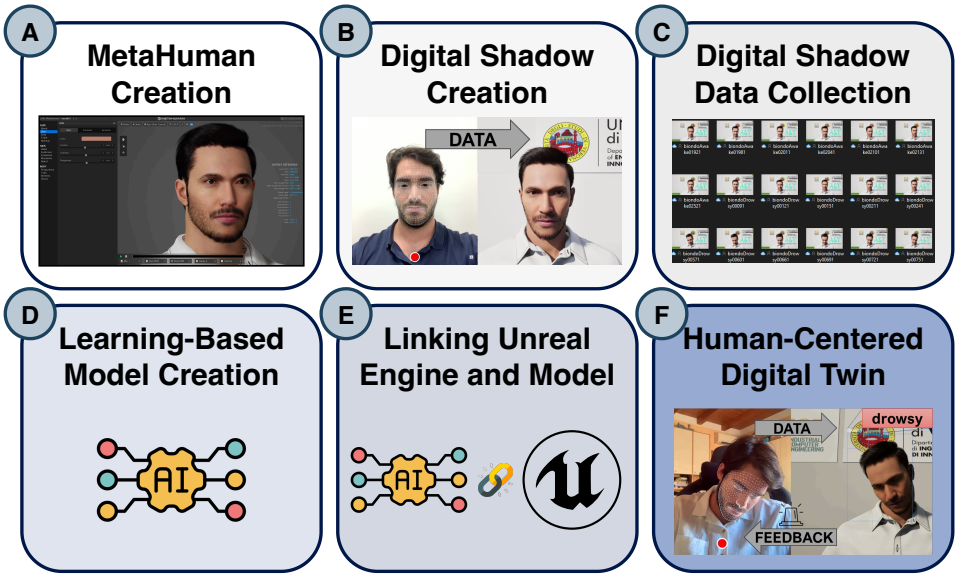


Figure 5.5 (A) The process starts by creating a MetaHuman, *i.e.*, a virtual replica of the real human. (B) The connection between the real human and the virtual replica is established using the LiveLink plugin, forming the Digital Shadow that allows real-time data acquisition. (C) Then, images were acquired and labeled to capture the operator's real-time movements under awake and drowsy conditions. (D) A state-of-the-art deep learning model, *i.e.*, the YOLOv8, is trained to detect the operator's state. (E) A client-server architecture is set up to facilitate the communication between YOLOv8 and Unreal Engine. (F) Finally, the DT reflects the operator's state (in our case, awake or drowsy) and provides alerts based on real-time conditions.

5.4 HC-DT

Our human-centered DT development process involves six stages, which are depicted in Figure 5.5 and explained in detail in this section. In this research, we focus on the wakefulness and drowsiness detection task, although our pipeline can be easily adapted to other scenarios by modifying component-specific details.

Section Organization

This section details the implementation steps for **HC-DT** (Section 5.4.1) and concludes with a summary of the findings (Section 5.4.2).

5.4.1 HC-DT Creation

A) MetaHuman Creation

The first stage involves creating a highly accurate digital replica of the human operator using Unreal Engine's MetaHuman Creator, the state-of-the-art tool for generating hyper-realistic virtual humans. The Web interface allows the creation of a detailed virtual replica by customizing facial features and general appearance, ensuring that even individuals with minimal computer graphics expertise can generate highly realistic digital humans.

B) Digital Shadow Creation

The second stage involves creating the Digital Shadow, *i.e.*, a virtual representation continuously synchronized with the operator's real-world movements. In this phase, we used the Epic Games LiveLink Face application. Unlike full-body motion capture systems that require specialized suits, LiveLink uses a simple smartphone or tablet, eliminating the need for complex hardware or sensors and making it easier to adopt and scale.

C) Digital Shadow Data Collection

In this stage, data were collected to capture the real-time movements of the operator under both awake and drowsy conditions. A Python script was used to automatically capture screenshots from Unreal Engine, ensuring diverse and high-quality data by acquiring images from various angles and positions.

Moreover, the Industrial Metaverse allows data collection to be conducted remotely, enabling operators to work from the office or even from home. This capability not only provides flexibility, but also ensures that factory operations remain uninterrupted.

D) Learning-Based Model Creation

Our wakefulness and drowsiness detection model, based on the YOLOv8 architecture [220], classifies each detected object as awake or drowsy. The model achieved

Figure 5.6 Our implemented human-centered DT integrated within the manufacturing DT of the ICE Laboratory at the University of Verona.

high evaluation metrics, with an accuracy of 99.5%, a precision of 1.000 (wakefulness) and 0.965 (drowsiness), and a recall of 0.998 (wakefulness) and 1.000 (drowsiness). These results demonstrate a highly accurate and reliable detection of wakefulness and drowsiness in our acquired data with minimal false positives or missed detections.

E) Linking Unreal Engine and the AI Model

After training, the model was integrated into Unreal Engine using a client-server architecture. This setup ensures that any changes in the operator's state are instantly reflected in the DT, facilitating immediate feedback and interaction.

F) Human-Centered Digital Twin

The DT system continuously monitors the operator's state, updating the virtual replica in real-time based on the model's predictions. In the final phase, the fully functional DT was evaluated for both its real-time interaction capabilities and the quality of its output.

Finally, the human-centered developed DT was validated in a real manufacturing plant DT of the ICE Laboratory, as shown in Figure 5.6. This validation demonstrates how our pipeline allowed easy integration of the human-centered DT into a new existing machine-oriented DT ecosystem, which was previously focused only on CPS and machine-centric applications.

5.4.2 Remarks and Outlook

The proposed human-centered DT pipeline demonstrates a practical and cost-effective approach to real-time worker monitoring and interaction within industrial environments.

Although extensive research has been conducted on DT technologies and their role in Industry 4.0 and 5.0, practical implementations remain limited. Many existing works are theoretical, and fully operational systems that integrate human DTs into industrial settings are scarce. This pipeline marks the initial effort to address this gap and offers the first working solution.

However, this system has some limitations. The reliance on camera-based facial tracking restricts the operator's movement to the camera's field of view, which can be a constraint in dynamic industrial environments. Therefore, our solution is better suited for operators working in fixed production cells or vehicles than for workers who need to move throughout the factory. Moreover, although MetaHuman Creator offers realistic digital replicas, it lacks customization options to represent various body types and individuals with disabilities. Addressing these limitations will make the solution more inclusive and versatile. Finally, relying on camera-based tracking alone presents inherent limitations in large-scale and dynamic industrial environments.

Despite these considerations, the pipeline represents a fundamental step toward the creation of human DTs, establishing a foundation for future advancements and practical applications across diverse industries. Moreover, it highlights that the Industrial Metaverse is not merely a futuristic concept, but a tangible and viable solution for addressing real-world challenges.

Chapter 6

Conclusions

In this final chapter, we discuss the findings and insights presented in this thesis, reviewing the contributions made in each area of intelligent manufacturing from a learning-based point of view. For each topic, *i.e.*, efficient on-edge computing, accurate anomaly detection, sustainable NFPPF, and human-centered systems design, we review the methods presented, highlighting their results and scientific achievements. We also account for the limitations and open issues we faced in developing these solutions, which are themselves as important as the successes. After that, we present some possible future work and possibilities for improvement, hoping they can help the research community create novel and more sophisticated computer vision techniques to serve the digital world of tomorrow.

6.1 Summary

In the introduction of this thesis, *i.e.*, Chapter 1, we discussed how the continuous advancement of technology is revolutionizing various aspects of intelligent manufacturing. The thesis addresses the foundational pillars of intelligent manufacturing: efficient on-edge computing, precise anomaly detection, sustainable forecasting of new product performance, and human-centered system design. By applying cutting-edge deep learning techniques and developing novel frameworks, we aim to enhance the efficiency, accuracy, and sustainability of industrial processes while ensuring the integration of human-centric principles.

Each chapter of this thesis focuses on a specific domain within these pillars. Chapter 2 elaborates on advances in on-edge computing, highlighting the importance of SC methodologies. The chapter presents innovative frameworks such as **I-SPLIT**, which utilizes deep network interpretability for optimal split point selection, and **LO-SC**, which enables local-only computation on edge devices without compromising accuracy. These contributions are essential for real-time data processing in resource-constrained environments, facilitating seamless integration between edge devices and cloud servers.

In Chapter 3, the focus shifts to anomaly detection, a critical component of predictive maintenance in intelligent manufacturing systems. The chapter introduces state-of-the-art methodologies such as **In&Out** and **ChronosAD**, which leverage data augmentation and time-series foundation models, respectively, to improve the identification of anomalies in both vision-based and time-series data. These approaches improve the robustness of anomaly detection systems, ensuring reliable operation and minimizing downtime in industrial settings.

Chapter 4 addresses the urgent need for sustainability in manufacturing, particularly in the fast fashion industry, known for its significant environmental impact. The chapter presents **MDiFF** and **Dif4FF**, two multimodal diffusion-based forecasting frameworks designed to accurately predict the performance of new fashion products. These frameworks incorporate advanced modeling techniques to support sustainable production decisions, thus reducing waste and optimizing resource utilization.

Finally, Chapter 5 explores the design of human-centered systems, a crucial aspect of Industry 5.0 that emphasizes the integration of human operators in automated environments. This chapter introduces systems such as **WiFi-Based Detection** and **HC-DT**, which focus on ensuring worker safety and improving human-machine interactions. By developing the first pipeline to implement a human-centered DTs, the chapter shows how advanced learning techniques can improve both operational efficiency and worker well-being.

The comprehensive exploration of these domains throughout the thesis provides a holistic view of intelligent manufacturing and its evolution in the context of Industry 5.0. By integrating advanced AI methodologies with practical industrial applications, we have demonstrated how these innovations can address the challenges faced by modern manufacturing systems. This thesis lays the foundation for future research that aims to further advance the field of intelligent manufacturing, emphasizing the

need for continued innovation in efficiency, accuracy, sustainability, and human-centric design.

6.2 Limitations and Future Works

Challenges in pipeline complexity and computational efficiency. Throughout this thesis, we addressed numerous challenges in intelligent manufacturing, focusing on the integration of advanced learning-based methods within on-edge environments, anomaly detection, sustainable product forecasting, and human-centered design. However, the proposed solutions often involved the development of complex pipelines that comprise multiple deep learning models that function together, where the output of one model serves as the input of the next. This approach, while effective, necessitates intricate coordination between various component models.

For example, in the edge cloud computing domain, the implementation of SC architectures requires a careful selection of split points, hardware devices, and network implementation to balance computation between edge devices and cloud servers. These complex setups highlight the need for robust techniques that can handle dependencies and potential failures.

Future research should focus on developing more resilient and flexible methodologies that can operate effectively even when individual components encounter partial or incorrect data. In addition, optimizing these pipelines to reduce computational overhead and enable real-time processing remains a critical area for improvement.

Specialized industrial applications require specialized annotators. Given the specialized nature of each manufacturing, the creation of ad hoc datasets tailored to specific use cases is often necessary. In domains such as predictive maintenance and human-centered system design, ensuring the quality and consistency of these annotations is paramount to achieving reliable results.

Future work should focus on expanding the availability of high-quality, domain-specific datasets by incorporating emerging sensor technologies and fostering collaborative efforts among industry partners, academic institutions, and research organizations. This collaboration can help build comprehensive, shared data resources that not only improve model performance but also facilitate the development of more

robust and generalizable solutions across different applications. By creating such datasets, we can bridge data availability gaps and enhance the scalability of solutions in complex, real-world environments.

Enhancing the generalization of sustainable forecasting models. The proposed multimodal diffusion-based forecasting models for the performance of new fashion products have shown promising results. However, a major difficulty in predicting fashion trends lies in the ever-changing nature of consumer tastes. Fashion does not follow a regular cycle; trends continuously shift and change. In fact, while Google Trends can indicate the popularity of a garment in terms of searches, it does not guarantee that the item searched matches the exact appearance and features of the target garment.

As a result, future work should aim to individuate techniques to increase the expressivity of these models. For example, automatic analysis of the human pose represents a source of valuable features in a wide range of tasks and domains, and its integration into fashion and garment fitting models presents several possibilities to improve the accuracy of NFPPF. One key development in this field is to use detailed pose and mesh analysis to enhance NFPPF models. By incorporating the knowledge of human body poses into these models, we can significantly improve how garments are represented and how well they align with different body types, enhancing both the virtual fitting experience and the overall garment design process.

Scalability of human-centered digital twins in industrial environments. The integration of human-centered DT in industrial systems presents challenges related to scalability. Although the proposed framework demonstrates feasibility in controlled environments, the deployment of HC-DT on a large scale in different industrial settings requires further investigation. Factors such as real-time synchronization, hardware limitations, and the adaptability of learning models to various working conditions must be addressed.

For example, the current implementation relies on camera-based facial tracking, which may not be practical in environments with high worker mobility. In addition, ensuring seamless interaction between DTs and factory control systems remains an open challenge. Future research should focus on developing lightweight tracking methods that operate with minimal infrastructure while maintaining accuracy. Fur-

thermore, integrating these systems with larger DT ecosystems in industrial plants will require standardized protocols and interoperability frameworks.

Bibliography

[1] Luigi Capogrosso, Federico Cunico, Dong Seon Cheng, Franco Fummi, and Marco Cristani. A Machine Learning-Oriented Survey on Tiny Machine Learning. *IEEE Access*, 12:23406–23426, 2024.

[2] Federico Cunico, Luigi Capogrosso, Francesco Setti, Damiano Carra, Franco Fummi, and Marco Cristani. I-SPLIT: Deep Network Interpretability for Split Computing. In *26th International Conference on Pattern Recognition (ICPR)*, 2022.

[3] Luigi Capogrosso, Federico Cunico, Michele Lora, Marco Cristani, Franco Fummi, and Davide Quaglia. Split-Et-Impera: A Framework for the Design of Distributed Deep Learning Applications. In *26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023.

[4] Luigi Capogrosso, Enrico Fraccaroli, Samarjit Chakraborty, Franco Fummi, and Marco Cristani. MTL-Split: Multi-Task Learning for Edge Devices using Split Computing. In *61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

[5] Luigi Capogrosso, Enrico Fraccaroli, Giulio Petrozziello, Francesco Setti, Samarjit Chakraborty, Franco Fummi, and Marco Cristani. Enhancing Split Computing and Early Exit Applications through Predefined Sparsity. In *Forum on Specification & Design Languages (FDL)*, 2024.

[6] Luigi Capogrosso, Enrico Fraccaroli, Marco Cristani, Franco Fummi, and Samarjit Chakraborty. LO-SC: Local-Only Split Computing for Accurate Deep Learning on Edge Devices. In *38th International Conference on VLSI Design (VLSID)*, 2025.

[7] Luigi Capogrosso, Shengjie Xu, Enrico Fraccaroli, Marco Cristani, Franco Fummi, and Samarjit Chakraborty. Learning-Enabled CPS for Edge-Cloud Computing. In *14th International Symposium on Industrial Embedded Systems (SIES)*, 2024.

[8] Uzair Khan, Dong Seon Cheng, Francesco Setti, Franco Fummi, Marco Cristani, and Luigi Capogrosso. A Comprehensive Survey on Deep Learning-based Predictive Maintenance. *ACM Transactions on Embedded Computing Systems*, 2025.

[9] Luigi Capogrosso, Federico Girella, Francesco Taioli, Michele Chiara, Muhammad Aqeel, Franco Fummi, Francesco Setti, and Marco Cristani. Diffusion-Based Image Generation for In-Distribution Data Augmentation in Surface Defect Detection. In *19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2024.

[10] Federico Girella, Ziyue Liu, Franco Fummi, Francesco Setti, Marco Cristani, and Luigi Capogrosso. Leveraging Latent Diffusion Models for Training-Free in-Distribution Data Augmentation for Surface Defect Detection. In *International Conference on Content-Based Multimedia Indexing (CBMI)*, 2024.

[11] Francesco Biondani, Luigi Capogrosso, Nicola Dall'Ora, Enrico Fraccaroli, Marco Cristani, and Franco Fummi. GAIA: A Comprehensive Pipeline for Enabling Aircraft Digital Twin Creation. *34th IEEE International Symposium on Industrial Electronics (ISIE)*, 2025.

[12] Uzair Khan, Dong Seon Cheng, Francesco Setti, Franco Fummi, Marco Cristani, and Luigi Capogrosso. ChronosAD: Leveraging Time-Series Foundation Models for Anomaly Detection. *Engineering Applications of Artificial Intelligence (EAAI)*, 2025.

[13] Andrea Avogaro, Luigi Capogrosso, Andrea Toaiari, Franco Fummi, and Marco Cristani. New Fashion Products Performance Forecasting: A Survey on Evolutions, Models and Emerging Trends. *arXiv preprint arXiv:2501.10324*, 2025.

[14] Andrea Avogaro, Luigi Capogrosso, Franco Fummi, and Marco Cristani. MD-iFF: Exploiting Multimodal Score-based Diffusion Models for New Fashion Product Performance Forecasting. *arXiv preprint arXiv:2412.06840*, 2024.

[15] Andrea Avogaro, Luigi Capogrosso, Franco Fummi, and Marco Cristani. Dif4FF: Leveraging Multimodal Diffusion Models and Graph Neural Networks for Accurate New Fashion Product Performance Forecasting. In *International Conference on Pattern Recognition (ICPR)*, 2024.

[16] Cristian Turetta, Geri Skenderi, Luigi Capogrosso, Florenc Demrozi, Philipp H. Kindt, Alejandro Masrur, Franco Fummi, Marco Cristani, and Graziano Pravadelli. Towards Deep Learning-based Occupancy Detection Via WiFi Sensing in Unconstrained Environments. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[17] Luigi Capogrosso, Andrea Toaiari, Andrea Avogaro, Uzair Khan, Aditya Jivoji, Franco Fummi, and Marco Cristani. SITUATE: Indoor Human Trajectory Prediction Through Geometric Features and Self-supervised Vision Representation. In *27th International Conference on Pattern Recognition (ICPR)*, 2024.

[18] Francesco Biondani, Luigi Capogrosso, Nicola Dall'Ora, Enrico Fraccaroli, Marco Cristani, and Franco Fummi. Human-Centered Digital Twin for Industry 5.0. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2025.

[19] Yaroslav Bulatov. notMNIST dataset. https://www.kaggle.com/datasets/jwjohnson314/notmnist. Accessed: 2024-11-03.

[20] Paul Mooney. Chest X-Ray Images (Pneumonia) dataset. https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia. Accessed: 2024-11-03.

[21] Marcel Kollovieh, Abdul Fatir Ansari, Michael Bohlke-Schneider, Jasper Zschiegner, Hao Wang, and Yuyang Bernie Wang. Predict, Refine, Synthesize: Self-Guiding Diffusion Models for Probabilistic Time Series Forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[22] Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[23] Jakob Božič, Domen Tabernik, and Danijel Skočaj. Mixed supervision for surface-defect detection: From weakly to fully supervised learning. *Computers in Industry*, 129:103459, 2021.

[24] Mario Hermann, Tobias Pentek, and Boris Otto. Design Principles for Industrie 4.0 Scenarios. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016.

[25] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

[26] Mohd Azeem, Abid Haleem, and Mohd Javaid. Symbiotic Relationship Between Machine Learning and Industry 4.0: A Review. *Journal of Industrial Integration and Management*, 7(03):401–433, 2022.

[27] Praveen Kumar Reddy Maddikunta, Quoc-Viet Pham, Prabadevi B, N Deepa, Kapal Dev, Thippa Reddy Gadekallu, Rukhsana Ruby, and Madhusanka Liyanage. Industry 5.0: A survey on enabling technologies and potential applications. *Journal of Industrial Information Integration*, 26:100257, 2022.

[28] Lucas Santos Dalenogare, Guilherme Brittes Benitez, Néstor Fabián Ayala, and Alejandro Germán Frank. The expected contribution of Industry 4.0 technologies for industrial performance. *International Journal of production economics*, 204:383–394, 2018.

[29] Anbesh Jamwal, Rajeev Agrawal, Monica Sharma, and Antonio Giallanza. Industry 4.0 Technologies for Manufacturing Sustainability: A Systematic Review and Future Research Directions. *Applied Sciences*, 11(12):5725, 2021.

[30] Morteza Ghobakhloo. Industry 4.0, digitization, and opportunities for sustainability. *Journal of Cleaner Production*, 252:119869, 2020.

[31] Federico Cunico, Stefano Aldegheri, Andrea Avogaro, Michele Boldo, Nicola Bombieri, Luigi Capogrosso, Ariel Caputo, Damiano Carra, Stefano Centomo, Dong Seon Cheng, Ettore Cinquetti, Marco Cristani, Mirco De Marchi, Florenc Demrozi, Marco Emporio, Franco Fummi, Luca Geretti, Samuele Germiniani, Andrea Giachetti, Federico Girella, Enrico Martini, Gloria Menegaz, Niek Muijs, Federica Paci, Marco Panato, Graziano Pravadelli, Elisa Quintarelli, Ilaria Siviero, Silvia Francesca Storti, Carlo Tadiello, Cristian Turetta, Tiziano Villa, Nicola Zannone, and Davide Quaglia. Enhancing Safety and Privacy in Industry 4.0: The ICE Laboratory Case Study. *IEEE Access*, 12:154570–154599, 2024.

[32] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Computing Surveys*, 55(5):1–30, 2022.

[33] Kirsi Niinimäki, Greg Peters, Helena Dahlbo, Patsy Perry, Timo Rissanen, and Alison Gwilt. The environmental price of fast fashion. *Nature Reviews Earth & Environment*, 1(4):189–200, 2020.

[34] Kerrice Bailey, Aman Basu, and Sapna Sharma. The Environmental Impacts of Fast Fashion on Water Quality: A Systematic Review. *Water*, 14(7):1073, 2022.

[35] Saeid Nahavandi. Industry 5.0—A Human-Centric Solution. *Sustainability*, 11(16):4371, 2019.

[36] Xun Xu, Yuqian Lu, Birgit Vogel-Heuser, and Lihui Wang. Industry 4.0 and Industry 5.0—Inception, conception and perception. *Journal of Manufacturing Systems*, 61:530–535, 2021.

[37] Christian Joppi, Geri Skenderi, and Marco Cristani. POP: Mining POtential Performance of New Fashion Products via Webly Cross-modal Query Expansion. In *European Conference on Computer Vision (ECCV)*, 2022.

[38] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ACM SIGPLAN Notices*, 52(1):615–629, 2017.

[39] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems. In *Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2019.

[40] Marion Sbai, Muhamad Risqi U Saputra, Niki Trigoni, and Andrew Markham. Cut, Distil and Encode (CDE): Split Cloud-Edge Deep Inference. In *18th*

Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2021.

[41] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[42] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity Checks for Saliency Maps. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[43] Rich Caruana. *Multitask Learning*, volume 28, pages 95–133. Springer US, 1998.

[44] Etienne Boursier, Mikhail Konobeev, and Nicolas Flammarion. Trace norm regularization for multi-task learning with scarce data. In *35th Annual Conference on Learning Theory (COLT)*, 2022.

[45] Harvey M. Salkin and Cornelis A. De Kluyver. The knapsack problem: A survey. *Naval Research Logistics Quarterly*, 22(1):127–144, 1975.

[46] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for MobileNetV3. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[47] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *Neurocomputing*, 461:370–403, 2021.

[48] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge Distillation: A Survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[49] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge. In *Artificial Neural Networks and Machine Learning (ICANN)*, 2018.

[50] Hyomin Choi and Ivan V Bajić. Deep Feature Compression for Collaborative Object Detection. In *25th International Conference on Image Processing (ICIP)*, 2018.

[51] Damiano Carra and Giovanni Neglia. DNN Split Computing: Quantization and Run-Length Coding are Enough. In *Global Communications Conference (GLOBECOM)*, 2023.

[52] Yoshitomo Matsubara, Davide Callegaro, Sameer Singh, Marco Levorato, and Francesco Restuccia. BottleFit: Learning Compressed Representations in Deep Neural Networks for Effective and Efficient Split Computing. In *23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022.

[53] Chi Lo, Yu-Yi Su, Chun-Yi Lee, and Shih-Chieh Chang. A Dynamic Deep Neural Network Design for Efficient Workload Allocation in Edge Computing. In *IEEE International Conference on Computer Design (ICCD)*, 2017.

[54] Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. *Autoencoders*, pages 193–208. Elsevier, 2020.

[55] Ya Le and Xuan Yang. Tiny ImageNet Visual Recognition Challenge. *CS 231N*, 7(7):3, 2015.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[57] F. Fummi, D. Quaglia, and F. Stefanni. A SystemC-based framework for modeling and simulation of networked embedded systems. In *2008 Forum on Specification, Verification and Design Languages*, 2008.

[58] Darcy Bullock, Brian Johnson, Richard B. Wells, Michael Kyte, and Zhen Li. Hardware-in-the-loop simulation. *Transportation Research Part C: Emerging Technologies*, 12(1):73–89, 2004.

[59] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[60] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference for Learning Representations (ICLR)*, 2015.

[61] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. *Toronto, ON, Canada*, 2009.

[62] Mingxing Tan and Quoc Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning (ICML)*, 2019.

[63] Chris Burgess and Hyunjik Kim. 3D Shapes Dataset. https://github.com/deepmind/3dshapes-dataset/. Accessed: 2024-11-03.

[64] Firoj Alam, Tanvirul Alam, Md Arid Hasan, Abul Hasnat, Muhammad Imran, and Ferda Ofli. MEDIC: A Multi-Task Learning Dataset for Disaster Image Classification. *Neural Computing and Applications*, 35(3):2609–2632, 2023.

[65] Natalie C Ebner, Michaela Riediger, and Ulman Lindenberger. FACES—A database of facial expressions in young, middle-aged, and older women and men: Development and validation. *Behavior Research Methods*, 42(1):351–362, 2010.

[66] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[67] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[68] Roberto Cipolla, Yarin Gal, and Alex Kendall. Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[69] Xiaogang Xu, Hengshuang Zhao, Vibhav Vineet, Ser-Nam Lim, and Antonio Torralba. MTFormer: Multi-task Learning via Transformer and Cross-Task Reasoning. In *European Conference on Computer Vision (ECCV)*, 2022.

[70] Sourya Dey, Kuan-Wen Huang, Peter A. Beerel, and Keith M. Chugg. Pre-Defined Sparse Neural Networks With Hardware Acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):332–345, 2019.

[71] Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O Arik, and Tomas Pfister. TSMixer: An All-MLP Architecture for Time Series Forecasting. *Transactions on Machine Learning Research (TMLR)*, 2023.

[72] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[73] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)n*, 2009.

[74] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

[75] Laurent Perron and Frédéric Didier. CP-SAT.

[76] Song Han, Jeff Pool, John Tran, and William Dally. Learning both Weights and Connections for Efficient Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[77] Tingan Zhu, Prateek Ganguli, Arkaprava Gupta, Shengjie Xu, Luigi Capogrosso, Enrico Fraccaroli, Marco Cristani, Franco Fummi, and Samarjit Chakraborty. Controllers for Edge-Cloud Cyber-Physical Systems. In *17th International Conference on COMmunication Systems and NETworks (COMSNETS)*, 2025.

[78] Tian Wang, Yang Chen, Meina Qiao, and Hichem Snoussi. A fast and robust convolutional neural network-based defect detection model in product quality control. *The International Journal of Advanced Manufacturing Technology*, 94(9–12):3465–3471, 2017.

[79] Colin SC Tsang, Henry YT Ngan, and Grantham KH Pang. Fabric inspection based on the Elo rating method. *Pattern Recognition*, 51:378–394, 2016.

[80] Saeed Hosseinzadeh Hanzaei, Ahmad Afshar, and Farshad Barazandeh. Automatic detection and classification of the ceramic tiles' surface defects. *Pattern Recognition*, 66:174–189, 2017.

[81] Hanqiu Deng and Xingyu Li. Anomaly Detection via Reverse Distillation from One-Class Embedding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[82] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Scholkopf, Thomas Brox, and Peter Gehler. Towards Total Recall in Industrial Anomaly Detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[83] Marco Rudolph, Bastian Wandt, and Bodo Rosenhahn. Same Same But DifferNet: Semi-Supervised Defect Detection with Normalizing Flows. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021.

[84] Yisheng Song, Ting Wang, Puyu Cai, Subrota K Mondal, and Jyoti Prakash Sahoo. A Comprehensive Survey of Few-shot Learning: Evolution, Applications, Challenges, and Opportunities. *ACM Computing Surveys*, 55(13s):1–40, 2023.

[85] Yajun Chen, Yuanyuan Ding, Fan Zhao, Erhu Zhang, Zhangnan Wu, and Linhao Shao. Surface Defect Detection Methods for Industrial Products: A Review. *Applied Sciences*, 11(16):7657, 2021.

[86] Luigi Capogrosso, Alvise Vivenza, Andrea Chiarini, Francesco Setti, and Marco Cristani. Exploiting Multimodal Latent Diffusion Models for Accurate Anomaly Detection in Industry 5.0. In *Ital-IA 2024: 4th National Conference on Artificial Intelligence, organized by CINI*, 2024.

[87] Sophia Boing Righetto, Leandro Takeshi Hattori, Guilherme Goncalves Nunes, Edgar Gerevini Carvalho, Marcos A. Izumida Martins, and Silvia De Francisci. Failure Prediction in Automatic Reclosers Using Machine Learning Approaches. In *IEEE URUCON*, 2021.

[88] Emin Mammadov, Mostafa Farrokhabadi, and Claudio A. Canizares. AI-enabled Predictive Maintenance of Wind Generators. In *IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, 2021.

[89] Jan Przepióra, Patryk Bałazy, Piotr Książek, Paweł Knap, Paweł Pasławski, and Michał Pietrzak. Smart IoT platform for Big Data analysis in predictive maintenance. In *24th International Carpathian Control Conference (ICCC)*, 2023.

[90] Shishir Shekhar and Shashwat Shekhar. Improving Utility Cables Diagnostics and Prognostics using Machine Learning. In *IEEE PES Grid Edge Technologies Conference & Exposition (Grid Edge)*, 2023.

[91] Guoxing Lan, Qing Li, and Nong Cheng. Remaining Useful Life Estimation of Turbofan Engine Using LSTM Neural Networks. In *IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, 2018.

[92] Alessandro Massaro, Giovanni Dipierro, Sergio Selicato, Emanuele Cannella, Angelo Galiano, and Annamaria Saponaro. Intelligent Inspection of Railways Infrastructure and Risks Estimation by Artificial Intelligence Applied on Non-invasive Diagnostic Systems. In *IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0 & IoT)*, 2021.

[93] Fariha Maqbool, Haroon Mahmood, and Hasan Ali Khattak. An Efficient Fault-Prediction Mechanism for Improving Yield in Industry 5.0. In *24th International Multitopic Conference (INMIC)*, 2022.

[94] Weiting Zhang, Dong Yang, Youzhi Xu, Xuefeng Huang, Jun Zhang, and Mikael Gidlund. DeepHealth: A Self-Attention Based Method for Instant Intelligent Predictive Maintenance in Industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, 17(8):5461–5473, 2020.

[95] Christian Gianoglio, Edoardo Ragusa, Paolo Gastaldo, Federico Gallesi, and Francesco Guastavino. Online Predictive Maintenance Monitoring Adopting Convolutional Neural Networks. *Energies*, 14(15):4711, 2021.

[96] Maxim Shcherbakov and Cuong Sai. A Hybrid Deep Learning Framework for Intelligent Predictive Maintenance of Cyber-physical Systems. *ACM Transactions on Cyber-Physical Systems (TCPS)*, 6(2):1–22, 2022.

[97] Timo Huuhtanen and Alexander Jung. Predictive maintenance of photovoltaic panels via deep learning. In *IEEE Data Science Workshop (DSW)*, 2018.

[98] Jing Li, Aidong Deng, Yong Yang, and Qiang Cheng. Fault Diagnosis of Wind Turbine Drive Train using Time-Frequency Estimation and CNN. In *Prognostics and System Health Management Conference (PHM-Qingdao)*, 2019.

[99] Ramanpreet Singh Pahwa, Vijay Ramaseshan Chandrasekhar, Jin Chao, Jestine Paul, Yiqun Li, Ma Tin Lay Nwe, Shudong Xie, Ashish James, Arulmurugan Ambikapathi, and Zeng Zeng. FaultNet: Faulty Rail-Valves Detection using Deep Learning and Computer Vision. In *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.

[100] Jiaxing Wang, Dazhi Wang, and Xinghua Wang. Fault Diagnosis of Industrial Robots Based on Multi-sensor Information Fusion and 1D Convolutional Neural Network. In *39th Chinese Control Conference (CCC)*, 2020.

[101] Hyunho Mo, Federico Lucca, Jonni Malacarne, and Giovanni Iacca. Multi-Head CNN-LSTM with Prediction Error Analysis for Remaining Useful Life Prediction. In *27th Conference of Open Innovations Association (FRUCT)*, 2020.

[102] Huanjie Wang, Xiwei Bai, and Jie Tan. Uncertainty Quantification of Bearing Remaining Useful Life Based on Convolutional Neural Network. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020.

[103] Chang Woo Hong, Min-Seung Ko, and Kyeon Hur. ConvNet-based Remaining Useful Life Prognosis of a Turbofan Engine. In *4th International Conference on Knowledge Innovation and Invention (ICKII)*, 2021.

[104] Priscile Fogou Suawa and Michael Hubner. Health Monitoring of Milling Tools under Distinct Operating Conditions by a Deep Convolutional Neural Network model. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

[105] Andrea Borghesi, Alessio Burrello, and Andrea Bartolini. ExaMon-X: A Predictive Maintenance Framework for Automatic Monitoring in Industrial IoT Systems. *IEEE Internet of Things Journal*, 10(4):2995–3005, 2021.

[106] Mahi Ayman, Mariam Othman, Nour Mahmoud, Zeina Tamer, Maha Sayed, and Yomna MI Hassan. Fault Detection in Wind Turbines using Deep Learning. In *2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, 2022.

[107] Sofia Giannakidou, Panagiotis Radoglou-Grammatikis, Sotirios Koussouris, Minas Pertselakis, Nikolaos Kanakaris, Alexios Lekidis, Konstantinos Kaltakis, Maria P. Koidou, Chrysi Metallidou, Konstantinos E. Psannis, Sotirios Goudos, and Panagiotis Sarigiannidis. 5G-Enabled NetApp for Predictive Maintenance in Critical Infrastructures. In *5th World Symposium on Communication Engineering (WSCE)*, 2022.

[108] Izaz Raouf, Prashant Kumar, Yubin Cheon, Mohad Tanveer, Soo-Ho Jo, and Heung Soo Kim. Advances in Prognostics and Health Management for Aircraft Landing Gear—Progress, Challenges, and Future Possibilities. *International Journal of Precision Engineering and Manufacturing-Green Technology*, pages 1–20, 2024.

[109] Izaak Stanton, Kamran Munir, Ahsan Ikram, and Murad El-Bakry. Predictive maintenance analytics and implementation for aircraft: Challenges and opportunities. *Systems Engineering*, 26(2):216–237, 2022.

[110] Francesco Biondani, Nicola Dall'Ora, Francesco Tosoni, Francesco Tosoni, and Franco Fummi. Fault Injection for Synthetic Data Generation in Aircraft: A Simulation-Based Approach. In *22nd International Conference on Industrial Informatics (INDIN)*, 2024.

[111] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2022.

[112] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[113] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit Optimizers via Block-wise Quantization. In *International Conference on Learning Representations (ICLR)*, 2022.

[114] Minghui Yang, Peng Wu, and Hui Feng. MemSeg: A semi-supervised method for image surface defect detection using differences and commonalities. *Engineering Applications of Artificial Intelligence*, 119:105835, 2023.

[115] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *International Conference on Machine Learning (ICML)*, 2015.

[116] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[117] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations (ICLR)*, 2020.

[118] Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[119] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[120] Luigi Capogrosso, Alessio Mascolini, Federico Girella, Geri Skenderi, Sebastiano Gaiardelli, Nicola Dall'Ora, Francesco Ponzio, Enrico Fraccaroli, Santa Di Cataldo, Sara Vinco, Enrico Macii, Franco Fummi, and Marco Cristani.

Neuro-Symbolic Empowered Denoising Diffusion Probabilistic Models for Real-Time Anomaly Detection in Industry 4.0: Wild-and-Crazy-Idea Paper. In *2023 Forum on Specification & Design Languages (FDL)*, 2023.

[121] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[122] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents. *arXiv preprint arXiv:2204.06125*, 2022.

[123] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[124] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[125] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers. Accessed: 2024-11-03.

[126] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[127] Mehdi Mirza. Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784*, 2014.

[128] Yin-Wong Cheung and Kon S Lai. Lag Order and Critical Values of the Augmented Dickey-Fuller Test. *Journal of Business & Economic Statistics*, 13(3):277, 1995.

[129] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[130] Evelyn Fix and J. L. Hodges. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238, 1989.

[131] Majid Jamil, Sanjeev Kumar Sharma, and Rajveer Singh. Fault detection and classification in electrical power transmission system using artificial neural network. *SpringerPlus*, 4(1):1–13, 2015.

[132] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the Language of Time Series. *Transactions on Machine Learning Research (TMLR)*, 2024.

[133] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR Time Series Archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[134] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *Information Processing in Medical Imaging*, 2017.

[135] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. Adversarially learned anomaly detection. In *IEEE International Conference on Data Mining (ICDM)*, 2018.

[136] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep One-Class Classification. In *35th International Conference on Machine Learning (ICML)*, 2018.

[137] Bin Zhou, Shenghua Liu, Bryan Hooi, Xueqi Cheng, and Jing Ye. BeatGAN: Anomalous Rhythm Detection using Adversarially Generated Time Series. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[138] Liron Bergman and Yedid Hoshen. Classification-Based Anomaly Detection for General Data. *arXiv preprint arXiv:2005.02359*, 2020.

[139] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In *26th ACM/SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2020.

[140] Zhuangwei Shi. Incorporating Transformer and LSTM to Kalman Filter with EM algorithm for state estimation. *arXiv preprint arXiv:2105.00250*, 2021.

[141] Zekai Chen, Dingshuo Chen, Xiao Zhang, Zixuan Yuan, and Xiuzhen Cheng. Learning Graph Structures With Transformer for Multivariate Time-Series Anomaly Detection in IoT. *IEEE Internet of Things Journal*, 9(12):9179–9189, 2022.

[142] Cheng Feng and Pengwei Tian. Time Series Anomaly Detection for Cyber-physical Systems via Neural System Identification and Bayesian Filtering. In *27th ACM/SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2021.

[143] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H. Chen. ECOD: Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12181–12193, 2023.

[144] Xunhua Huang, Fengbin Zhang, Ruidong Wang, Xiaohui Lin, Han Liu, and Haoyi Fan. Kalmanae: Deep embedding optimized kalman filter for time series anomaly detection. *IEEE Transactions on Instrumentation and Measurement*, 72:1–11, 2023.

[145] W. A. Smith, R. B. Randall, and et al. Rolling Element Bearing Fault Diagnosis Using Vibration Signals. Case Western Reserve University Bearing Data Center Website, 1997. Accessed: 2024-11-26.

[146] G.B. Moody and R.G. Mark. The impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[147] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A dataset to support research in the design of secure water treatment systems. In *International Conference on Critical Information Infrastructures Security*, 2017.

[148] Arthur Zimek, Matthew Gaudet, Ricardo J.G.B. Campello, and Jörg Sander. Subsampling for Efficient and Effective Unsupervised Outlier Detection Ensembles. In *19th ACM/SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2013.

[149] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews*, 29(5–6):594–621, 2010.

[150] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, 2021.

[151] Geri Skenderi, Christian Joppi, Matteo Denitto, and Marco Cristani. Well Googled is Half Done: Multimodal Forecasting of New Fashion Product Sales with Image-based Google Trends. *Journal of Forecasting*, 43(6):1982–1997, 2024.

[152] Lequan Lin, Zhengkun Li, Ruikun Li, Xuliang Li, and Junbin Gao. Diffusion models for time-series applications: a survey. *Frontiers of Information Technology & Electronic Engineering*, 25(1):19–41, 2023.

[153] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion Models: A Comprehensive Survey of Methods and Applications. *ACM Computing Surveys*, 56(4):1–39, 2023.

[154] Shuyun Ren, Hau-Ling Chan, and Pratibha Ram. A Comparative Study on Fashion Demand Forecasting Models with Multiple Sources of Uncertainty. *Annals of Operations Research*, 257(1–2):335–355, 2016.

[155] Giuseppe Craparotta, Sébastien Thomassey, and Amedeo Biolatti. A siamese neural network application for sales forecasting of new fashion products using heterogeneous data. *International Journal of Computational Intelligence Systems*, 12(2):1537, 2019.

[156] Pawan Kumar Singh, Yadunath Gupta, Nilpa Jha, and Aruna Rajan. Fashion Retail: Forecasting Demand for New Items. *arXiv preprint arXiv:1907.01960*, 2019.

[157] Vijay Ekambaram, Kushagra Manglik, Sumanta Mukherjee, Surya Shravan Kumar Sajja, Satyam Dwivedi, and Vikas Raykar. Attention based Multi-Modal New Product Sales Time-series Forecasting. In *26th ACM/SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2020.

[158] Geri Skenderi, Christian Joppi, Matteo Denitto, Berniero Scarpa, and Marco Cristani. The multi-modal universe of fast-fashion: the Visuelle 2.0 benchmark. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022.

[159] Ling Carlos, García, Elizabeth, FridaRim, and Ferrando Jaime. HM Personalized Fashion Recommendations. https://kaggle.com/competitions/h-and-m-personalized-fashion-recommendations. Accessed: 2024-09-16.

[160] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[161] Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), 2001.

[162] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition. In *AAAI Conference on Artificial Intelligence*, 2018.

[163] Ryotaro Shimizu, Yuki Saito, Megumi Matsutani, and Masayuki Goto. Fashion intelligence system: An outfit interpretation utilizing images and rich abstract tags. *Expert Systems with Applications*, 213:119167, 2023.

[164] Fei Tao, Bin Xiao, Qinglin Qi, Jiangfeng Cheng, and Ping Ji. Digital twin modeling. *Journal of Manufacturing Systems*, 64:372–389, 2022.

[165] Xianming Huang. Intelligent remote monitoring and manufacturing system of production line based on industrial internet of things. *Computer Communications*, 150:421–428, 2020.

[166] Chong Tang, Wenda Li, Shelly Vishwakarma, Kevin Chetty, Simon Julier, and Karl Woodbridge. Occupancy Detection and People Counting Using WiFi Passive Radar. In *IEEE Radar Conference (RadarConf20)*, 2020.

[167] Imran Ullah Khan, Sitara Afzal, and Jong Weon Lee. Human activity recognition via hybrid deep learning based model. *Sensors*, 22(1):323, 2022.

[168] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. *Sustainable Cities and Society*, 38:697–713, 2018.

[169] Michele Boldo, Nicola Bombieri, Stefano Centomo, Mirco De Marchi, Florenc Demrozi, Graziano Pravadelli, Davide Quaglia, and Cristian Turetta. Integrating Wearable and Camera Based Monitoring in the Digital Twin for Safety Assessment in the Industry 4.0 Era. In *11th International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2022.

[170] Florenc Demrozi, Graziano Pravadelli, Azra Bihorac, and Parisa Rashidi. Human activity recognition using inertial, physiological and environmental sensors: A comprehensive survey. *IEEE Access*, 8:210816–210836, 2020.

[171] Han Zou, Yuxun Zhou, Jianfei Yang, and Costas J. Spanos. Device-free occupancy detection and crowd counting in smart buildings with WiFi-enabled IoT. *Energy and Buildings*, 174:309–322, 2018.

[172] Zhu Wang, Bin Guo, Zhiwen Yu, and Xingshe Zhou. Wi-Fi CSI-Based Behavior Recognition: From Signals and Actions to Activities. *IEEE Communications Magazine*, 56(5):109–115, 2018.

[173] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[174] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[175] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-Feasible Trajectory Forecasting with Heterogeneous Data. In *European Conference on Computer Vision (ECCV)*, 2020.

[176] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Transformer Networks for Trajectory Forecasting. In *25th International Conference on Pattern Recognition (ICPR)*, 2021.

[177] Tianpei Gu, Guangyi Chen, Junlong Li, Chunze Lin, Yongming Rao, Jie Zhou, and Jiwen Lu. Stochastic Trajectory Prediction via Motion Indeterminacy Diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[178] Chenxin Xu, Weibo Mao, Wenjun Zhang, and Siheng Chen. Remember Intentions: Retrospective-Memory-based Trajectory Prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[179] Inhwan Bae, Jin-Hwi Park, and Hae-Gon Jeon. Learning Pedestrian Group Representations for Multi-modal Trajectory Prediction. In *European Conference on Computer Vision (ECCV)*, 2022.

[180] Pranav Mantini and Shishir K. Shah. Human Trajectory Forecasting In Indoor Environments Using Geometric Context. In *Indian Conference on Computer Vision Graphics and Image Processing (ICVGIP)*, 2014.

[181] Andrey Rudenko, Tomasz P Kucner, Chittaranjan S Swaminathan, Ravi T Chadalavada, Kai O Arras, and Achim J Lilienthal. THÖR: Human-Robot Navigation Data Collection and Accurate Motion Trajectories Dataset. *IEEE Robotics and Automation Letters*, 5(2):676–682, 2020.

[182] Luca Rossi, Marina Paolanti, Roberto Pierdicca, and Emanuele Frontoni. Human trajectory prediction and generation using LSTM models and GANs. *Pattern Recognition*, 120:108136, 2021.

[183] Peng Wang, Jing Yang, and Jianpei Zhang. Location Prediction for Indoor Spaces based on Trajectory Similarity. In *4th International Conference on Data Science and Information Technology (DSIT)*, 2021.

[184] Geri Skenderi, Alessia Bozzini, Luigi Capogrosso, Enrico Carlo Agrillo, Giovanni Perbellini, Franco Fummi, and Marco Cristani. DOHMO: Embedded Computer Vision in Co-Housing Scenarios. In *Forum on specification & Design Languages (FDL)*, 2021.

[185] Peng Wang, Jing Yang, and Jianpei Zhang. Indoor Trajectory Prediction for Shopping Mall via Sequential Similarity. *Information*, 13(3):158, 2022.

[186] Andrea Toaiari, Federico Cunico, Francesco Taioli, Ariel Caputo, Gloria Menegaz, Andrea Giachetti, Giovanni Maria Farinella, and Marco Cristani. SCENE-pathy: Capturing the Visual Selective Attention of People Towards Scene Elements. In *22nd International Conference on Image Analysis and Processing (ICIAP)*, 2023.

[187] Jiewu Leng, Dewen Wang, Weiming Shen, Xinyu Li, Qiang Liu, and Xin Chen. Digital twins-based smart manufacturing system design in Industry 4.0: A review. *Journal of Manufacturing Systems*, 60:119–137, 2021.

[188] Junlang Guo, Jiewu Leng, J. Leon Zhao, Xueliang Zhou, Yu Yuan, Yuqian Lu, Dimitris Mourtzis, Qinglin Qi, Sihan Huang, Xueguan Song, Qiang Liu, and Lihui Wang. Industrial metaverse towards Industry 5.0: Connotation, architecture, enablers, and challenges. *Journal of Manufacturing Systems*, 76:25–42, 2024.

[189] Shiying Zhang, Jun Li, Long Shi, Ming Ding, Dinh C Nguyen, Wen Chen, and Zhu Han. Industrial Metaverse: Enabling Technologies, Open Problems, and Future Trends. *arXiv preprint arXiv:2405.08542*, 2024.

[190] Epic Games. Unreal Engine. https://www.unrealengine.com. Accessed: 2024-09-16.

[191] Sebastiano Gaiardelli, Stefano Spellini, Michele Lora, and Franco Fummi. Modeling in Industry 5.0: What Is There and What Is Missing: Special Session 1: Languages for Industry 5.0. In *Forum on specification & Design Languages (FDL)*, 2021.

[192] Samad M. E. Sepasgozar. Differentiating Digital Twin from Digital Shadow: Elucidating a Paradigm Shift to Expedite a Smart, Sustainable Built Environment. *Buildings*, 11(4):151, 2021.

[193] Eric VanDerHorn and Sankaran Mahadevan. Digital Twin: Generalization, characterization and implementation. *Decision Support Systems*, 145:113524, 2021.

[194] National Academies of Sciences, Engineering, and Medicine and others. *Foundational Research Gaps and Future Directions for Digital Twins*. National Academies Press, 2024.

[195] Jiewu Leng, Xiaofeng Zhu, Zhiqiang Huang, Xingyu Li, Pai Zheng, Xueliang Zhou, Dimitris Mourtzis, Baicun Wang, Qinglin Qi, Haidong Shao, Jiafu Wan, Xin Chen, Lihui Wang, and Qiang Liu. Unlocking the power of industrial artificial intelligence towards Industry 5.0: Insights, pathways, and challenges. *Journal of Manufacturing Systems*, 73:349–363, 2024.

[196] Jay Lee, Moslem Azamfar, Jaskaran Singh, and Shahin Siahpour. Integration of digital twin and deep learning in cyber-physical systems: towards smart manufacturing. *IET Collaborative Intelligent Manufacturing*, 2(1):34–36, 2020.

[197] Zhiming Zheng, Tan Li, Bohu Li, Xudong Chai, Weining Song, Nanjiang Chen, Yuqi Zhou, Yanwen Lin, and Runqiang Li. Industrial Metaverse: Connotation, Features, Technologies, Applications and Challenges. In *Methods and Applications for Modeling and Simulation of Complex Systems*, 2022.

[198] Microsoft. Azure Digital Twins. https://azure.microsoft.com/en-us/products/digital-twins. Accessed: 2024-09-16.

[199] Siemens. Plant Simulation. https://plm.sw.siemens.com/en-US/tecnomatix/products/plant-simulation-software/. Accessed: 2024-09-16.

[200] NVIDIA. Omniverse Platform. https://www.nvidia.com/en-us/omniverse/. Accessed: 2024-09-16.

[201] Epic Games. Live Link Plugin in Unreal Engine. https://dev.epicgames.com/documentation/en-us/unreal-engine/live-link-in-unreal-engine. Accessed: 2024-09-16.

[202] Rokoko. Rokoko Studio. https://www.rokoko.com/products/smartsuit-pro. Accessed: 2024-09-16.

[203] Xsens Technologies BV. Xsens 3D Motion Tracking. https://www.movella.com/products/motion-capture. Accessed: 2024-09-16.

[204] NeuronMocap. Perception Neuron Motion Capture. https://neuronmocap.com/?srsltid=AfmBOoq2izy4qXuBwWm_IipFgEjfqTYGkc96DdwMa2TCI4AYjB9pqjkk. Accessed: 2024-09-16.

[205] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[206] Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2018.

[207] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) Equivariant Graph Neural Networks. In *International Conference on Machine Learning (ICML)*, 2021.

[208] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT Pre-Training of Image Transformers. In *International Conference on Learning Representations (ICLR)*, 2021.

[209] Chenxin Xu, Robby T. Tan, Yuhong Tan, Siheng Chen, Yu Guang Wang, Xinchao Wang, and Yanfeng Wang. EqMotion: Equivariant Multi-Agent Motion Prediction with Invariant Interaction Reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[210] Wenbing Huang, Jiaqi Han, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Equivariant Graph Mechanics Networks with Constraints. In *International Conference on Learning Representations (ICLR)*, 2021.

[211] Patrizia Gabellini, Mauro D'Aloisio, Matteo Fabiani, and Valerio Placidi. A Large Scale Trajectory Dataset for Shopper Behaviour Understanding. In *New Trends in Image Analysis and Processing – 22nd International Conference on Image Analysis and Processing (ICIAP)*, 2019.

[212] Stefano Pellegrini, Andreas Ess, and Luc Van Gool. Improving Data Association by Joint Modeling of Pedestrian Trajectories and Groupings. In *European Conference on Computer Vision (ECCV)*, 2010.

[213] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by Example. *Computer Graphics Forum*, 26(3):655–664, 2007.

[214] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It Is Not the Journey But the Destination: Endpoint Conditioned Trajectory Prediction. In *European Conference on Computer Vision (ECCV)*, 2020.

[215] Yingfan Huang, Huikun Bi, Zhaoxin Li, Tianlu Mao, and Zhaoqi Wang. STGAT: Modeling Spatial-Temporal Interactions for Human Trajectory Prediction. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[216] Cunjun Yu, Xiao Ma, Jiawei Ren, Haiyu Zhao, and Shuai Yi. Spatio-Temporal Graph Transformer Networks for Pedestrian Trajectory Prediction. In *European Conference on Computer Vision (ECCV)*, 2020.

[217] Yue Hu, Siheng Chen, Ya Zhang, and Xiao Gu. Collaborative Motion Prediction via Neural Motion Message Passing. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[218] Ye Yuan, Xinshuo Weng, Yanglan Ou, and Kris Kitani. AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[219] Chenxin Xu, Maosen Li, Zhenyang Ni, Ya Zhang, and Siheng Chen. GroupNet: Multiscale Hypergraph Neural Networks for Trajectory Prediction with Relational Reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[220] Rejin Varghese and Sambath M. YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. In *International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 2024.