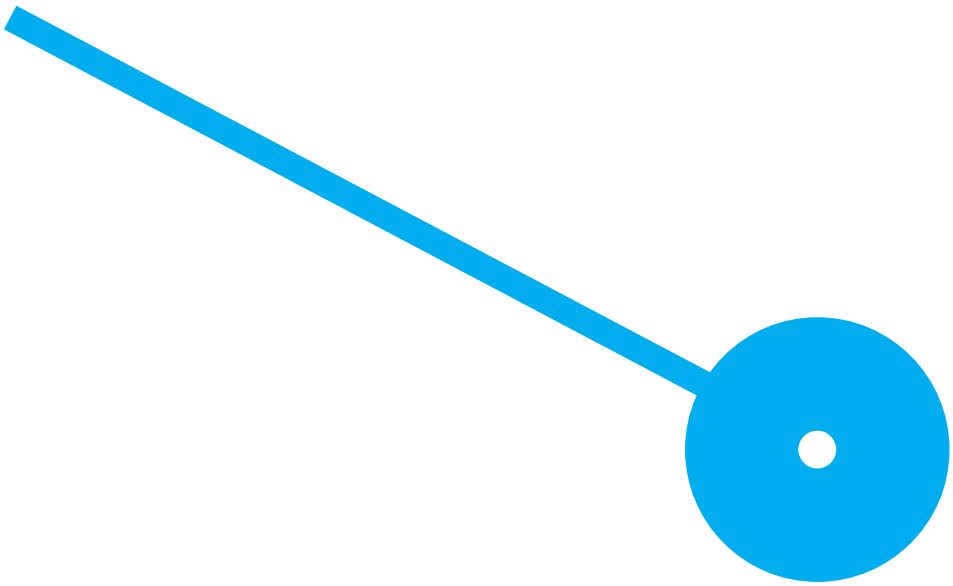
SCHOOL OF
MANAGEMENT AND
TECHNOLOGY
POLYTECHNIC OF
PORTO

M MASTER
COMPUTER SCIENCE

# Real-Time Data Analysis Tool for Decision Support in Streaming Environments

Diogo Pinto Torres

11/25



SCHOOL OF
MANAGEMENT AND
TECNOLOGY
POLYTECHNIC OF
PORTO

MASTER
COMPUTER SCIENCE

# Real-Time Data Analysis Tool for Decision Support in Streaming Environments

Diogo Pinto Torres
8200836

## Advisor

Dr. Davide Rua Carneiro

Dissertation submitted in fulfilment of the requirements for the Master's degree in Computer Science in the School of Management and Technology of the Polytechnic of Porto.

11/25

# Declaration of Integrity

I, Diogo Pinto Torres, student nº 8200836, enrolled in the Master's in Computer Science at School of Management and Technology of the Polytechnic of Porto, declare that I have neither committed plagiarism nor self-plagiarism. Therefore, the work entitled "Real-Time Data Analysis Tool for Decision Support in Streaming Environments" is original and my own, and has not been previously used for any other purpose. I further declare that all sources used are cited in the text and final bibliography according to the referencing rules adopted by the institution.

# Acknowledgements

This work has been supported by the European Union under the Next Generation EU, through a grant of the Portuguese Republic's Recovery and Resilience Plan (PRR) Partnership Agreement, within the scope of the project PRODUTECH R3 – "Agenda Mobilizadora da Fileira das Tecnologias de Produção para a Reindustrialização", Total project investment: 166.988.013,71 Euros; Total Grant: 97.111.730,27 Euros.

# Abstract

The growing reliance on artificial intelligence (AI) in industrial environments highlights the need for tools that support rapid prototyping, evaluation, and deployment of machine learning models under streaming conditions. While current platforms for data science and stream processing provide partial solutions, they often fall short when applied to dynamic industrial scenarios that require low-latency responses, continuous adaptation, and transparent monitoring.

This work introduces *MultiFlow*, a flexible and containerized tool designed to simulate and analyze real-time industrial data streams. MultiFlow provides a functional environment where users can organize projects, configure data streams, develop and deploy processing applications, and monitor results through live dashboards. Its architecture combines industry-standard technologies such as Kafka, Faust, InfluxDB, and Grafana to bridge the gap between research-driven AI models and production-oriented streaming applications.

The tool is validated through various use cases, including: reusing previously trained models in new machine learning tasks, anomaly detection using ensembles of Isolation Forests, and handling concept drift in evolving data. These scenarios demonstrate MultiFlow's ability to support flexible experimentation, facilitate comparison of configurations through instances, and deliver real-time insights in industrial contexts.

By offering a digital twin environment for continuous evaluation of streaming AI workflows, MultiFlow contributes both as a research sandbox and as a practical tool for industry partners. The findings highlight its potential to accelerate the development of adaptive AI systems while reducing resource costs and improving decision support in Industry.

**Keywords:** Digital Twins, Data Streams, Real-Time Analytics, Prototyping, Industrial Applications, Machine Learning

# Resumo

A crescente dependência da inteligência artificial (IA) em ambientes industriais destaca a necessidade de ferramentas que apoiem prototipagem rápida, avaliação e implementação de modelos de *Machine Learning* (ML) em ambientes de *streaming*. Embora as plataformas atuais de *data science* e processamento de *streams* ofereçam soluções parciais, estas revelam-se frequentemente insuficientes quando aplicadas em cenários industriais dinâmicos, que exijam respostas de baixa latência, adaptação contínua e monitorização transparente.

Este trabalho apresenta a *MultiFlow*, uma ferramenta flexível e conteinerizada, concebida para simular e analisar fluxos de dados industriais em tempo real. *MultiFlow* fornece um ambiente funcional onde os utilizadores podem organizar projetos, configurar *streams*, desenvolver e executar aplicações de processamento, e monitorar resultados através de *dashboards* em tempo real. A sua arquitetura combina tecnologias amplamente adotadas na indústria, como Kafka, Faust, InfluxDB e Grafana, de modo a diminuir a margem entre os modelos de IA baseados na investigação e de aplicações de *streaming* orientadas para a produção.

A ferramenta é validada através de multiplos casos de uso, entre eles: a reutilização de modelos previamente treinados em novas tarefas de ML, a deteção de anomalias recorrendo a *ensembles* de *Isolation Forests*, e a deteção de *concept drift* em dados que estão em constante evolução. Estes cenários demonstram a capacidade de MultiFlow em suportar experiências de forma flexível, facilitar a comparação de configurações através de instâncias, e fornecer *insights* em tempo real em contextos industriais.

Ao disponibilizar um ambiente de *Digital Twins* para a avaliação contínua de *workflows* de IA em *streaming*, *MultiFlow* contribui tanto como um espaço de experimentação para investigação, como uma ferramenta prática para parceiros industriais. Os resultados destacam o potencial da mesma para acelerar o desenvolvimento de sistemas de IA adaptativos, ao mesmo tempo que reduz custos em termos de recursos e melhora o apoio à decisão na Indústria.

**Palavras-chaves:** Digital Twins, Streams de Dados, Análise em Tempo-Real, Prototipagem, Aplicações Industriais, Machine Learning

# Contents

# List of Figures

# List of Tables

# List of Listings

# Acronyms

**AI** Artificial Intelligence. 1, 13, 15–19

**CD** Continuous Delivery. 17

**CI** Continuous Integration. 17

**DSR** Design Science Research. 7, 8

**IoT** Internet of Things. 9, 11, 12, 17

**M2M** Machine to Machine. 11

**MES** Manufacturing Execution System. 4, 18

**ML** Machine Learning. 1, 9, 13, 14, 16, 17, 30

**MLOps** Machine Learning Operations. 4, 14, 16, 17, 46–48

**MMD** Maximum Mean Discrepancy. 31, 32

**RAMI 4.0** Reference Architectural Model for Industry 4.0. 18

**XAI** explainable AI. 18

# Chapter 1

# Introduction

In recent years, the digital transformation of industries has significantly accelerated [1, 2], driven by the increasing availability of data and the growing capabilities of Artificial Intelligence (AI) and Machine Learning (ML) techniques [3, 4]. This shift is largely framed within the paradigm of Industry 4.0, where interconnected systems, intelligent automation, and real-time analytics are reshaping how decisions are made in production environments [5, 4, 3, 2]. At the core of this transformation lies the need to extract timely and actionable insights from vast amounts of data generated continuously by sensors, machines, and systems [6, 7].

Unlike traditional batch processing methods, many modern industrial applications require handling data in motion [8, 9]. Data streaming enables continuous ingestion, processing, and analysis of time-sensitive information [10, 11, 12], which is essential in scenarios such as predictive maintenance, anomaly detection, process optimization, or quality control [5, 13, 14, 15, 16, 17]. However, leveraging streaming data effectively introduces several challenges, from managing high-throughput pipelines to designing models capable of learning from dynamic environments, often with limited time and computing resources [18, 9, 19, 12].

The integration of AI into these environments has shown a big potential [4, 2, 3], but it also reveals critical gaps in tooling and flexibility [19, 5]. Some existing solutions tend to be tailored to specific domains, can be difficult to customize, or may not fully support real-time streaming contexts.

For instance, tools such as Apache NiFi provide dataflow orchestration capabilities, but they can be complex to adapt for workflows that involve frequent testing and reconfiguration of machine learning algorithms [20, 21, 22]. Platforms like RapidMiner or KNIME offer accessible visual interfaces for building data pipelines, yet they are primarily oriented toward batch processing and general data science tasks, making them less suitable for continuous data streams or evolving sensor data [23]. Similarly, frameworks such as Apache Flink and Spark Structured Streaming deliver scalable and fault-tolerant stream processing, but they often require additional engineering effort to couple with real-time AI prototyping and evaluation [24, 25, 5, 11, 12]. Even tools such as MLflow, which are widely adopted for tracking and deploying models, do not natively support real-time simulation of model behavior prior to deployment [26].

To address these challenges, this dissertation presents *MultiFlow*, a flexible and extensible

tool for building, configuring, and executing real-time data analysis workflows. *MultiFlow* functions as a form of Digital Twin, enabling the live representation and monitoring of industrial processes through continuous data streams [2]. Users can define multiple applications, each composed of a customizable sequence of data processing, machine learning, and decision-making components, that run on real-time data. By simulating and analyzing operational behavior under different configurations, *MultiFlow* supports timely, data-driven decision-making and provides a practical environment for testing anomaly detection methods, concept drift strategies, and other AI-driven techniques. The tool thus aims to bridge the gap between experimentation and deployment in real-world industrial scenarios. Figure 1.1 illustrates this iterative process, where users test, deploy, and continuously refine machine learning models or algorithms using *Multiflow* as a digital twin environment.

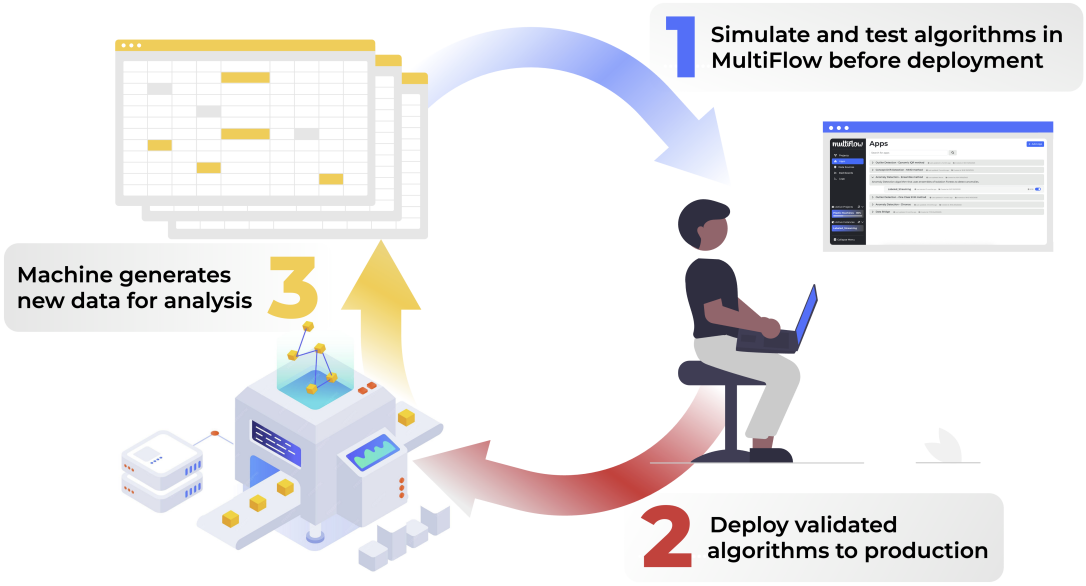



Figure 1.1: *Multiflow* as a digital twin supporting the machine learning development lifecycle: (1) algorithms are tested and validated using simulated real-time streams; (2) validated algorithms are deployed to production environments; (3) the resulting operational data can be reintegrated into *Multiflow* for ongoing monitoring, retraining, or evaluation. (Source: author)

The following sections will define the specific problem that this tool aims to solve, outline the motivation behind its development, and describe the primary objectives and contributions of this work.

## 1.1 Context & Problem Statement

Modern industries are increasingly becoming data-driven, relying heavily on real-time insights to support critical decision-making processes [18, 8, 5, 6, 27, 9]. The widespread adoption of sensors, automation systems, and interconnected devices has led to the continuous generation of large volumes of streaming data [7, 28, 29, 15]. This data holds a lot of potential to improve operational efficiency, detect anomalies, and adapt to changing conditions, but only if it can be processed and analyzed in real time [30, 31, 15].

However, deriving value from streaming data introduces several challenges [5, 13]. Firstly, industrial environments are dynamic and constantly evolving, making it difficult for static models to maintain consistent performance over time [30, 1, 14]. Phenomena such as concept drift, where the underlying data distribution changes, can degrade the effectiveness of machine learning models if not addressed promptly [17, 30, 1]. Similarly, the presence of unexpected anomalies requires models that can adapt quickly without human intervention [16, 15].

Secondly, the velocity of data in streaming applications demands rapid processing and minimal latency [12, 11, 32]. Traditional batch-oriented approaches are often too slow or rigid to respond to time-sensitive events, especially in scenarios where immediate actions are needed to prevent failures or optimize processes [17, 33].

In addition to speed, industrial data is also characterized by its variety and scale [17, 15]. It comes from multiple heterogeneous sources, ranging from low-level sensors to high-level systems, and often requires complex preprocessing before it can be used for model inference or analysis [7, 2]. Managing this complexity efficiently, while maintaining system performance and scalability, is a major obstacle for many real-time AI solutions [31, 30, 19].

Finally, ensuring model robustness both before and during deployment remains a critical issue [17, 10]. Models that perform well in offline environments can struggle once exposed to live, noisy, and unpredictable data streams [6]. There is a need for tools that not only allow flexible experimentation with different model configurations but also facilitate continuous evaluation and adaptation in production-like settings.

These challenges highlight the need for tools that enable the rapid development, deployment, and monitoring of AI-driven applications in real-time industrial environments. Industrial systems require solutions that can handle high-velocity, heterogeneous data streams, adapt to changes in underlying processes, and respond quickly to anomalies or unexpected events. Traditional approaches often struggle to meet these demands, highlighting the need for more specialized tools.

*Multiflow* provides a structured framework to address these challenges. It allows the creation of applications that integrate multiple data sources, simulate streaming scenarios, and evaluate model performance under realistic conditions. By supporting continuous monitoring and adaptive responses, *Multiflow* ensures that deployed applications can maintain robust performance even in dynamic industrial settings.

Although developing applications within *Multiflow* requires technical knowledge, using, configuring, and running these applications is designed to be accessible to industrial practitioners without deep expertise in machine learning or stream processing. This combination of flexibility and operational simplicity makes *Multiflow* particularly well-suited for industry scenarios.

## 1.2 Motivation

As industries evolve toward increasingly automated and intelligent systems, the ability to make decisions based on real-time data has become a core requirement [8, 18, 3, 33]. This shift has been driven by the rise of Industry 4.0 and the explosion of sensor-rich

environments, where the continuous monitoring of machines, processes, and conditions offers unprecedented opportunities for optimization and innovation [7, 5].

Despite these advancements, many organizations still face major barriers when trying to adopt real-time AI-based solutions [19]. The available tools are often fragmented, complex, or tailored to very specific use cases, making experimentation and adaptation difficult, especially for non-expert users [4]. There is a clear gap between the theoretical capabilities of AI in streaming environments and the practicality of deploying those solutions in real-world settings [2, 3].

A concrete example of these challenges arises from Muvu Technologies[1], an industry partner and collaborator in this project. Muvu develops and commercializes a Manufacturing Execution System (MES) that is deployed across a variety of client-owned production lines. These clients are the ones who generate data from their operations, including sensor readings, process metrics, and production outcomes, which Muvu can leverage to deliver AI-powered functionalities as part of their services.

However, maintaining and scaling these AI capabilities across different clients poses several challenges [2, 26]. Since each production line can involve different machines, products, or workflows, machine learning models often need to be tailored to specific use cases. As a result, generalization is difficult, and models frequently require retraining, particularly when new types of defects or operational behaviors emerge [10]. Furthermore, validating these models in a way that reliably simulates real-time dynamics, without affecting live systems, is a complex and resource-intensive task.

Muvu's broader objective is to improve the management of their Machine Learning Operations (MLOps) pipelines to support faster, more robust, and more scalable delivery of AI-based services. This includes streamlining the processes of testing, deploying, and monitoring models in varied and ever-changing industrial environments [28].

From a research and engineering standpoint, there is also a strong motivation to better understand how to build, test, and deploy intelligent systems that can adapt to live data, detect anomalies, handle concept drift, and maintain performance over time. These are not trivial challenges, and solving them requires not just better models, but better tools that support iterative and configurable workflows [31, 3].

This project was born from the recognition of that gap. By developing *Multiflow*, the goal is to create a tool that allows users—researchers, data scientists, or engineers, to simulate and evaluate data analysis workflows in a way that reflects the dynamic nature of real-world industrial processes. It is motivated by the need for transparency, flexibility, and performance in a domain where mistakes can be costly and opportunities are time-sensitive [1, 3, 14].

Although Muvu Technologies serves as a primary validation partner and use case, the challenges they face are broadly applicable to many other organizations operating in data-rich, high-stakes environments. The tool is designed with generalizability in mind, aiming to provide a reusable and scalable framework for any industrial player seeking to bring real-time AI and digital twin capabilities into their decision-making processes.

Ultimately, the motivation lies in enabling faster, more reliable, and more scalable decision-making in industrial settings through a tool that brings together real-time data handling,

[1] https://muvu.tech

machine learning, and digital twin concepts into a unified framework.

## 1.3 Objectives

The main goal of this dissertation is to design and develop a tool capable of supporting real-time data analysis workflows in streaming environments, with a particular focus on industrial use cases.

To achieve this, the following specific objectives have been defined:

- Develop a system capable of ingesting, processing, and analyzing real-time data streams;
- Allow users to define and manage multiple applications, each composed of configurable sequences of processing and decision-making components;
- Support the integration and evaluation of machine learning models, particularly in the context of anomaly detection and concept drift handling;
- Enable the monitoring and validation of model performance during live execution;
- Design the tool to act as a Digital Twin, reflecting the behavior of real-world industrial processes for testing and decision support;
- Ensure the system is extensible, user-friendly, and suitable for experimentation and prototyping in industrial or dynamic environments.

These objectives aim to bridge the gap between research-oriented experimentation and practical, deployable solutions for AI-driven decision support in industrial streaming contexts.

## 1.4 Work Plan

The development of this dissertation project was structured into several key phases, each focusing on a specific aspect of the work. Although some phases occurred sequentially, others may have overlapped as the project evolved. The following plan outlines the organization of the work and the main tasks associated with each stage.

In addition, the following Gantt diagram (Figure 1.2) describes in a visual way the tasks displacement.

### Literature Review

This phase involved the study and analysis of relevant research in the areas of real-time data streaming, machine learning, anomaly detection, concept drift, and Industry 4.0. While the formal time allocated to this phase was one month, the literature review continued in parallel with other phases as needed.

This initial phase consisted of the following tasks:

- Research and analyze state-of-the-art tools and techniques;
- Understand the challenges in real-time AI for industrial environments;

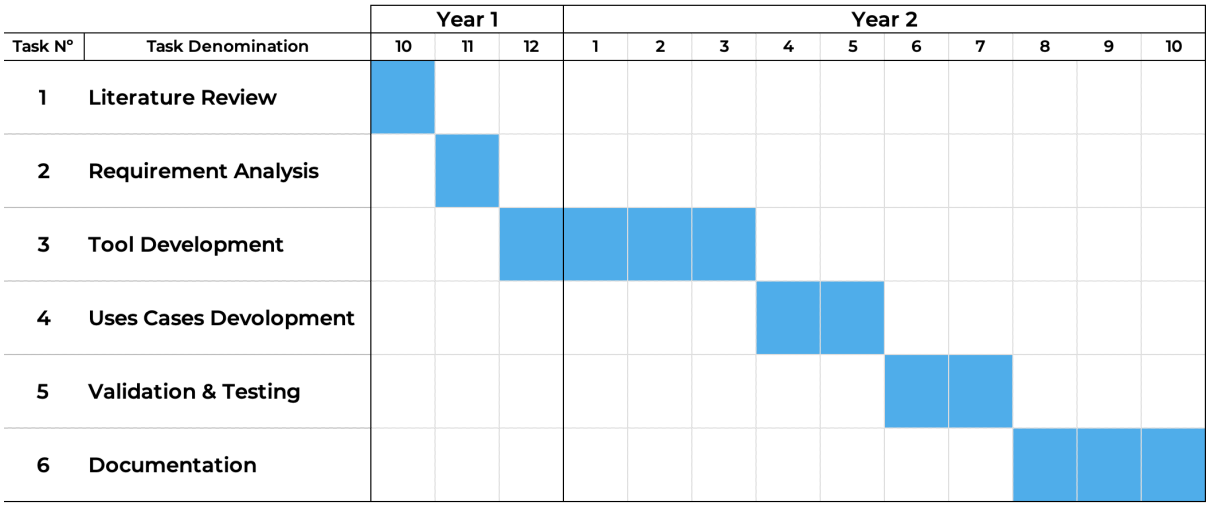



Figure 1.2: Gantt Diagram of the Work Plan. (Source: author)

- Explore Digital Twin frameworks and streaming platforms.

## Requirement Analysis

In this phase, the functional and non-functional requirements of the *Multiflow* tool were defined. This included understanding user needs, technical constraints, and project scope.

Some of the tasks here were as follows:

- Define system architecture and user roles;
- Specify required features and workflows;
- Identify potential technologies, libraries, and platforms.

## Tool Development

This was the core implementation phase, where the *Multiflow* tool was developed. Although this phase was estimated to span around 3 to 4 months, it was expected that development would continue iteratively with updates, improvements, and added features.

This phase involved many different tasks (consisting in the most extensive one), some of them being:

- Develop core components for data ingestion, processing, and pipeline execution;
- Implement support for multiple apps and configurable workflows.

## Use Case Development

To demonstrate the capabilities of the tool, several example applications ("Apps") were developed based on real-world industrial scenarios. These use cases were proposed by the project's industrial partner, Muvu Technologies, and reflect recurring challenges faced in production environments, such as data drift and fault detection.

This phase was crucial in showcasing how *Multiflow* can be used to simulate streaming conditions and support the design and evaluation of AI-based solutions in a flexible and controlled setup.

The tasks involved in this phase included:

- Designing and implementing applications for concept drift detection, anomaly detection, and other streaming-related tasks;
- Simulating streaming environments using both real and synthetic data;
- Validating whether *Multiflow* supports the development needs of Muvu's production scenarios.

### Validation and Testing

This phase focused on evaluating the robustness and practical utility of *Multiflow* in realistic usage conditions.**While formal user studies were not conducted**, the emphasis was on validating the system's performance and responsiveness when applied to representative industrial challenges.

The tasks in this phase included:

- Performing functional and integration testing of the tool;
- Evaluating the performance of Apps under real-time constraints;
- Identifying limitations and iteratively improving the system based on technical feedback.

### Documentation

Proper documentation is essential to ensure usability, reproducibility, and future maintenance. This final phase included both technical and user-oriented documentation, as well as community sharing. Thus, this contained the following tasks:

- Write system documentation and usage guides;
- Prepare the dissertation and supporting materials;
- Document use cases and validation results;
- Share the tool's code in open repositories (such as GitHub).

## 1.5 Methodology

This work adopts the Design Science Research (DSR) methodology, which is well-suited for the development of innovative technological artifacts aimed at solving practical problems [34]. DSR is particularly applicable in contexts where the goal is not only to understand a phenomenon but also to build a solution, in this case, a flexible and centralized tool for real-time algorithm prototyping in streaming environments [35].

*Multiflow* was developed in response to the lack of integrated tools that allow machine learning and algorithm experimentation under conditions that simulate real-world data

streams. DSR provides a structured, iterative framework to address this gap through six key phases:

1. **Problem Identification and Motivation:** The first step focuses on recognizing a real-world problem and motivating the need for an innovative artifact. This is addressed in the *Context & Problem Statement*, *Motivation*, and *Objectives* sections, where the challenges of prototyping AI algorithms in streaming contexts are explored, along with the industrial need for modular testable systems.

2. **Definition of Objectives for a Solution:** This step involves formulating the desired characteristics and requirements of the artifact. These are captured in the *Work Plan* chapter, where each phase of the development is outlined with clear goals and intended outcomes.

3. **Design and Development:** Here, the artifact is built according to the objectives previously established. This phase is documented in the *Design and Specification* and *Implementation* chapters. These sections provide both the architectural vision and the technical details behind *Multiflow*, covering everything from data ingestion to real-time processing and user interaction.

4. **Demonstration:** The artifact must be applied in a realistic setting to demonstrate its utility. In this case, the development and execution of *Multiflow* itself directly address the central challenge, the lack of a centralized, extensible tool for simulating and validating AI methods in realistic streaming scenarios. The demonstration is primarily illustrated through the use cases and the Apps developed within the tool.

5. **Evaluation:** This step assesses how well the artifact performs relative to the defined objectives. In this work, evaluation is carried out in the *Validaion* chapter (chapter 5), where *MultiFlow* is applied to several real-time use cases such as anomaly detection, model reuse, and concept drift analysis. These use cases demonstrate the tool's effectiveness in enabling rapid experimentation, monitoring through dashboards, and supporting industrial partners like Muvu Technologies. Rather than formal benchmarking, the evaluation emphasizes practical applicability and versatility in streaming industrial contexts.

6. **Communication:** Finally, the outcomes must be shared with relevant communities to contribute to broader academic and professional knowledge. This step is reflected in the *Documentation and Dissemination* efforts outlined in the *Work Plan*, as well as in the submission of a scientific article presenting *Multiflow* to a peer-reviewed conference. Additionaly, this criteria is directly met by having the *Multiflow* tool published and available in a GitHub repository[2].

By following this methodology, this work ensures that the development of *Multiflow* is both problem-driven and validated, creating a meaningful contribution to the domain of AI experimentation in streaming and industrial environments.

## 1.6 Document Structure

This dissertation is organized into seven chapters, each building upon the previous to present the motivation, development, and validation of the MultiFlow tool. The document

[2]https://github.com/davidecarneiro/multiflow

follows the following structure:

- **Chapter 1 – Introduction:** establishes the context, problem statement, and motivation behind the work. It defines the objectives, outlines the work plan, and explains the methodology adopted, providing a roadmap for the dissertation.
- **Chapter 2 – Technology Landscape:** reviews the key technological domains relevant to this research. It covers concepts such as Big Data, Internet of Things (IoT), ML, and Concept Drift, offering the theoretical foundation required to understand the challenges of real-time data analysis in industrial scenarios.
- **Chapter 3 – Design and Specification:** describes the functional and non-functional requirements of the tool, presents early interface mockups, and details the architectural vision of MultiFlow. This chapter serves as a blueprint for how the tool was conceptualized to balance flexibility for research with industrial applicability.
- **Chapter 4 – Implementation:** provides a comprehensive account of how MultiFlow was developed. It introduces the tool's functional building blocks (Projects, Streams, Apps, and Instances), before detailing the underlying infrastructure layers, from data streaming with Kafka to the backend, orchestration, and visualization components.
- **Chapter 5 – Validation:** demonstrates the practical application of MultiFlow through several use cases, including model reuse, anomaly detection, and concept drift analysis. These experiments highlight the tool's ability to support real-time prototyping, monitoring, and decision-making in realistic industrial conditions.
- **Chapter 6 – Results:** synthesizes the main findings obtained from the development and evaluation of MultiFlow. It reflects on the contributions of the tool, its impact for both academic and industrial contexts, and the dissemination efforts carried out, including open-source release and scientific publications.
- **Chapter 7 – Conclusion:** summarizes the work, discusses its contributions, and highlights limitations and opportunities for future development, closing the dissertation with reflections on the broader relevance of the results.

# Chapter 2

# Technology Landscape

The development of intelligent tools for real-time data analysis in industrial environments requires a solid understanding of several interrelated technological domains [3, 8]. As organizations increasingly rely on AI-driven solutions to process high volumes of data, especially in streaming scenarios, it becomes essential to understand the current landscape of methods, tools, and challenges that shape this field [2, 4, 1].

This chapter presents the technology landscape relevant to the development of *Multiflow*, a tool designed to support real-time decision-making in data-intensive industrial contexts. It provides an overview of the most significant areas that underpin this work.

## 2.1 Big Data

Big Data refers to the vast and complex datasets generated from various sources, including social media, sensors, financial markets, and e-commerce platforms [32, 8, 31]. Traditionally characterized by the "3 Vs" (Volume, Velocity, and Variety) [32], the concept has evolved to encompass additional dimensions [29]: Veracity, Value, Variability, and Visualization.

- **Volume**: Represents the massive quantities of data generated, often in terabytes or petabytes, necessitating scalable storage solutions.
- **Velocity**: Denotes the speed at which data is produced and processed, with real-time data streams requiring rapid ingestion and analysis.
- **Variety**: Pertains to the diverse formats of data, ranging from structured databases to unstructured media like images and videos.
- **Veracity**: Concerns the trustworthiness and quality of data, emphasizing the need for accurate and reliable information.
- **Value**: Focuses on extracting meaningful insights that can inform decision-making and drive business value.
- **Variability**: Addresses the inconsistencies and fluctuations in data flows, which can complicate analysis and interpretation.

- **Visualization**: Involves the representation of data in graphical formats to facilitate understanding and communication of insights.

In industrial contexts, such as those encountered by Muvu Technologies, the principles of Big Data are particularly pertinent. Muvu oversees multiple production lines equipped with sensor-rich environments, generating continuous streams of data. Managing this data effectively is crucial for monitoring machine performance, detecting anomalies, and optimizing processes [14, 3, 22, 12, 11, 18].

One of the significant challenges Muvu faces is the frequent retraining of machine learning models to accommodate new types of defects or changes in production conditions. The dynamic nature of industrial data, characterized by concept drift and evolving patterns, necessitates robust data management and analysis tools [30, 17, 1, 14]. Furthermore, ensuring the veracity and value of data is essential for maintaining the reliability of predictive models and decision-making processes [1, 4, 18].

The development of *MultiFlow* aims to address these challenges by providing a tool that facilitates the simulation and evaluation of data analysis workflows in real-time streaming environments. By leveraging the principles of Big Data, *MultiFlow* seeks to enhance the adaptability and efficiency of AI-driven solutions in industrial settings, not only benefiting Muvu but also offering a framework applicable to similar organizations facing comparable challenges.

## 2.2 Internet of Things

The IoT represents the convergence of physical and digital worlds, where interconnected devices, systems, and sensors work together via the internet to perform tasks that once required human intervention [8, 9, 18]. These devices gather, transmit, and sometimes even analyze data in real-time with minimal human input, enabling smarter operations across various sectors [33, 1, 32].

Numerous organizations have defined IoT in terms of its capabilities. For example, the International Telecommunication Union (ITU-T) describes it as a "global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving information and communication technologies" [36]. This emphasizes its foundational goal: integrating diverse systems and devices to boost productivity, convenience, and decision-making [8]. Despite the increasing level of autonomy, human oversight remains necessary to some extent [15].

IoT has rapidly expanded across a wide range of domains, including healthcare, transportation, agriculture, logistics, and smart cities [6, 37, 38]. Fueled by innovations in wireless technologies, sensor networks, and embedded systems, IoT has enabled unprecedented data collection and device orchestration at scale [7]. Cisco famously estimated that more than 50 billion devices would be connected to the internet by 2020 [38]. Yet, the adoption of IoT still faces critical challenges, particularly concerning security, standardization, data governance, and system scalability [28, 2, 39, 37].

Closely related technologies like the Internet of Everything (IoE), the Web of Things (WoT), and Machine to Machine (M2M) communication have broadened the scope of IoT, incorporating not only devices, but also people, processes, and contextual data [40, 41, 37, 39]. This expanding ecosystem significantly contributes to the generation of massive,

high-velocity, and heterogeneous data streams — core characteristics of Big Data [18, 29, 15, 17].

In industrial contexts, such as the production lines operated by Muvu Technologies, IoT infrastructure plays a critical role. Sensors and devices embedded across machines generate continuous flows of data, which can be leveraged for monitoring, predictive maintenance, and real-time decision-making [14, 13, 5]. However, extracting value from this data requires scalable processing pipelines and intelligent systems capable of adapting to dynamic behaviors [17]. This naturally links IoT with Big Data analytics and machine learning techniques, setting the stage for real-time AI applications like those enabled by *MultiFlow*.

This constant stream of real-time data from IoT systems introduces new processing challenges, which leads us to Data Streaming.

## 2.3 Data Streaming

Data streaming refers to the continuous flow of data generated by various sources in real-time [12, 11]. Unlike traditional batch processing, where data is collected, stored, and processed at intervals, data streaming processes data as it arrives [22, 9, 18]. This is especially valuable in domains where low-latency processing is critical, such as financial markets, healthcare monitoring, and industrial control systems.

The proliferation of IoT devices and the massive data volumes they produce has further amplified the demand for streaming solutions [29]. Combined with the scale and heterogeneity of Big Data, this introduces new challenges and opportunities in designing flexible, real-time systems [5].

In the industrial domain, and specifically in the context of Muvu Technologies, streaming data plays a central role [12]. Muvu's manufacturing systems are equipped with numerous IoT sensors that continuously generate telemetry, defect logs, and quality control data. For such environments, waiting for batch-processed insights is impractical and potentially costly. Decisions must be made in near real-time, whether to adjust machine parameters, detect anomalies, or trigger quality assurance protocols [12, 5]. This demand aligns with the overarching goals of Industry 4.0, where automation, adaptability, and intelligence are key [33, 31, 16].

Implementing effective data streaming pipelines introduces specific challenges: ensuring scalability across high-throughput data, maintaining consistency across distributed nodes, and managing evolving data distributions [5, 10, 4]. These become especially relevant in industrial settings, where concept drift is frequent, and predictive models need to be retrained continuously to preserve accuracy [30, 17].

*MultiFlow* aims to address these concerns by offering a flexible and transparent environment where data analysis pipelines can be developed, tested, and deployed in real-time. By simulating the same continuous data flow observed in production environments like Muvu's, *MultiFlow* helps bridge the gap between theoretical models and real-world deployment, allowing for practical experimentation and iterative development.

## 2.4 Machine Learning

ML is a subset of AI that enables systems to learn and improve from experience without being explicitly programmed. By using algorithms that build models based on sample data, often called "training data", machine learning systems can make predictions or decisions without direct human intervention. In recent years, machine learning has become a key technology in various fields, including finance, healthcare, marketing, and autonomous systems [4, 42].

At its core, machine learning operates through different types of learning paradigms:

- **Supervised Learning:** In supervised learning, models are trained on labeled datasets, where the input-output relationship is clearly defined. The model learns to map inputs to the correct output based on the given labels. This approach is widely used for tasks such as classification and regression. For example, in spam email detection, the model learns to classify emails as spam or not spam based on historical labeled data.
- **Unsupervised Learning:** Unlike supervised learning, unsupervised learning deals with unlabeled data. The system tries to find hidden patterns or intrinsic structures in the data. Clustering algorithms, which group similar data points together, are common examples of unsupervised learning. This is useful in applications like customer segmentation or anomaly detection.
- **Reinforcement Learning:** In this paradigm, agents learn by interacting with their environment, receiving feedback through rewards or penalties. Reinforcement learning is used in applications such as robotics, gaming, and self-driving cars, where systems continuously learn and adapt based on their actions and outcomes.

In data-intensive domains such as those targeted by Muvu Technologies, ML models are central to unlocking insights and automating decisions [4, 1, 18]. Muvu, which oversees multiple production lines, faces the constant challenge of dealing with vast volumes of sensor data. Their goal is to optimize operations and quickly respond to abnormalities or defects. However, building and maintaining robust ML models in such an environment is not straightforward.

Frequent model retraining is necessary to accommodate new conditions, shifts in behavior, or updated sensor readings [17, 30, 10]. This is further complicated by the need to ensure reliability and performance at scale, especially in streaming contexts. Furthermore, ML in industrial settings requires not only high predictive accuracy but also strong interpretability and low latency, as decisions often impact physical processes in real-time [18, 1].

The relevance of ML in this domain is therefore twofold, since it provides the core intelligence behind predictive and adaptive systems, and it shapes the foundation for more complex workflows involving real-time inference, automated monitoring, and proactive system maintenance [17, 4]. The development of tools like *MultiFlow* aims to bridge the gap between ML research and its application in production environments, allowing engineers and data scientists to simulate, evaluate, and iterate on streaming ML applications with greater ease and control.

As machine learning models operate on dynamic data, they must cope with distribution

changes, known as Concept Drift [30, 17].

Although this section provides a high-level overview of ML concepts, later chapters will delve deeper into more specific challenges such as model retraining strategies, evaluation in non-stationary settings, and the broader context of MLOps and industrial deployment.

## 2.5 Concept Drift

Concept drift occurs when the statistical properties of the target variable that a machine learning model aims to predict shift over time, altering the relationship between the features and the target [1, 30, 17]. This drift can reduce model accuracy and reliability, as the underlying data distribution changes and no longer aligns with the model's learned patterns [17, 14]. Concept drift presents a particular challenge in machine learning for dynamic environments, where the data or conditions change frequently [30, 43, 44].

Concept drift can take various forms, each posing unique challenges:

- Sudden Drift: This involves abrupt, significant changes in data distribution that occur instantaneously.
- Gradual Drift: In this case, changes in the data distribution happen slowly and incrementally, allowing for potential adaptation if detected in time.
- Incremental Drift: Characterized by continuous but minor changes that accumulate over time, leading to a steady shift in data characteristics.
- Recurrent Drift: This type occurs cyclically, where old patterns in the data reappear, often following seasonality or trends that resurface periodically.

The following figure (Figure 2.1) displays in a visual way the different forms of Concept Drift that can occur.

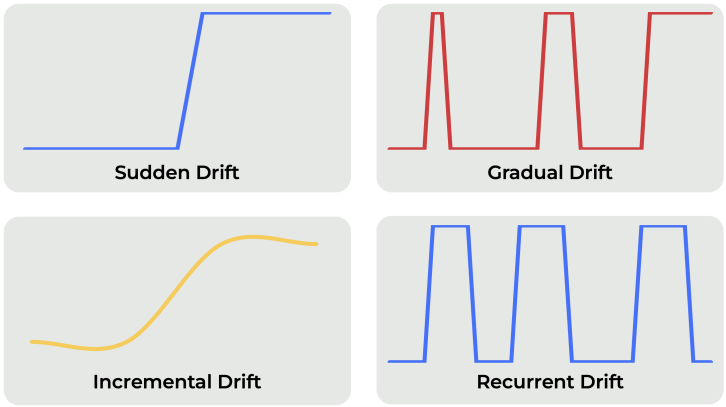



Figure 2.1: Visual representation of different forms of Concept Drift. (Source: author)

Concept drift complicates the deployment of machine learning models in real-world settings by requiring constant monitoring, adaptation, and possibly retraining [30]. It is particularly problematic in high-stakes applications, such as medical diagnosis or financial forecasting, where shifts in data patterns can lead to substantial impacts [1, 17]. For instance, concept drift has a significant effect on clinical models used for sepsis predic-

tion, as changing patient demographics or evolving disease patterns can disrupt model performance.

Mitigating concept drift requires several strategies: regular model retraining, the use of adaptive algorithms that adjust as new data becomes available, and proactive monitoring systems to detect and address drift as it occurs [30]. These approaches are essential to maintaining the effectiveness and robustness of machine learning solutions in changing environments.

## 2.6 Frugal AI

Frugal AI emphasizes the development of artificial intelligence solutions that maximize efficiency and minimize dependency on extensive resources [30, 17, 45]. Its goal is to maintain competitive performance while reducing the demands on data, computation, time, and human labor. While this paradigm is especially valuable in scenarios where data is scarce, expensive to collect, or subject to rapid changes, as is often the case in real-world industrial and streaming environments, it is also increasingly adopted as a deliberate strategy to reduce energy consumption and environmental impact [18, 30, 17]. In this sense, Frugal AI is relevant even in high-resource contexts, where sustainability or general scalability improvement goals exist.

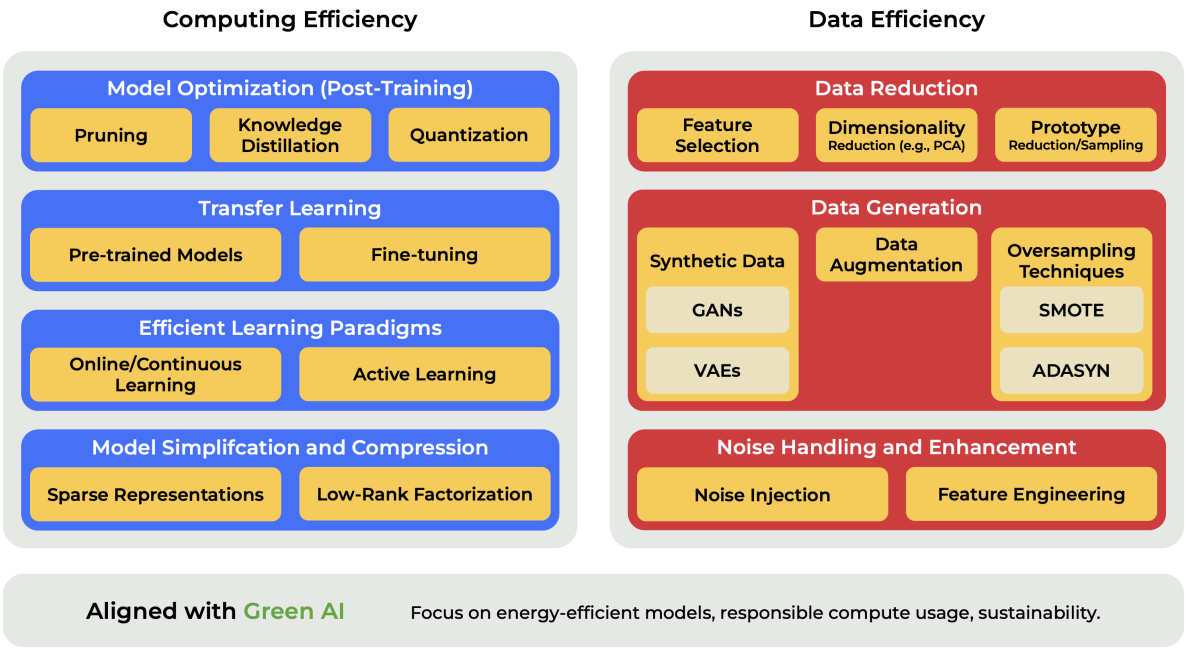



Figure 2.2: Main approaches to Frugal AI from both computing and data perspectives. (Source: author)

As illustrated in Figure 2.2, Frugal AI strategies can be broadly divided into two categories: those that optimize the computing aspect of AI (Computing Efficiency), and those that address the data aspect (Data Efficiency).

**Computing-oriented approaches** revolve around reducing the computational footprint of machine learning models, especially deep learning models. Key techniques include:

- **Post-training methods** such as pruning, quantization, and distillation, which simplify trained models by reducing their size and computational requirements without significant performance loss;
- **Transfer learning**, including semantic domain adaptation and fine-tuning, to reduce training costs and required data;
- **Online and continuous learning** (e.g., active learning), which allows models to evolve with streaming data with minimal retraining;
- **Ensemble methods**, like stacked generalization or rotation-based strategies, to boost robustness while using simpler learners.

**Data-oriented approaches** aim to optimize how data is used, especially when collecting or labeling it is costly. Key strategies include:

- **Data reduction**, such as random sampling, PCA, or prototype reduction, to keep datasets compact without losing essential information;
- **Data generation**, via augmentation, synthetic creation (GANs, VAEs), or oversampling (e.g., SMOTE), to enrich training without needing more real data;
- **Noise injection and transformation**, to improve generalization by making models more robust to imperfect or evolving inputs.

Frugal AI also aligns with **Green AI** by emphasizing sustainability, reducing energy consumption, and lowering the environmental footprint of machine learning [46, 30]. In practical terms, this translates into using lighter models, reusing existing models through transfer learning, or minimizing retraining frequency through adaptive learning techniques.

In the context of real-time streaming and industrial environments, Frugal AI becomes even more relevant. The ability to adapt quickly to shifting data distributions while maintaining performance under resource constraints is crucial [17]. When integrated with platforms like Muvu, which emphasize responsiveness and efficient decision-making in dynamic contexts, Frugal AI strategies provide the foundation for scalable and sustainable AI applications [30].

In summary, Frugal AI is not just a reaction to limited resources, it is a proactive strategy that enhances the accessibility, scalability, and environmental responsibility of AI systems [46].

## 2.7 MLOps

MLOps is a discipline that brings together machine learning (ML), software engineering, and DevOps principles to streamline the development, deployment, and maintenance of ML systems in production environments [10, 30]. In dynamic domains such as industrial monitoring and decision support (like those addressed by the Muvu platform), MLOps plays a crucial role in ensuring that ML solutions are not only robust and scalable but also responsive to real-time changes in data and operating conditions [17].

Like DevOps, which automates and optimizes the software development lifecycle [47], MLOps introduces automation and governance into the ML lifecycle. However, it must

also tackle unique challenges, such as managing the evolving behavior of ML models, ensuring reproducibility, and handling continuous integration of new data sources.

Key components of MLOps include:

- **Continuous Integration (CI)/Continuous Delivery (CD)**: Automated pipelines that allow teams to integrate new model versions quickly, test them, and deploy them into production with minimal human intervention. This is particularly valuable in streaming applications where quick iteration is essential.
- **Automated Testing and Validation**: ML models must be rigorously tested to ensure they maintain performance under diverse and evolving conditions. Automated testing frameworks validate models not only for accuracy but also for stability and reliability.
- **Version Control for Models and Data**: Managing code alone is insufficient in ML systems. MLOps enforces tracking and managing versions of models and their training datasets, ensuring full traceability and reproducibility, a must for regulated or critical applications.
- **Monitoring and Alerting**: In real-time systems, ongoing model performance monitoring is vital. When deployed in environments like industrial data streams (e.g., in Muvu), performance degradations must be detected early. This can involve tracking data drift, model accuracy, and system latency.
- **Infrastructure as Code and Automation**: MLOps promotes using tools like Kubernetes, MLflow, and Terraform to manage infrastructure. This supports scalable training and deployment environments that mirror production scenarios and support rapid experimentation.

MLOps is especially relevant in scenarios where data is continuously generated, as in IoT systems or industrial process monitoring [9, 7]. These contexts demand that ML systems adapt without significant downtime, making MLOps essential for automating retraining, redeployment, and rollback mechanisms.

Despite its potential, MLOps adoption still faces obstacles. Many ML projects fail to go beyond the proof-of-concept stage due to a lack of automated workflows, insufficient monitoring, or difficulty integrating with existing IT infrastructure [2, 5]. In initiatives like Muvu, which aim to support decision-making in real-time environments, MLOps offers a structured approach to operationalizing ML, thereby bridging the gap between experimental development and production-grade reliability.

In summary, MLOps is foundational for scaling ML in complex, high-frequency environments. By integrating automation, monitoring, and governance across the ML lifecycle, MLOps enables systems to remain robust, adaptable, and aligned with ever-changing data and operational needs.

## 2.8 Industrial AI

AI is widely recognized as a key driver of Industry 4.0, particularly for applications such as automation, predictive maintenance, and real-time decision-making in industrial environments [1, 13, 31, 33, 40]. However, applying AI in this context introduces domain-specific

challenges that differ significantly from traditional or consumer-focused applications [10].

A key constraint in industrial AI is the real-time nature of many use cases. Models must often process streaming data with minimal latency to detect anomalies, trigger alerts, or support critical decisions before faults escalate [30, 17]. This demands not only fast inference times but also efficient use of computational resources, particularly when models are deployed on edge devices with limited memory and processing power.

The quality and characteristics of industrial data present another major hurdle. Sensor data is frequently noisy, incomplete, and highly imbalanced—especially in scenarios like anomaly detection, where failure events are rare by design [17]. Moreover, interpreting such data often requires strong domain knowledge, making it difficult to rely solely on purely data-driven approaches without incorporating expert input or contextual understanding [7, 1, 2].

In addition, trust and interpretability are crucial in industrial settings. AI models influence decisions that can affect safety, productivity, or product quality [3]. As a result, black-box models are often inadequate. Human operators and engineers need to understand, validate, and sometimes override AI-driven outputs, increasing the relevance of explainable AI (XAI) methods in this field [12].

Another significant challenge is technology adoption itself. Many industrial organizations face structural and cultural barriers when trying to implement AI-driven solutions [4, 19, 28]. These may include a lack of specialized human resources, limited data science expertise, weak data governance practices, or even organizational resistance to change. In some cases, companies struggle with low digital maturity or insufficient infrastructure, making it difficult to support the full lifecycle of AI, from development and deployment to monitoring and retraining.

Frameworks like Reference Architectural Model for Industry 4.0 (RAMI 4.0) help conceptualize how AI fits within industrial systems [48]. The RAMI 4.0 architecture describes systems across three dimensions:

- Hierarchy levels, from field devices to enterprise systems;
- Life cycle and value stream, from development to decommissioning;
- Layers, including physical, communication, information, and functional domains.

AI technologies span multiple layers, especially the information and functional ones, and play a key role in enabling higher levels of autonomy and decision support [1].

AI contributes directly to several core principles of Industry 4.0:

- Interoperability: Through standards-based APIs and protocols, AI systems can communicate across heterogeneous environments (e.g., linking sensor data to cloud analytics or MES systems).
- Information transparency: By transforming raw sensor data into meaningful metrics (e.g., KPIs or health indicators), MF-Workflow makes hidden patterns visible for operators and decision-makers.
- Decentralized decision-making: Edge-deployed MF-Workflow models can take real-time actions (e.g., shutdown machines on anomaly detection) without waiting for centralized systems.

- Technical assistance: MF-Workflow supports workers via predictive maintenance alerts, automatic quality inspection, or even recommendation systems in control rooms.

In conclusion, while the potential of AI in industrial environments is immense, its effective deployment requires navigating constraints such as real-time requirements, data challenges, explainability needs, legacy system integration, and organizational readiness [19]. These challenges emphasize the importance of resource-efficient and context-aware AI approaches, such as those promoted under the umbrella of Frugal AI, which aim to deliver impact without excess [17, 45].

## 2.9 Digital Twins

Digital Twins represent one of the most transformative paradigms in modern industrial and cyber-physical systems [2, 49]. At their core, a Digital Twin is a virtual representation of a physical system that is continuously updated with real-time data from its physical counterpart [49, 50]. This connection allows the virtual model to simulate, monitor, and even predict the behavior of the real system under varying conditions. Originally conceived in the context of aerospace and manufacturing, the concept has expanded into domains such as healthcare, energy, logistics, and smart cities.

A Digital Twin goes beyond traditional simulations by maintaining a live link to the real-world entity it represents. This continuous synchronization enables dynamic behavior modeling, predictive maintenance, anomaly detection, and "what-if" scenario analysis, often in real time [50]. A key aspect of this paradigm is the ability to not only mirror the state of the system but to test hypothetical changes and observe their virtual impact before implementation in the physical environment.

There are generally three core components of a Digital Twin system:

- The physical system, which generates data through sensors and other sources;
- The virtual model, which simulates the state, behavior, or processes of the physical system;
- The data connection layer, which synchronizes the two in real-time and enables bi-directional communication.

In the context of industrial MF-Workflow, Digital Twins offer a way to prototype and validate AI-driven decision-making processes in a safe and cost-effective environment [49, 50]. For instance, one can test predictive maintenance algorithms on a twin before applying them to a real machine, avoiding unnecessary downtime or failures.

*MultiFlow* embodies many of these characteristics. While it does not aim to fully replicate every mechanical or structural detail of a system, acts as a lightweight Industrial Digital Twin, focused on the flow and processing of real-time data. The system allows users to simulate operational behavior based on live or historical data streams, inject configurable algorithms into the processing pipeline, and monitor outputs in real-time dashboards. This makes it possible to prototype complex machine learning workflows, evaluate them under dynamic and realistic conditions, and iterate rapidly based on feedback.

Moreover, *MultiFlow* supports multiple concurrent applications and datasets, enabling

comparisons between algorithms, parameter configurations, and behavioral outcomes. This aligns closely with one of the main promises of Digital Twins: enabling experimentation without risk to the physical system. For industrial partners such as Muvu, this capability is particularly valuable in assessing the performance of anomaly detection or concept drift strategies before deploying them to production environments.

As industries become increasingly reliant on MF-Workflow for operational intelligence, Digital Twins like *MultiFlow* serve as a bridge between abstract models and their real-world implications. By merging simulation, data streaming, and algorithmic experimentation into a single integrated tool, *MultiFlow* contributes to the broader vision of Industry 4.0, where digital and physical layers work in concert to create smarter, more adaptive systems.

# Chapter 3

# Design and Specification

## 3.1 Requirements

Defining clear system requirements is an essential step to ensure that *MultiFlow* aligns with its intended purpose of supporting real-time industrial data analysis and experimentation. The requirements provide a structured view of what the tool must achieve (functional requirements) and how it should perform under different conditions (non-functional requirements).

Functional requirements describe the concrete features and behaviors of the system, such as how users manage projects, configure apps, and visualize outputs. Non-functional requirements, on the other hand, capture quality attributes such as performance, scalability, security, and usability. Together, these requirements form the foundation that guided the design and development of *MultiFlow*, ensuring that the resulting platform is both practical and reliable for its target use cases.

### Functional Requirements

The functional requirements of *MultiFlow* define the expected features and behaviors of the tool. They describe what the system must do to support real-time data stream simulation, app management, and visualization.

- **FR1. Navigation and Layout:** The web interface must be organized into distinct pages for Projects, Apps, Datasets, and Logs. This separation ensures clarity of workflows and avoids cluttering the interface with unrelated functions.
- **FR2. Intuitive Menu and State Preservation:** The navigation menu must provide straightforward access to all pages. Switching between pages must not reset or lose project status, progress tracking, or user-defined configurations, guaranteeing continuity during experimentation.
- **FR3. Collapsible Navigation Menu:** The navigation sidebar must support expand/collapse functionality, enabling users to focus on content when needed and maintain a clean, organized workspace.
- **FR4. Project Information Overview:** The Projects page must present summary information for each project, including name, creation date, last execution,

and description. This allows users to quickly identify and manage active or historical projects.

- **FR5. Expandable Project Items:** Each project must be expandable (default collapsed), displaying associated streams and their status. This hierarchical structure makes it easier to manage projects with multiple streams.
- **FR6. Stream Configuration:** Each stream must allow configuration of simulation parameters such as speed and duration. The list of available datasets must always reflect the contents of the system's "datasets" folder, ensuring consistency between the UI and stored files.
- **FR7. Project Execution:** Projects must be executable directly from the Projects page (via a start icon) or from the detailed Project view, providing flexibility for both quick actions and in-depth management.
- **FR8. Progress Tracking:** A project's overall progress must be defined by the lowest progress value across its streams. Both project-level and stream-level progress must be visible in the interface. This ensures accurate feedback and prevents misleading results when streams finish at different times.
- **FR9. Reset on Stop:** Stopping a running project must reset its progress instead of pausing it. This avoids inconsistencies in repeated simulations and guarantees reproducibility when comparing experimental runs.
- **FR10. Project Shortcuts:** Active projects must appear as shortcuts in the navigation menu, displaying project name and progress percentage. Clicking the shortcut must redirect to the corresponding Project details page, allowing quick access to ongoing work.
- **FR11. App Information Overview:** The Apps page must display summary information (name, creation date, last update, description) for each App, allowing users to identify relevant algorithms quickly.
- **FR12. Expandable App Items:** Each App must be expandable (default collapsed), listing its associated Instances. This allows users to see active and historical deployments in context.
- **FR13. App Configuration:** Each App must allow configuration of custom parameters (e.g., thresholds, batch size) required by its script. The available Python scripts must always reflect the contents of the system's "code" folder.
- **FR14. Instance Execution:** Instances must be "startable" from either the Apps page (when expanded) or from the App or Instance details pages. This provides flexibility for quick launches and detailed configuration.
- **FR15. Instance Creation Workflow:** When creating a new Instance, the tool must automatically display all required configuration fields, allowing the user to simply fill in values. This reduces setup time and lowers the barrier for rapid prototyping.
- **FR16. Instance Shortcuts:** Active Instances must appear as shortcuts in the navigation menu, displaying instance name. Clicking the shortcut must redirect to the Instance details page.

- **FR17. Logs and Debugging:** App and Instance details pages must display live logs from the underlying Faust container. Basic utilities such as copy, clear, and search must be provided to make debugging easier and more transparent.
- **FR18. Live Visualization:** Instance outputs must be visualized in real time through Grafana dashboards, providing metrics, alerts, and performance indicators. This enables immediate feedback on algorithm behavior.
- **FR19. Input Validation:** When users fill out forms (e.g., creating or editing Projects, Apps, or Instances), the system must validate inputs for duplicate values, invalid formats, or missing mandatory fields. This prevents errors and ensures data integrity.

## Non-Functional Requirements

In addition to the functional aspects described earlier, *MultiFlow* must also satisfy a set of non-functional requirements to ensure that the tool is robust, reliable, and usable in both research and industrial contexts. These requirements address aspects such as performance, scalability, usability, maintainability, and portability.

- **NF1. Performance and Responsiveness:** The system must support real-time data streaming with low latency to allow smooth simulations. User interactions (e.g., starting or stopping a project, launching an app) should be reflected within a few seconds to preserve the sense of live control. This requirement is essential in industrial contexts, where delayed feedback can hinder experimentation or misrepresent system behavior.
- **NF2. Scalability:** *MultiFlow* must be able to handle increasing numbers of projects, apps, and instances without significant degradation of performance. The use of containerized services (via Docker) ensures that additional components can be deployed horizontally if needed. Scalability is crucial when simulating multiple parallel industrial scenarios or testing several algorithms simultaneously.
- **NF3. Reliability and Fault Tolerance:** The tool must be resilient to individual component failures (e.g., Kafka broker crash, Faust app error). Mechanisms for recovery or restart must be in place to minimize disruption. This reflects real industrial requirements, where continuous operation is expected even under partial system faults.
- **NF4. Usability:** The user interface must remain simple and intuitive, even for users with limited technical background. Clear navigation, contextual shortcuts, and error messages should guide the user through workflows. While no formal usability study has yet been conducted, design decisions should prioritize reducing cognitive load and simplifying repetitive tasks.
- **NF5. Maintainability and Extensibility:** The system must be easy to extend with new features, apps, or visualizations. Code should follow modular design practices, ensuring that changes in one component (e.g., Kafka configuration, Faust processing logic) have minimal impact on others. This guarantees that the tool can evolve alongside new research directions or industrial needs.
- **NF6. Documentation and Transparency:** The system must include clear doc-

umentation for installation, configuration, and usage. Transparency in how apps are executed (e.g., through logs or dashboards) is required to make the tool not only a black-box executor but also a learning and validation environment for both researchers and industrial practitioners.

These requirements provide the foundation for the design decisions described in the following sections, ensuring that *MultiFlow* balances research flexibility with industrial applicability.

Before delving into the architectural details of the tool, it is useful to illustrate how the system was envisioned from a user perspective.

## 3.2 Interface Mockups and Conceptual Design

Before the implementation of *MultiFlow*, a set of mockups was created to outline the intended structure and user experience of the tool. These mockups were developed using Figma, a collaborative design tool that enables rapid prototyping and visualization of user interfaces. While they do not represent the final version of the system, they provided a valuable reference point for guiding development decisions and aligning expectations regarding the layout, navigation, and interaction flow. In practice, the mockups helped validate ideas early and offered a near-realistic preview of the final interface.

The high-level view shown in Figure 3.1 illustrates the structure of the tool, centered around a sidebar that grants access to the key functional components: Projects, Streams, Apps, and Instances. This structure was designed to minimize complexity while ensuring that users can quickly switch between tasks. It also emphasizes consistency, with a uniform layout across different sections to support smoother on-boarding and usability.

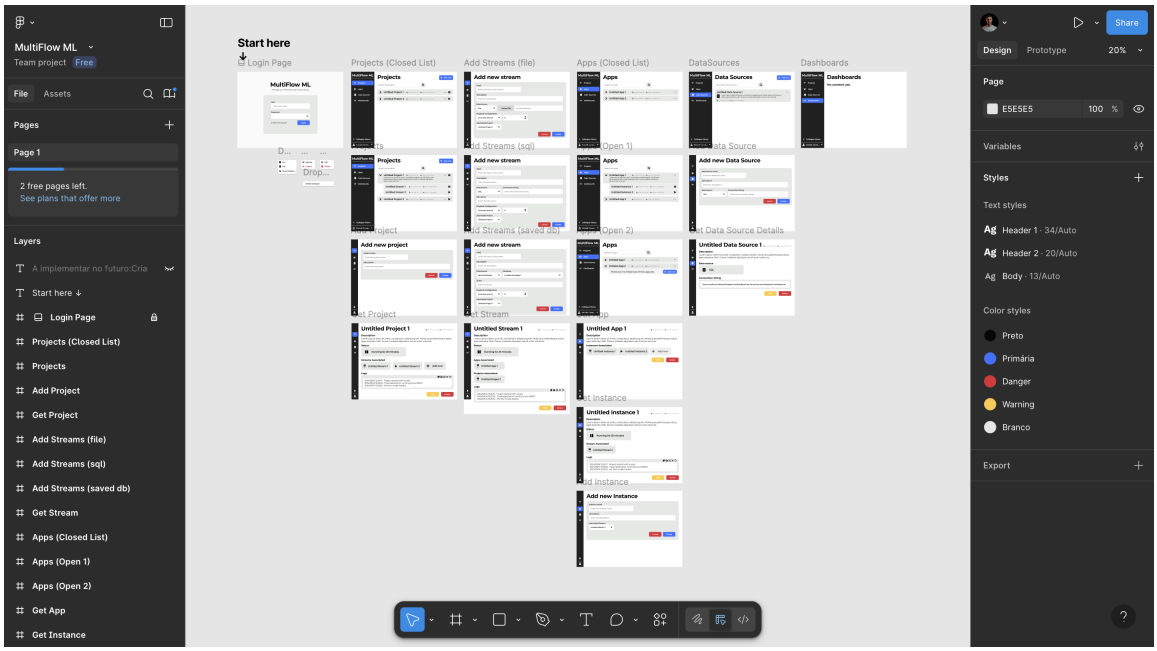



Figure 3.1: High-level view of *MultiFlow*'s mockups project created in Figma.

The mockups for the Projects and Stream Details pages (Figure 3.2) highlight how workflows are structured around Projects as top-level containers. Each Project card displays

essential metadata such as the name, description, creation date, and last execution. When expanded, the card reveals associated Streams, which represent configured data ingestion scenarios. Stream details allow users to configure dataset replay options (rows per second, all in seconds, or real-time mode) and connect them to downstream Apps. This design reinforces logical grouping and provides a compact yet informative overview of ongoing work.

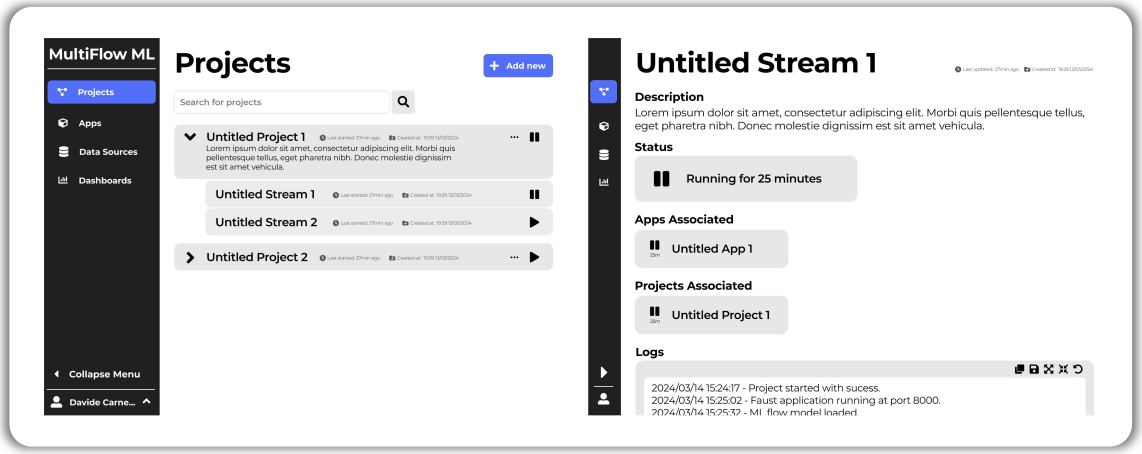



Figure 3.2: Mockups of the Projects page (left) and Stream Details view (right).

The Apps and Instance Details mockups (Figure 3.3) demonstrate the mechanism for managing algorithm scripts and their deployments. Apps serve as reusable templates for stream processing logic, while Instances represent specific executions linked to a chosen Stream and configuration. This design enables side-by-side comparisons of multiple runs of the same algorithm, supporting fast experimentation. The Instance Details view includes runtime logs, which is a feature that was later integrated into this page View, allowing the user to get more control on the executed script's feedback.

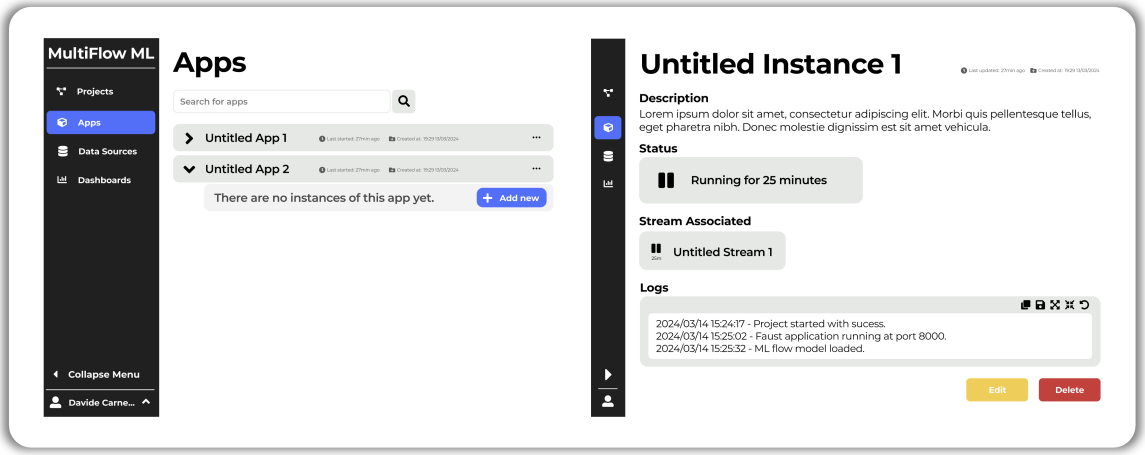



Figure 3.3: Mockups of the Apps view (left) and Instance Details view (right).

In summary, the mockups provided an early blueprint of the tool and ensured that usability considerations were integrated from the start. Many of the final interface decisions, including the sidebar navigation, expandable project cards, and instance logs, trace their origins to these initial conceptual designs.

## 3.3 System Architecture and Tool Design

*MultiFlow* was conceived with industrial applicability in mind, using technologies that are both robust and widely adopted in production environments. This ensures easier integration with live data streams and a smoother transition from prototype to deployment.

The tool follows a modular and containerized architecture. Core backend components, including Kafka, Faust, InfluxDB, MongoDB, and the Node.js backend, are deployed via Docker, guaranteeing consistent behavior across environments, simplified orchestration, and high portability. Kafka serves as the backbone for data transmission: as a distributed messaging system, it enables high-throughput, fault-tolerant stream handling [22]. Data streams are organized into Kafka topics, which can be consumed in parallel by multiple applications. Faust, a Python stream processing library built on Kafka, powers the creation of user-defined "Apps" that implement real-time logic such as filtering, feature extraction, anomaly detection, or inference.

MongoDB stores project configurations, stream definitions, and metadata, offering a flexible schema suitable for evolving systems under active experimentation. The Node.js backend exposes REST APIs for all services, manages project lifecycles and app deployments, and maintains real-time communication with the interface through WebSockets.

On the user-facing side, *MultiFlow* includes a React.js interface designed to provide a clear and intuitive experience. Its layout is organized around a side menu with key tabs such as "Projects" and "Apps". Projects allow users to configure multiple data streams, while the Apps tab manages algorithms, their configurations, and deployed instances. Features like clickable shortcuts on active streams and running instances guide navigation across views, enabling users to start or pause simulations, and configure apps with minimal friction.

Although no formal usability study has yet been conducted, the design prioritizes simplicity, clarity, and extensibility, supporting future additions such as new tabs, settings, or monitoring views.

To support real-time monitoring, processed outputs from Faust Apps are written to InfluxDB, a time-series database that serves as a temporary store for metrics such as predictions, aggregated statistics, or detected anomalies. These results are visualized through Grafana dashboards, chosen for its seamless integration with InfluxDB, industrial relevance, and the ability to provide real-time insights. Grafana enables users to observe application behavior, compare results between versions, and identify issues or improvements that are valuable for model tuning or detecting concept drift.

As mentioned above, a key strength of the tool lies in its modularity. With Docker, each component can be independently swapped, scaled, or updated with minimal disruption to the system. This flexibility allows *MultiFlow* to operate both as a research sandbox and as an industrial grade simulation tool, bridging the gap between academic experimentation and deployment in real-world scenarios.

Figure 3.4 illustrates the overall architecture of the toll, showing the interaction between the back-end services, the user interface, and the visualization stack.

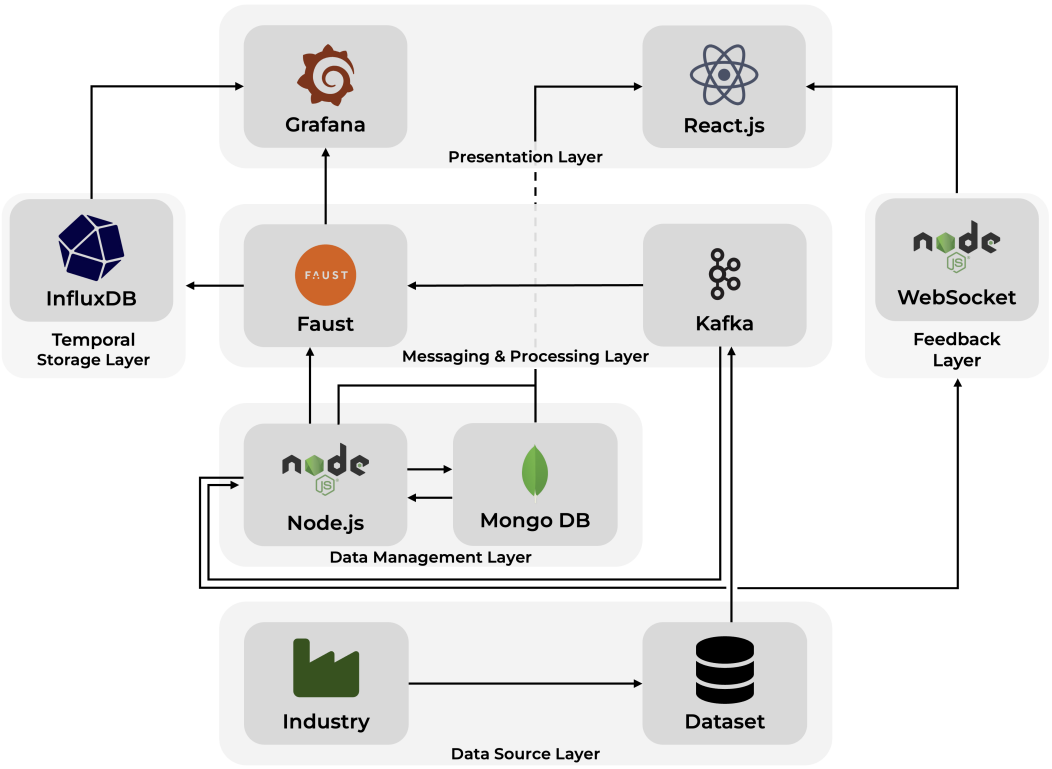



Figure 3.4: Overall architecture of the *MultiFlow* tool and its core technologies. (Source: author)

# Chapter 4

# Implementation

This chapter provides a detailed breakdown of the internal architecture and development of *MultiFlow*, a tool designed for simulating and analyzing real-time industrial data streams. The chapter follows a top-down structure, beginning with user-facing concepts and workflows before exploring the underlying infrastructure that enables them. This order ensures that readers first understand what the tool does, and then how each technological layer supports these capabilities.

## 4.1 Functional Overview

At a functional level, *MultiFlow* is designed to help users prototype, test, and monitor machine learning workflows in streaming environments. To achieve this, it introduces several core building blocks:

- **Projects** — High-level containers that group multiple data streams. Projects typically correspond to an industrial process or experimental setup, providing a logical boundary for organizing work.
- **Streams** — Configurable data flows that mimic real-time sensor inputs. Streams are created by uploading datasets (e.g., CSV files) and defining replay parameters such as rows per second, total duration in seconds, or real-time mode (for direct connections to external sources).
- **Apps** — Python-based processing scripts built to run by Faust, a stream processing library. These scripts implement real-time tasks such as anomaly detection, drift detection, or statistics computation.
- **Instances** — Deployments of Apps, each linked to a specific Stream and parameter configuration. Multiple Instances can run in parallel to explore different configurations or scenarios. Outputs are directed to dashboards for real-time monitoring.

A typical user workflow unfolds as follows:

1. The user creates a new Project to represent an experiment or production scenario.
2. Within the Project, one or more Streams are defined from historical datasets or live sources.

3. Apps are then created or imported, encapsulating the desired processing logic.
4. Instances of these Apps are launched, each bound to a Stream with custom configuration parameters.
5. The system streams data into Kafka, Apps consume and process the events, and outputs are written into InfluxDB.
6. Finally, Grafana dashboards display the results in real time, allowing the user to evaluate performance, compare scenarios, or detect anomalies.

Figure 4.1 illustrates the overall workflow of *MultiFlow*, from user-level actions to system-level operations. At the top, users interact with the interface to create Projects, configure Streams, and deploy Apps as Instances. At the bottom, the system automatically handles data streaming, processing, and visualization: Kafka delivers the simulated data, Apps consume and analyze events, results are stored in InfluxDB, and Grafana provides real-time dashboards. This separation highlights how *MultiFlow* bridges user-friendly configuration with robust backend infrastructure.

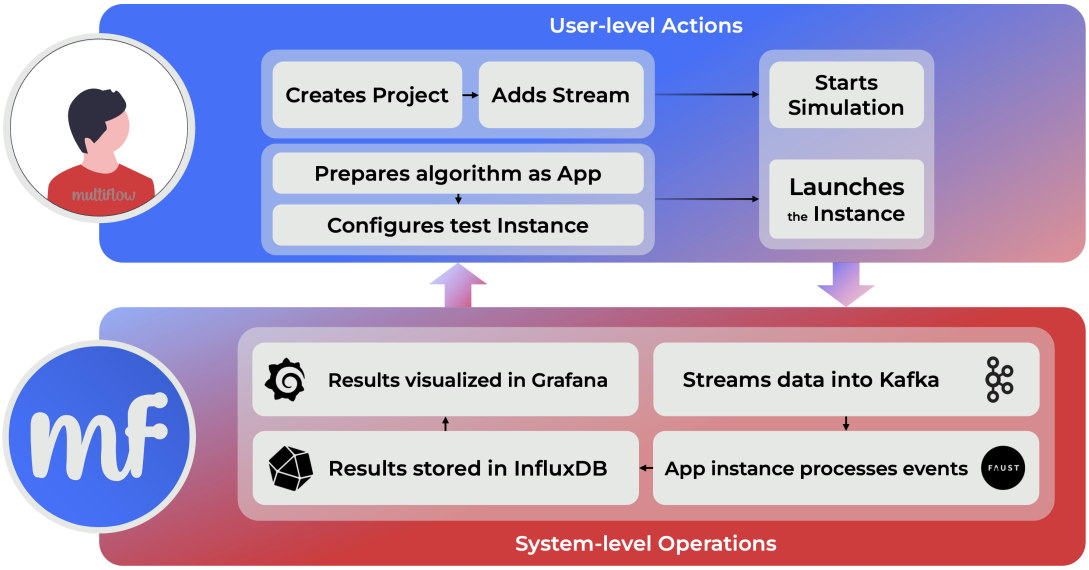



Figure 4.1: Overall workflow of *MultiFlow*, showing user-level actions and system-level operations. (Source: author)

## 4.2 Data Streaming Layer

At the core of *MultiFlow*'s infrastructure lies Apache Kafka, a distributed event streaming platform widely adopted in industrial systems for high-throughput and low-latency data pipelines [12, 22]. Kafka acts as the communication backbone, linking simulated or real sensor inputs to the processing logic implemented in Apps.

When a dataset is uploaded, it becomes available within the scope of the selected Project and Stream. The dataset is stored and can be replayed multiple times under different configurations. During a simulation, records are published sequentially into a dedicated Kafka topic at a user-defined rate:

- **Rows per second**: publishes a fixed number of dataset rows per second, simulating sensors with a given frequency.
- **All in seconds**: replays the entire dataset over a specified duration, distributing rows proportionally.
- **Real-time (experimental)**: designed for direct connections to live machines or external servers, using Kafka purely as a transport layer.

This design allows multiple streams to reuse the same dataset with different replay rates, creating distinct Kafka topics for different scenarios. In practice, this enables "what-if" testing without altering the dataset itself.

Kafka is deployed through Docker containers, and its producers, consumers, and topics are dynamically managed by the backend. This removes the burden of manual setup while preserving Kafka's scalability and reliability.

## 4.3 Application Layer: Real-Time Processing Apps

The processing logic in *MultiFlow* is handled by Faust, a Python library for stream processing inspired by Kafka Streams. Faust allows users to define asynchronous *agents* that consume messages from Kafka topics and apply transformations or analysis in real time.

As represented in Figure 4.2, each App follows a standard structure:

1. **Imports** — packages for stream handling, ML, and utilities;
2. **Config injection** — runtime parameters supplied by the frontend (e.g., thresholds);
3. **Faust setup** — topic bindings and stream definitions;
4. **Processing logic** — user-defined function executed per incoming record;
5. **Outputs** — optional writing of results to InfluxDB, logs, or files.

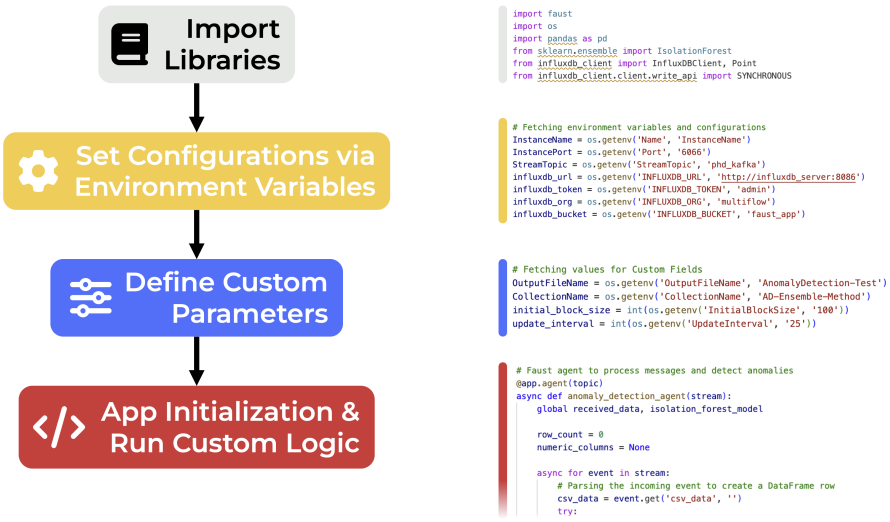



Figure 4.2: Structure of a typical *MultiFlow* App. (Source: author)

Apps are designed to be modular and reusable. The same algorithm can be deployed across different Streams by launching multiple Instances with varying parameters, enabling side-by-side comparison of results.

### 4.3.1 Example App: Concept Drift Detection with MMD

To illustrate how *MultiFlow* Apps are implemented in practice, a complete example is presented: an App that detects *concept drift* using the Maximum Mean Discrepancy (MMD) method. This App consumes streaming data, monitors distributional changes, and outputs detection results to InfluxDB for real-time visualization in Grafana.

```
1  import faust
2  import os
3  import pandas as pd
4  from frouros.detectors.data_drift import MMD
5  from influxdb_client import InfluxDBClient, Point
6
7  # Environment configurations
8  InstanceName = os.getenv('Name', 'InstanceName')
9  StreamTopic = os.getenv('StreamTopic', 'phd_kafka')
10 concept_samples = int(os.getenv('ConceptSamples', '100'))
11 batch_size = int(os.getenv('BatchSize', '10'))
12 mmd_threshold = float(os.getenv('MMDThreshold', '0.8'))
13
14 # Faust app setup
15 app = faust.App(InstanceName, broker='kafka_server://localhost:9092')
16 topic = app.topic(StreamTopic)
17
18 # InfluxDB configurations
19 CollectionName = os.getenv('CollectionName', 'drift_detection')
20 influxdb_url = os.getenv('INFLUXDB_URL', 'http://influxdb_server:8086')
21 influxdb_token = os.getenv('INFLUXDB_TOKEN', 'admin')
22 influxdb_org = os.getenv('INFLUXDB_ORG', 'multiflow')
23 influxdb_bucket = os.getenv('INFLUXDB_BUCKET', 'faust_app')
24
25 influx_client = InfluxDBClient(url=influxdb_url, token=influxdb_token, org=influxdb_org)
26 write_api = influx_client.write_api()
27
28 # Drift detection setup
29 reference_data = pd.DataFrame()
30 detector = None
31 initialized = False
32
33 def send_to_influxdb(df):
34     for _, row in df.iterrows():
35         point = Point(CollectionName).tag("drift_detected", row['drift_detected'])
36         write_api.write(bucket=influxdb_bucket, record=point)
37
38 @app.agent(topic)
39 async def drift_detection_agent(stream):
40     global reference_data, detector, initialized
41     async for event in stream:
42         # Convert incoming event into DataFrame row
43         row_values = list(map(float, event['csv_data'].split(',')))
44         row_df = pd.DataFrame([row_values], columns=['col1'])
45
46         # Initialize MMD detector with reference data
```

```
47          if not initialized:
48              reference_data = pd.concat([reference_data, row_df])
49              if len(reference_data) >= concept_samples:
50                  detector = MMD()
51                  detector.fit(X=reference_data.to_numpy())
52                  initialized = True
53              continue
54
55          # Compare batch with reference
56          drift_result, _ = detector.compare(X=row_df.to_numpy())
57          drift_detected = drift_result.distance > mmd_threshold
58          row_df['drift_detected'] = 'yes' if drift_detected else 'no'
59
60          # Send results to InfluxDB
61          send_to_influxdb(row_df)
62
63  if __name__ == '__main__':
64      app.main()
```

Listing 4.1: Simplified version of a Concept Drift Detection App using MMD in Faust

This App follows the general structure introduced earlier, with each section corresponding to a specific responsibility:

- **Imports and setup:** In addition to core libraries (`faust`, `pandas`), the script imports the MMD detector from the `frouros` library, as well as the InfluxDB client to persist results.
- **Configuration:** All runtime parameters (e.g., instance name, stream topic, threshold) are injected via environment variables. This design allows the same script to be reused across multiple Instances with different configurations.
- **Faust initialization:** The script creates a Faust application bound to Kafka, defining the topic from which data will be consumed.
- **Reference data collection:** The first set of samples received are used to initialize the MMD detector, establishing the baseline distribution against which new data will be compared.
- **Drift detection logic:** Once initialized, the App processes incoming batches of records. For each batch, the MMD distance is computed against the reference data. If the distance exceeds the configured threshold, a drift event is flagged.
- **Output handling:** Detection results are both appended to local storage (CSV) and streamed to InfluxDB, where they can be visualized in Grafana in real time.

This breakdown illustrates how *MultiFlow* enables the rapid prototyping of real-time processing tasks. The App developer only needs to focus on the detection logic, while the system takes care of data ingestion, orchestration, and result visualization.

### 4.3.2 Example App: Anomaly Detection with Isolation Forest Ensembles

Another example demonstrates how *MultiFlow* can support anomaly detection tasks using an ensemble of Isolation Forest models. The App consumes unlabelled data streams, learns

the data distribution, and flags anomalies in real time. Results are written to InfluxDB and visualized through Grafana dashboards.

```
import faust
import os
import pandas as pd
from sklearn.ensemble import IsolationForest
from influxdb_client import InfluxDBClient, Point

# Environment and configuration
InstanceName = os.getenv('Name', 'InstanceName')
StreamTopic = os.getenv('StreamTopic', 'phd_kafka')
contamination = int(os.getenv('CONTAMINATION', '0.1'))
estimators = int(os.getenv('N_ESTIMATORS', '100'))

# Faust setup
app = faust.App(InstanceName, broker='kafka_server://localhost:9092')
topic = app.topic(StreamTopic)

# InfluxDB configurations
CollectionName = os.getenv('CollectionName', 'anomaly_detection')
influxdb_url = os.getenv('INFLUXDB_URL', 'http://influxdb_server:8086')
influxdb_token = os.getenv('INFLUXDB_TOKEN', 'admin')
influxdb_org = os.getenv('INFLUXDB_ORG', 'multiflow')
influxdb_bucket = os.getenv('INFLUXDB_BUCKET', 'faust_app')

influx_client = InfluxDBClient(url=influxdb_url, token=influxdb_token, org=influxdb_org)
write_api = influx_client.write_api()

# Placeholder for data and model
received_data = pd.DataFrame()
isolation_forest_model = None

def train_isolation_forest(dataframe, contamination=contamination):
    model = IsolationForest(n_estimators=estimators, contamination=contamination)
    model.fit(dataframe)
    return model

def send_to_influxdb(df):
    for _, row in df.iterrows():
        point = Point(CollectionName).tag("anomaly", row['anomaly'])
        write_api.write(bucket=influxdb_bucket, record=point)

@app.agent(topic)
async def anomaly_detection_agent(stream):
    global received_data, isolation_forest_model
    async for event in stream:
        # Parse event into DataFrame
        row_values = list(map(float, event['csv_data'].split(',')))
        row_df = pd.DataFrame([row_values], columns=['col1', 'col2'])

        # Initialize or update model
        if isolation_forest_model is None:
            isolation_forest_model = train_isolation_forest(row_df)

        scores = isolation_forest_model.decision_function(row_df)
        predictions = isolation_forest_model.predict(row_df)
        row_df['scores'] = scores
```

```
56            row_df['anomaly'] = ['yes' if p == -1 else 'no' for p in predictions]
57
58            # Append and send results
59            received_data = pd.concat([received_data, row_df], ignore_index=True)
60            send_to_influxdb(row_df)
61
62    if __name__ == '__main__':
63        app.main()
```

Listing 4.2: Simplified version of an Anomaly Detection App using Isolation Forest in Faust

The main steps of this App are as follows:

- **Imports and setup:** Alongside `faust` and `pandas`, the script uses `IsolationForest` from scikit-learn to perform anomaly detection.
- **Configuration:** Similar to other Apps, runtime parameters are injected via environment variables, allowing flexible deployment with different settings.
- **Faust initialization:** The App is bound to the Kafka topic representing the input Stream. Incoming records are consumed asynchronously by the Faust agent.
- **Model training and updating:** An initial block of data is used to train the first Isolation Forest. The model is periodically retrained on recent data windows to adapt to evolving patterns in the stream.
- **Anomaly detection logic:** Each incoming record is evaluated by the ensemble, producing a score and a binary prediction (normal or anomaly). Predictions are appended to the DataFrame and later written to InfluxDB for visualization.
- **Persistence:** Results are streamed into InfluxDB for real-time monitoring, while periodic snapshots are saved to CSV for offline analysis.

This App highlights how *MultiFlow* enables practical experimentation: the same anomaly detection script can be instantiated multiple times across different Streams, each with distinct parameters such as contamination rate, ensemble size, or retraining interval.

## 4.4 Backend and Orchestration Layer

The backend, implemented in Node.js, orchestrates all services and translates user actions in the interface into system operations. Its responsibilities include:

- Managing Projects, Streams, Apps, and Instances metadata in MongoDB;
- Creating or deleting Kafka topics on demand;
- Deploying and stopping Docker containers for Faust-based App Instances;
- Injecting runtime configuration variables into containers;
- Handling REST API requests and WebSocket communication for real-time updates.

To support these responsibilities, the backend exposes a REST API that the frontend relies on to manage entities such as Projects, Streams, and Instances. Table 4.1 provides an overview of the main backend endpoints currently available, grouped by entity.

Processed outputs are written to InfluxDB, a time-series database optimized for fast inserts and temporal queries. Although used as a temporary store, it provides the performance required for real-time visualization.

Visualization is handled by Grafana, which connects directly to InfluxDB to display dashboards for each running Instance. These dashboards present metrics such as anomaly scores, thresholds, and system statistics, enabling operators to monitor performance or compare scenarios in real time. A detailed example of how Grafana dashboards were used in practice is provided later in the *Validation and Results* chapter (see Section 5.2).

Together, Node.js, InfluxDB, and Grafana form the orchestration and visualization layer, bridging the gap between raw data processing and actionable insights.

## 4.5 Frontend and User Interface

The user interface of *MultiFlow* is implemented in React.js, providing an accessible and responsive way to interact with the tool. It is structured around a sidebar that groups main functionalities, such as:

- **Projects** — organize and configure multiple data streams;
- **Apps** — manage processing scripts and their Instances;
- **Logs** — record changes and user actions for traceability.

Beyond these implemented features, additional interface components are planned for future development. The **Data Sources** tab is envisioned as a shortcut manager for frequently used datasets, allowing users to import and even customize them (e.g., by selecting subsets of columns or rows). The **Dashboards** tab is also expected to support statistics about Instances and models, and potentially incorporate generative features with ethical reasoning mechanisms to support decision-making.

The interface layouts remain consistent across tabs, reducing the learning curve for new users. Active Streams and Instances are also displayed as shortcuts in the sidebar, enabling at-a-glance progress monitoring and quick navigation to details.

The Projects page lists all projects, showing metadata such as name, description, creation date, and last execution timestamp. Each entry can be expanded to reveal its associated Streams, which can then be launched, stopped, or configured. These features are revealed in Figure 4.3.

The Stream creation page allows users to select the simulation parameters, including the topic name and replay speed (rows per second, all-in-seconds, or real-time), as visible in Figure 4.4.

The Stream Details view, represented in Figure 4.5, allows users to check the parameter configurations, as well as the simulation status and quick access to the related Project details page, are also accessible here.

The Apps tab manages Python scripts that implement real-time logic. As shown in Figure 4.6, each App can be expanded to show its Instances, and metadata such as creation date, last update, or description are displayed. The App details view also provides access to container logs for debugging.

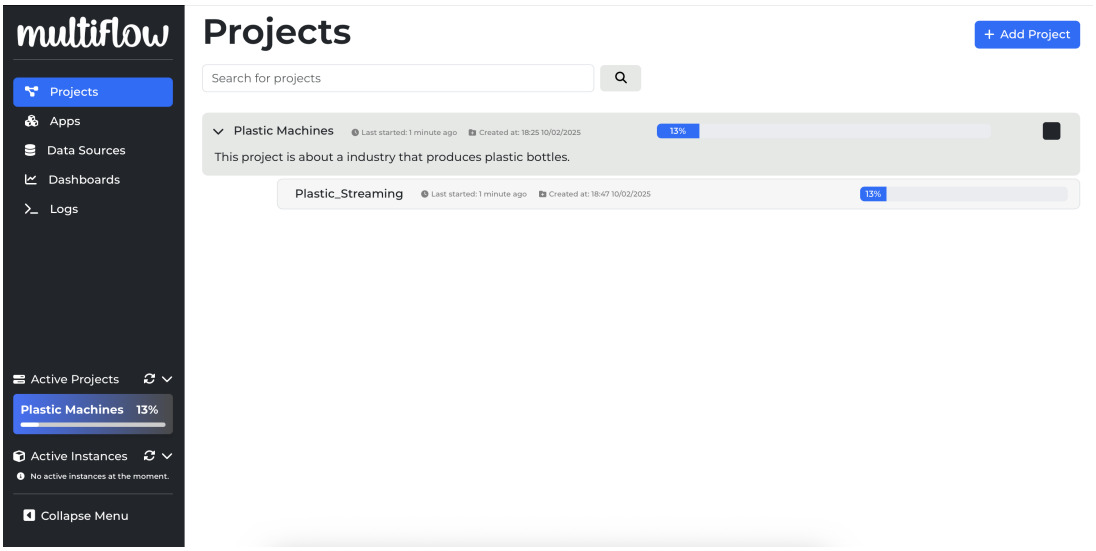



Figure 4.3: *MultiFlow* Projects page with expandable project and stream entries.

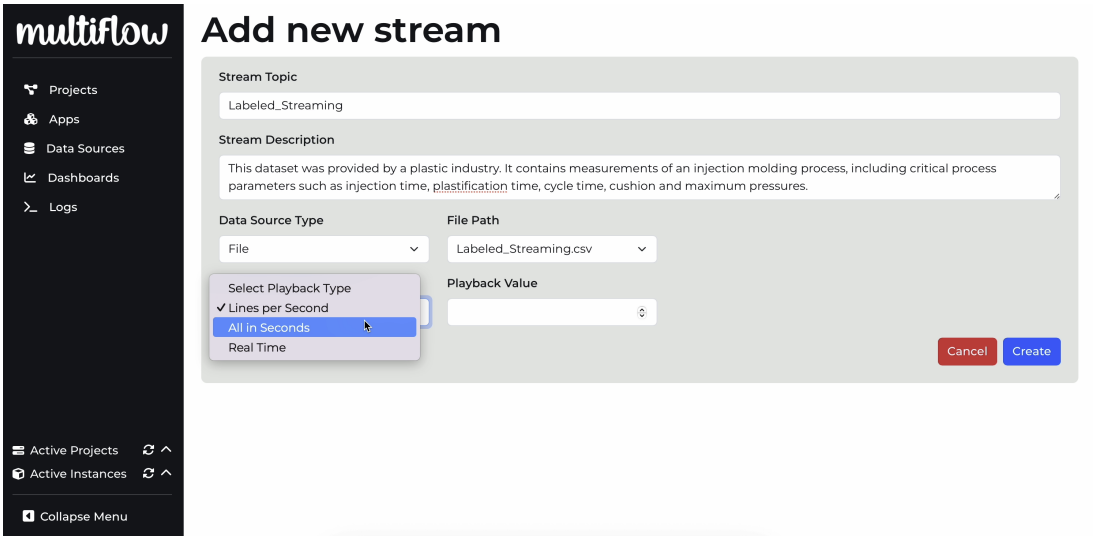



Figure 4.4: Stream creation page with simulation configuration form.

As represented in Figure 4.7, each Instance can be inspected in detail, showing its configuration and runtime logs. This makes it possible to compare multiple configurations of the same App running in parallel.

Additionally, the associated stream topic can be directly visited trough the details page of each Instance, giving quick access to each stream, wether it is for auditing the current streaming parameter, dataset, or changing the status.

As previously mentioned, active projects and instances appear as shortcuts in the sidebar, as seen in Figure 4.8. These display project or instance names with progress indicators, providing quick status feedback without leaving the current page. Clicking a shortcut redirects to the corresponding detail view.

This feature can be minimized by pressing the anchor buttons on each tab (Active Projects and Active Instances), resulting in a more cleaner side bar menu when required.

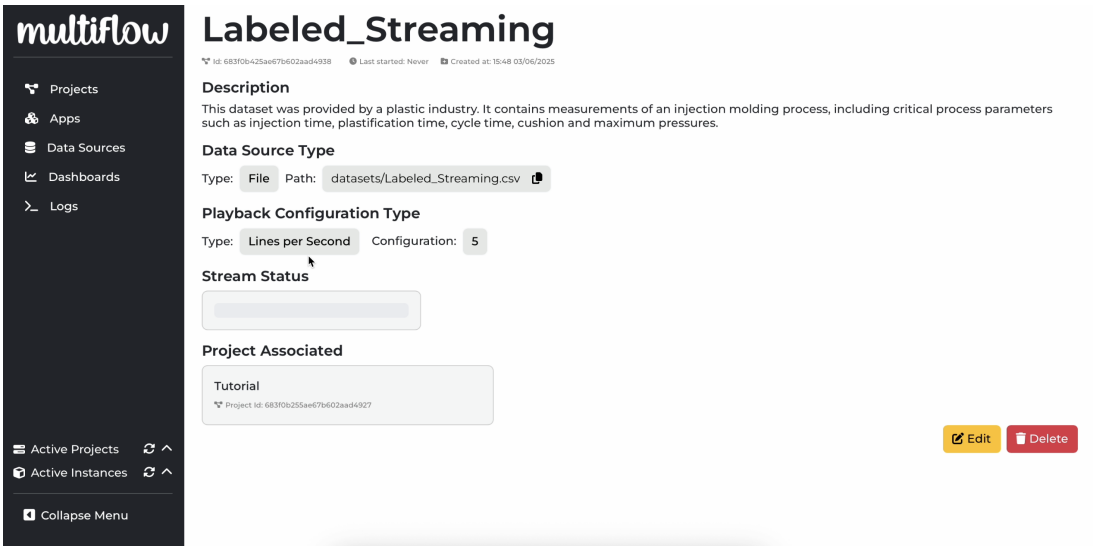



Figure 4.5: Stream details page with simulation information and controls.

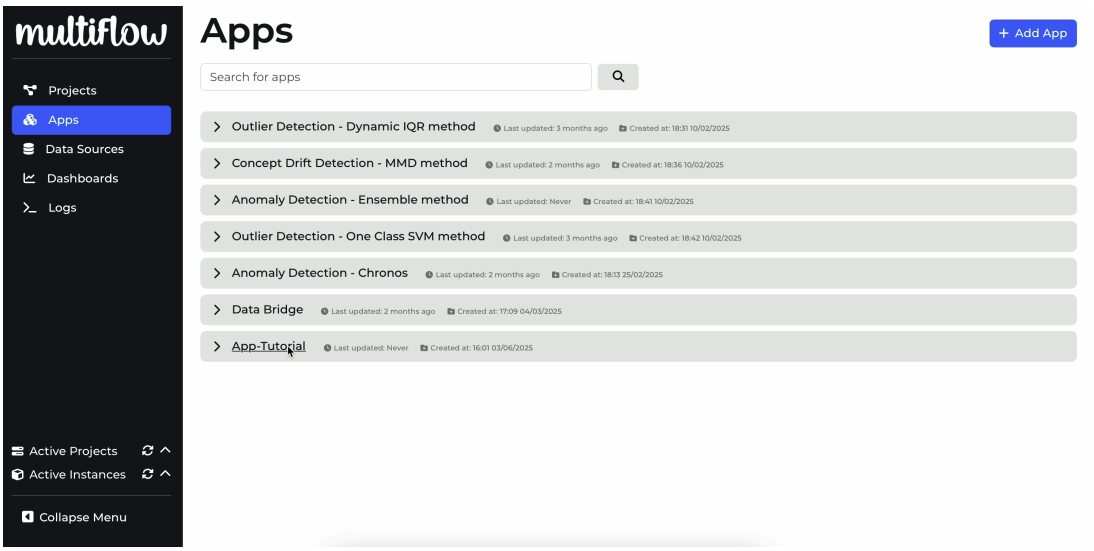



Figure 4.6: *MultiFlow* Apps page with expandable entries and instance controls.

In summary, the screenshots presented here provide a high-level demonstration of *MultiFlow*'s workflow and interface. While they focus on the essential navigation paths, Projects, Streams, Apps, and Instances, the tool offers additional features and configuration options that can be explored in greater depth during practical use. The goal of this overview is therefore not to exhaustively document every detail, but to highlight the core user experience and the central elements that define the tool.

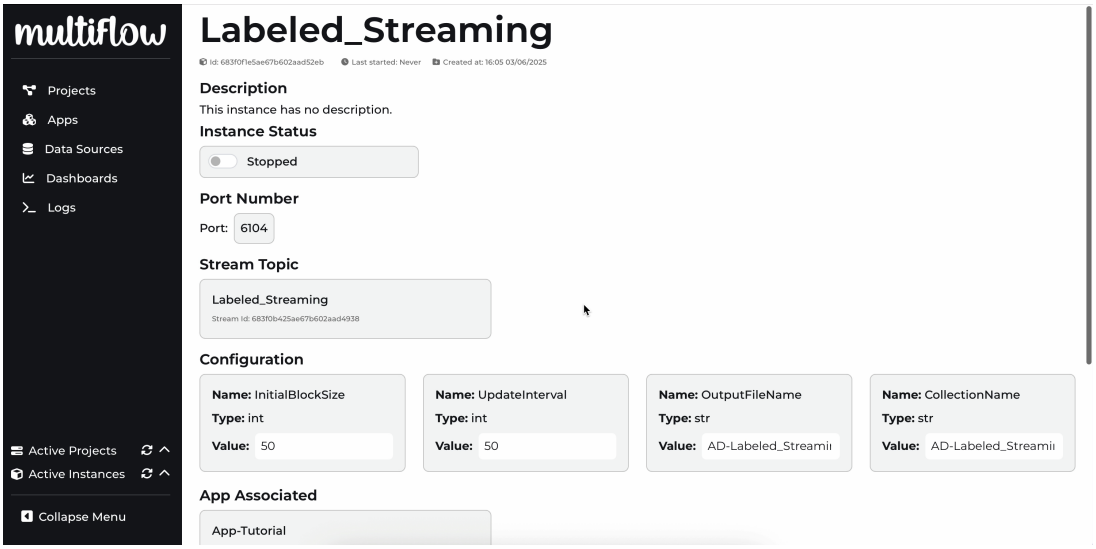



Figure 4.7: Instance details page with configuration, logs, and monitoring links.

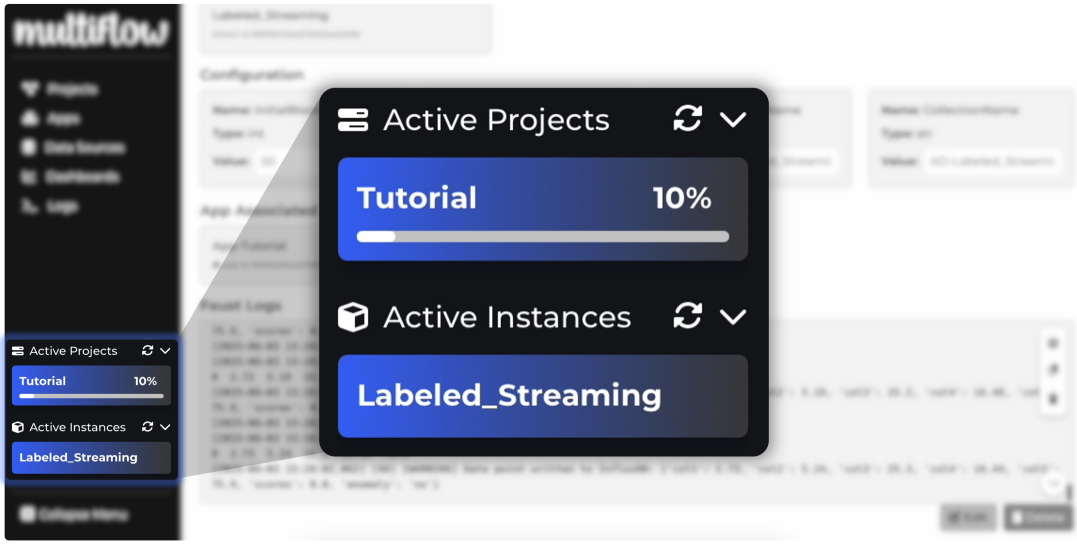



Figure 4.8: Sidebar shortcuts showing active projects and instances with progress tracking.

| Method | Endpoint | Description |
|---|---|---|
| **Projects** | | |
| GET | /projects/search?query=... | Search for projects by name or description. |
| GET | /projects | Retrieve all projects (and their streams). |
| GET | /projects/active | Retrieve active projects. |
| GET | /projects/{id} | Retrieve details of a specific project. |
| POST | /projects | Create a new project. |
| POST | /projects/{id} | Update a project with given ID. |
| PUT | /projects/start/{id} | Start a project. |
| PUT | /projects/stop/{id} | Stop a project. |
| DELETE | /projects/{id} | Delete a project. |
| **Streams** | | |
| GET | /streams/search?query=... | Search for streams by topic or description. |
| GET | /streams/datasets | Retrieve the list of datasets in the folder. |
| GET | /streams | Retrieve all streams. |
| GET | /streams/topics | Retrieve all stream topics. |
| GET | /streams/{id} | Retrieve details of a specific stream. |
| POST | /streams | Create a new stream. |
| POST | /streams/{id} | Update a stream with given ID. |
| PUT | /streams/start/{id} | Start a stream (automatically triggered by starting the respective project). |
| PUT | /streams/stop/{id} | Stop a stream (automatically triggered by stoping the respective project). |
| DELETE | /streams/{id} | Delete a stream. |
| **Apps** | | |
| GET | /apps/search?query=... | Search for apps by name, description, or python file. |
| GET | /apps/code | Retrieve the list of python files in the folder. |
| GET | /apps | Retrieve all apps (and their instances). |
| GET | /apps/{id} | Retrieve details of a specific app. |
| POST | /apps | Create a new app. |
| POST | /apps/run/{filename} | Run a specific Python file. |
| POST | /apps/{id} | Update an app with given ID. |
| DELETE | /apps/{id} | Delete an app. |
| **Instances** | | |
| GET | /instances/search?query=... | Search for instances by name or description. |
| GET | /instances | Retrieve all instances. |
| GET | /instances/active | Retrieve active instances. |
| GET | /instances/{id} | Retrieve details of a specific instance. |
| POST | /instances | Create a new instance. |
| POST | /instances/start/{id} | Start a specific Faust Application. |
| POST | /instances/stop/{id} | Stop a specific Faust Application. |
| PUT | /instances/{id} | Update instance configuration with given ID. |
| DELETE | /instances/{id} | Delete an instance. |

Table 4.1: Summary of available API endpoints for the *MultiFlow* backend.

# Chapter 5

# Validation

This chapter presents the validation of *MultiFlow* through selected use cases, demonstrating its ability to simulate industrial environments and support the development and evaluation of streaming-based AI methods. Rather than focusing solely on technical implementation, this chapter highlights how the tool performs when applied to real-world inspired challenges. Each use case describes the industrial motivation, explains the experimental setup, and discusses how *MultiFlow* enabled testing and analysis. Results are illustrated using both dashboards and outputs generated by the applications running within the tool.

## 5.1 Use Case 1: Reusing Previously Trained Models in New ML Tasks

A recurring challenge in industrial AI is the efficient training of machine learning models under dynamic conditions [17, 10, 51]. In many manufacturing contexts, new problems, such as emerging defects or quality-control needs, often trigger the training of models from scratch, even when relevant knowledge already exists from prior tasks [30]. This results in wasted computational resources, unnecessary labeling effort, and slower deployment.

To address this, recent work proposed a method for reusing previously trained models in new tasks by leveraging data similarity. The core idea is to treat incoming streaming data in small windows, summarize each window using statistical *meta-features* [52, 53], and compare these to past data via a similarity metric (Bray–Curtis dissimilarity) [30, 17]. If a match is found, a previously trained model is reused; otherwise, a new model is trained. Selected models are combined in a rotating ensemble, ensuring that the system adapts to new conditions while avoiding redundant retraining.

As shown in Table 5.1, this approach achieved high accuracy in six datasets (five synthetic and one real). Importantly, it was able to reuse prior models in 4% to 52% of the batches, instead of always training new ones (the traditional approach). This not only reduced computational cost but also improved the sustainability of the learning process compared to the traditional approach of training one model per batch.

*MultiFlow* played a crucial role in enabling the experiments behind this approach. Specifically:

Table 5.1: Results of the proposed method (P) compared to a traditional approach (T).

| Metric | LineA | | LineB | | LineC | | LineD | | LineE | | Real Molding | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **P** | **T** | **P** | **T** | **P** | **T** | **P** | **T** | **P** | **T** | **P** | **T** |
| Pr. | 1,000 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,000 |
| Rec. | **0,889** | 0,722 | **1,00** | 0,920 | 1,00 | 1,00 | **1,00** | 0,60 | **1,00** | 0,720 | **1,00** | 0,989 |
| FS | **0,941** | 0,839 | **1,00** | 0,958 | 1,00 | 1,00 | **1,00** | 0,75 | **1,00** | 0,837 | **1,00** | 0.995 |
| Acc. (%) | **0,9998** | 0,9995 | **1,00** | 0.9992 | 1,00 | 1,00 | **1,00** | 0,9988 | **1,00** | 0,9986 | **1,00** | 0,9997 |
| R (%) | 52,00 | – | 12,00 | – | 4,00 | – | 48,00 | – | 32,00 | – | 41,38 | – |

- It provided a flexible environment for simulating industrial data streams, both synthetic and real, enabling controlled experimentation with evolving datasets.

- Through its App–Instance structure, it allowed researchers to run multiple configurations in parallel (e.g., varying the number of models in the ensemble or the thresholds for reuse), supporting fast iteration.

- By streaming outputs into Grafana dashboards, *MultiFlow* made it possible to monitor the method's real-time behavior. In this work, additional charts were also generated directly from the Python scripts used within Faust, showing that results can be visualized both externally and within the tool.

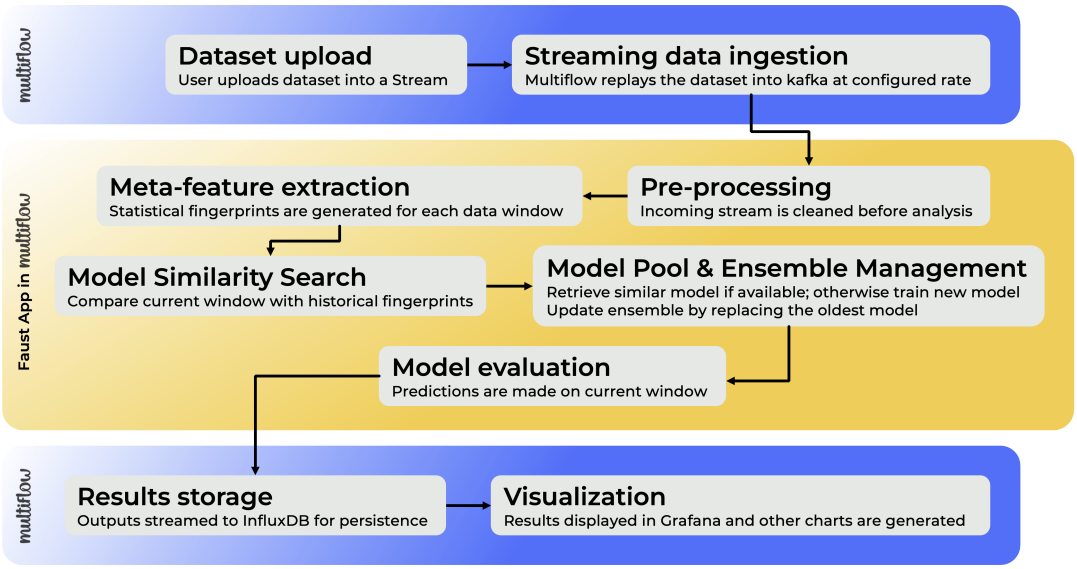



Figure 5.1: Overview of the model reuse method proposed in the use case, tested using *MultiFlow*. (Source: author)

This use case validates *MultiFlow*'s capacity to support complex machine learning workflows in streaming environments, allowing rapid experimentation and realistic validation of methods with direct industrial relevance.

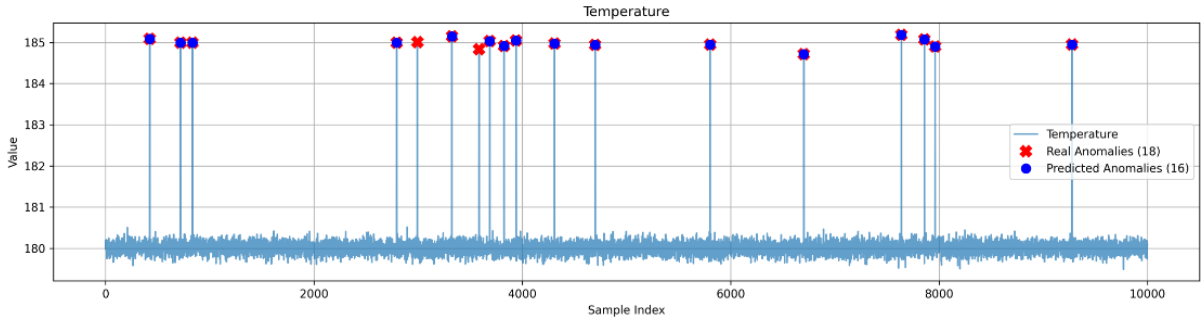



Figure 5.2: Example of results generated by the Faust App implementing the model reuse approach. Extracted from the script executed in *MultiFlow*, as one of the outputs. (Source: author)

## 5.2 Use Case 2: Ensemble-Based Anomaly Detection in Streaming Data

Anomaly detection is one of the most critical applications of industrial AI, as unexpected deviations in sensor readings can indicate faults, degradation, or safety issues [44, 1, 3, 16]. In this use case, we explored the ability of *MultiFlow* to support experimentation with ensemble-based anomaly detection under streaming conditions.

The method employed relies on an ensemble of *Isolation Forest* models, a well-known algorithm for unsupervised anomaly detection [15, 16]. Although the original datasets used for validation contained ground-truth labels, these were intentionally removed during the streaming and processing phases. Labels were only reintroduced at the end of the pipeline, enabling evaluation of the anomaly scores produced by the ensemble without influencing the detection logic. This setup reflects realistic industrial scenarios, where labels are usually unavailable in real time.

*MultiFlow* was instrumental in testing the generality of this approach. By uploading multiple datasets as independent Streams, we were able to create and configure several Instances of the same anomaly detection App. Each Instance subscribed to a different Kafka topic and ran with custom parameters, such as the number of Isolation Forest models in the ensemble, thresholds, and decision strategies. This setup allowed direct comparison of the algorithm's behavior across heterogeneous data sources, demonstrating both its robustness and its sensitivity to parameter tuning.

The results of this use case were observed in two complementary ways. First, *Grafana* dashboards (see Figure 5.3) displayed real-time anomaly scores and detection rates, providing immediate feedback during simulations. Second, the same App scripts also produced Python-generated charts for deeper offline analysis, two of which are shown in Figure 5.4. Together, these outputs highlight the dual capability of *MultiFlow* to deliver both online and offline insights.

Overall, this use case illustrates how *MultiFlow* enables rapid experimentation with anomaly detection algorithms in streaming environments. By combining flexible configuration of Instances with real-time and offline visualization, it supports both iterative model tuning and systematic evaluation of generalization across datasets.

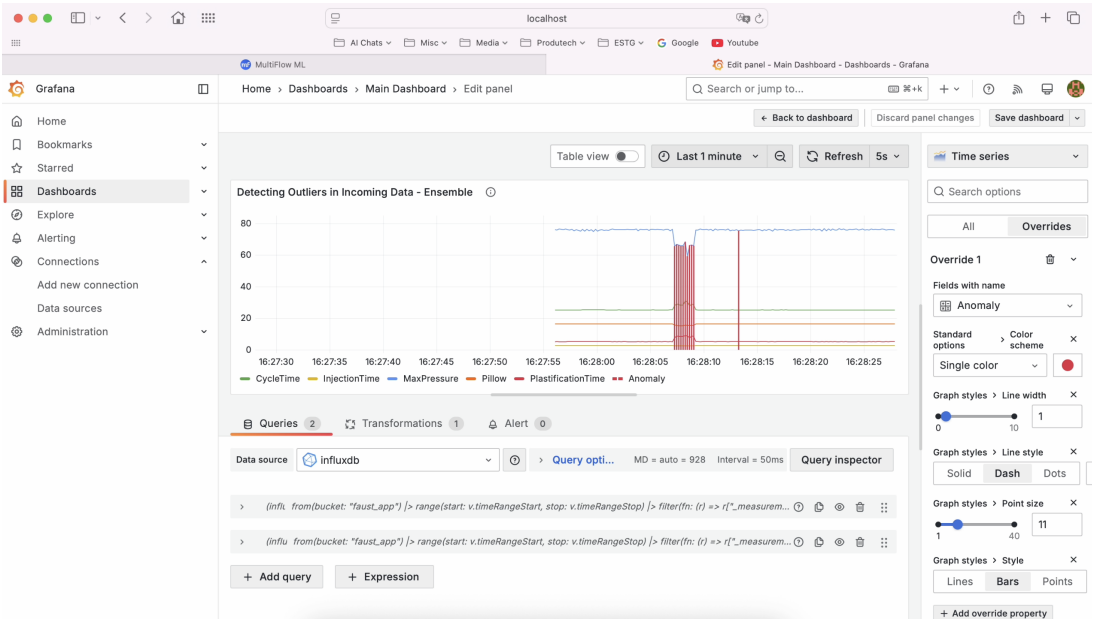



Figure 5.3: Real-time Grafana dashboard showing anomaly scores from an ensemble of Isolation Forest models. (Source: author)

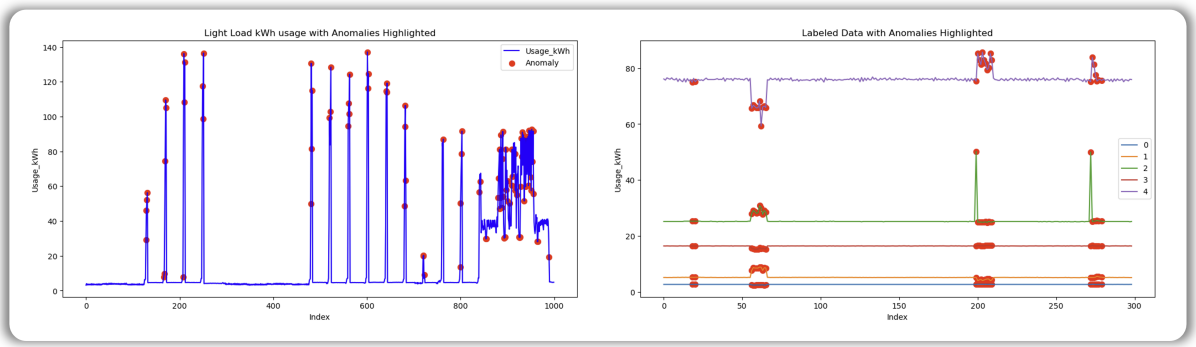



Figure 5.4: Python-generated chart of the anomaly detection results across different App Instances with varied configurations. (Source: author)

## 5.3 Use Case 3: Concept Drift Detection in Streaming Data

One of the recurring challenges in industrial AI is the presence of *concept drift*, where the statistical properties of sensor data evolve over time [43, 44, 1]. Drift may be caused by gradual equipment wear, seasonal effects, or sudden changes in operating conditions, all of which can degrade the accuracy of predictive models if not addressed [30, 17].

In this use case, MultiFlow was employed to simulate multiple data streams containing controlled drifts. Historical datasets were replayed at configurable rates, and drift detection algorithms were deployed as Apps consuming these streams. Each Instance corresponded to a different drift detection strategy (e.g., statistical tests, error-rate monitoring, adaptive ensembles), enabling a side-by-side comparison of performance.

The main advantage of using MultiFlow in this context was its ability to create reproducible scenarios where different drift types could be injected or simulated. The

Kafka–Faust pipeline ensured that detection results were produced in real time, while Grafana dashboards allowed for immediate visualization of drift alarms, delays, and false positives (see Figure 5.5). This setup provided valuable insights into which methods were most robust under realistic streaming conditions.


Figure 5.5: Real-time Grafana dashboard showing Concept Drift detection results from one of the algorithms. (Source: author)

## 5.4 Use Case 4: Benchmarking Algorithms Across Industrial Scenarios

Another practical challenge for practitioners is assessing the generality of machine learning methods across different industrial datasets [54]. A model tuned for one scenario may not perform equally well when applied to another, which raises questions about robustness and transferability.

This use case focused on benchmarking multiple algorithms on streams derived from different industrial-like datasets. MultiFlow was particularly valuable because it allowed users to instantiate the same App script across multiple Streams simultaneously. Each Instance ran with specific parameters (e.g., thresholds, ensemble size, feature selection options), enabling broad experimentation without duplicating code.

The results of each Instance were collected in InfluxDB and visualized in Grafana, where researchers could compare metrics across datasets in real time. By centralizing both configuration and monitoring, MultiFlow reduced the engineering effort typically needed to benchmark algorithms, while also ensuring that experiments remained reproducible. This highlights the tool's strength in supporting systematic evaluations under heterogeneous data conditions.

## 5.5 Use Case 5: Evaluating Frugal AI Strategies

A final use case explored MultiFlow's ability to support the development and evaluation of *Frugal AI* methods. In industrial environments, resource efficiency is often as important

as predictive accuracy, particularly when deploying on constrained edge devices or when minimizing energy costs is a priority [46, 30, 17].

In this scenario, multiple lightweight anomaly detection and drift detection algorithms were deployed as Apps within MultiFlow. Each Instance was configured with different resource-aware parameters (e.g., reduced model complexity, sampling strategies, or simplified feature extraction). The outputs were not only evaluated in terms of accuracy but also monitored for latency and computational overhead, providing a holistic view of trade-offs.

MultiFlow played a key role by simulating realistic streaming conditions, allowing frugal approaches to be tested under load before deployment. Grafana dashboards made it possible to contrast detection performance against system metrics such as processing time per message or memory footprint. This approach demonstrated how MultiFlow can help bridge the gap between academic research on efficient AI and its practical deployment in industrial settings.

# Summary

The validation of *MultiFlow* through multiple use cases, including model reuse, anomaly detection, concept drift, benchmarking, and frugal AI, highlights its versatility as both a research and industrial tool. Across these scenarios, the tool demonstrated its ability to simulate realistic data streams, support rapid experimentation through multiple Instances, and provide real-time insights via dashboards. While each use case focused on a specific challenge, together they emphasize the broader applicability of *MultiFlow* as a digital twin environment for testing, comparing, and refining AI-driven solutions in streaming industrial contexts.

# Chapter 6

# Results

This chapter presents the broader outcomes of the work developed throughout this dissertation. While the previous chapter focused on validating *MultiFlow* through concrete use cases and experiments, here the main results obtained are highlighted at both the technical and scientific level.

## 6.1 MultiFlow Tool

The main technical result of this dissertation is the development of *MultiFlow*, a modular, containerized tool for real-time data analysis and simulation. Its core features include:

- Simulation of real-time streams from historical or live data sources;
- Orchestration of Apps and Instances with configurable parameters;
- Real-time visualization of results through Grafana dashboards;
- Modular extensibility via Dockerized services and open APIs.

*MultiFlow* bridges research and industry by offering a frugal but effective tool for experimentation in streaming AI. The source code and documentation are publicly available at:

`https://github.com/davidecarneiro/multiflow`

Figure 6.1 shows the public repository and its accompanying documentation.

## 6.2 Industry Collaboration

Another significant outcome is the collaboration with Muvu Technologies[1], an industry partner. *MultiFlow* supported anomaly detection and concept drift experiments in realistic settings, providing the company with a flexible testbed for MLOps experimentation. Their feedback also helped refine design decisions, increasing the practical relevance of the tool.

[1] https://muvu.tech

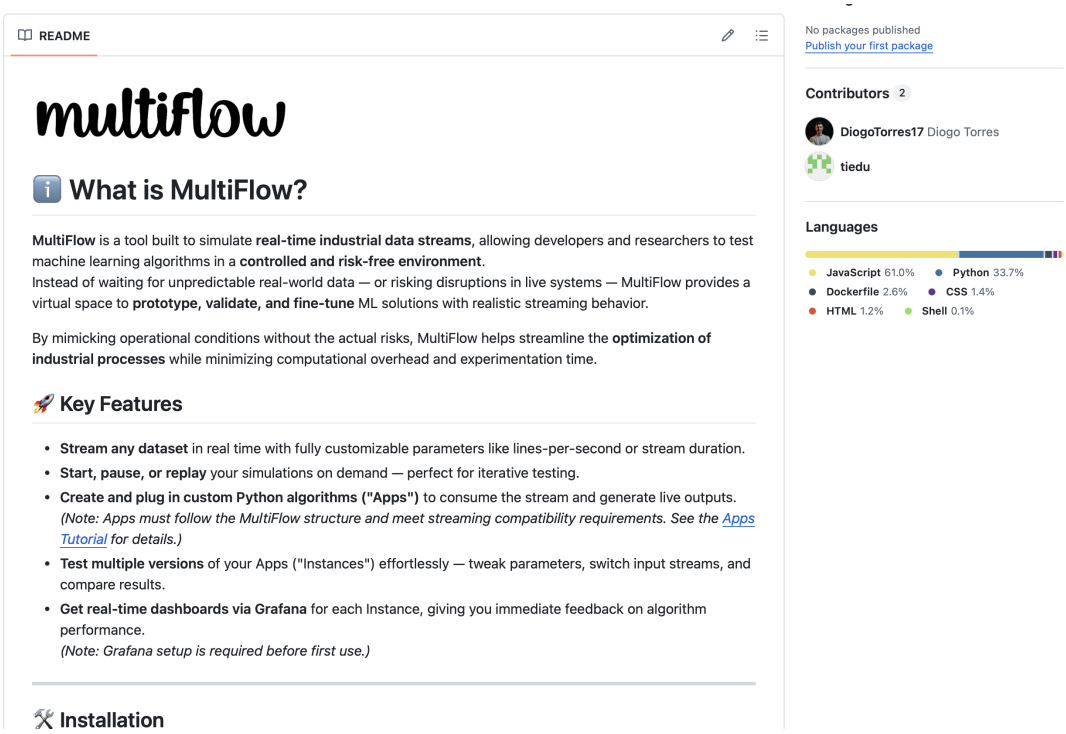



Figure 6.1: Public GitHub repository of *MultiFlow*, including setup instructions and documentation.

## 6.3 Scientific Publications

The work conducted throughout this dissertation resulted in a series of scientific publications in international conferences and journals. These publications disseminated both the *MultiFlow* tool and related methodologies in MLOps, anomaly detection, and model reuse.

1. E. Peixoto, **D. Torres**, D. Carneiro, B. Silva, and P. Novais, "Efficient MLOps: Meta-learning meets Frugal AI", in Proc. 2nd European Symposium on Artificial Intelligence in Manufacturing (ESAIM 2024), Greece, 2024. **Abstract:** The advent of large Machine Learning models and the steep increase in the demand for AI solutions occurs at the same point in time in which policies are being enacted to implement more sustainable processes in virtually every sector. This means there is a need for more, better and larger models, which require significant computational resources, while at the same time a call for a decrease in the energy spent in the processes associated to MLOps. In this paper we propose a reduced set of meta-features that can be used to characterize sets of data and their relationship with model performance. We start from a large set of 66 features, and reduce it to only 10 while maintaining the strength of this relationship. This ensures a process of meta-feature extraction and prediction of model performance that is in line with the desiderata of Frugal AI, allowing to develop more efficient ML processes.

2. E. Peixoto, D. Carneiro, **D. Torres**, B. Silva, and P. Novais, "Reusing past Machine Learning models based on data similarity metrics", in Proc. 13th International Symposium on Ambient Intelligence (ISAMI 2024), Salamanca, Spain, 2024. **Abstract:** Many of today's domains of application of Machine Learning (ML) are dynamic in

the sense that data and their patterns change over time. This has a significant impact in the ML lifecycle and operations, requiring frequent model (re-)training, or other strategies to deal with outdated models and data. This need for dynamic and responsive solutions also has an impact on the use of computational resources and, consequently, on sustainability indicators. This paper proposes an approach in line with the concept of Frugal AI, whose main aim is to minimize the resources and time spent on training models by re-using models from a pool of past models, when appropriate. Specifically, we present and validate a methodology for similarity-based model selection in data streaming environments with concept drift. Rather than training a new model for each new block of data, this methodology considers a pool with only a subset of the models and, for each new block of data, will select the best model from the pool. The best model is determined based on the distance between its training data and the current block of data. Distance is calculated based on a set of meta-features that characterizes the data, and on the Bray-Curtis distance. We show that it is possible to reuse previous models using this methodology, leading to potentially significant saving of resources and time, while maintaining predictive quality.

3. Peixoto, E., **Torres, D.**, Carneiro, D., Silva, B., & Marques, R. (2025). "Reusing ML Models in Dynamic Data Environments: A Data Similarity-Based Approach for Efficient MLOps". *Big Data and Cognitive Computing*, MDPI, 2025. **Abstract:** The rapid integration of Machine Learning (ML) in organizational practices has driven demand for substantial computational resources, incurring both high economic costs and environmental impact, particularly from energy consumption. This challenge is amplified in dynamic data environments, where ML models must be frequently retrained to adapt to evolving data patterns. To address this, more sustainable Machine Learning Operations (MLOps) pipelines are needed for reducing environmental impacts while maintaining model accuracy. In this paper, we propose a model reuse approach based on data similarity metrics, which allows organizations to leverage previously trained models where applicable. We introduce a tailored set of meta-features to characterize data windows, enabling efficient similarity assessment between historical and new data. The effectiveness of the proposed method is validated across multiple ML tasks using the cosine and Bray–Curtis distance functions, which evaluate both model reuse rates and the performance of reused models relative to newly trained alternatives. The results indicate that the proposed approach can reduce the frequency of model retraining by up to 70% to 90% while maintaining or even improving predictive performance, contributing to more resource-efficient and sustainable MLOps practices.

4. **Torres, D.**, Peixoto, E., Carneiro, D., Palumbo, G., & Alves, V. (2025). "MultiFlow: An Ambient Intelligence Digital Twin." ISaMI, 2025. **Abstract:** Ambient intelligence (AmI) refers to environments where smart devices, sensors, and AI-driven systems work seamlessly to enhance human interactions with their surroundings. Through the combination of real-time data, context-awareness, and adaptive learning, AmI enables environments to respond proactively to user needs, improving efficiency, comfort, and decision-making. However, since AmI systems are inherently human-centric and often operate autonomously, they must be designed with robust ethical, privacy, and safety considerations. Ensuring that these systems function reliably, fairly, and without harm is crucial, especially in sensitive domains like

healthcare, security, and smart infrastructure. This work introduces a novel tool, conceptualized as an AmI Digital Twin, which allows developers to simulate or monitor AmI data streams, and develop and thoroughly test AmI applications before and during their real use. Built on a modular architecture leveraging technologies like React.js, Node.js, Kafka, Faust, MongoDB, InfluxDB, Grafana, and Docker, the tool ensures adaptability to different application environments, scalability, and ease of deployment. Besides the description of the tool itself, we provide some early validation results in common AmI tasks such as anomaly and concept drift detection. The tool is available in a public repository, and comes pre-packaged with a set of applications for AmI use-cases.

5. Peixoto, E., Carneiro, **D., Torres**, D., Silva, B., & Marques, R. (2025). "Towards generalizable machine learning pipelines in complex industrial scenarios". The 5th IEEE International Workshop on Distributed Intelligent Systems, 2025. **Abstract:** The increasing prevalence of ML in industrial environments is driven by the growing availability of user-friendly frameworks and industrial data. Manufacturing Execution Systems (MES) enabled easy data collection and utilization for decision support, namely for anomaly detection, quality control, or object detection/classification. However, models for new ML problems are often trained without regard for previous models or data, potentially wasting resources and hindering knowledge transfer. This is due to a lack of systematic methods for identifying and leveraging relevant prior knowledge. In this paper, we propose an approach designed to address this inefficiency by reusing previously trained models in new ML tasks. We reuse models based on data similarity metrics to create ensembles on-the-fly. This allows for accurate predictions on new data while minimizing the need for training from scratch. This approach has the potential to significantly reduce resource expenditure on data labeling and model training within industrial organizations.

6. Peixoto, E., **Torres, D.**, Carneiro, D., Silva, B., & Marques, R. (2025). "Using transfer learning to minimize data labeling in anomaly detection: a comparative analysis". 3rd European Symposium on Artificial Intelligence in Manufacturing, 2025. **Abstract:** The need for sufficient high-quality labeled data is often one of the barriers of industrial organizations to adopt Machine Learning (ML) in their processes. This happens for various reasons, including the cost in human resources, the lack of expertise, the lack of dedicated personnel, or the cost of collecting and handling data. Another barrier is the lack of knowledge regarding the processes through which ML models are trained, evaluated, served and monitored. Minimizing human dependence is thus paramount to make these technologies available to organizations. In this paper, we propose a novel approach for anomaly detection in manufacturing that minimizes both the need for labeled data and the need to train new models, thus minimizing human and technical resource demands. This approach is based on the notion of reusing past trained models based on data similarity metrics, to build ensembles of models that are considered the most suited at a given moment, given the current properties of the data. It is thus especially suited for data streaming scenarios, namely with concept drift. We compare this approach with a state-of-the-art Transformer-based one, in terms of predictive performance and computational efficiency. This work contributes to the simplification of MLOps pipelines in manufacturing domains, thus fostering technology adoption.


Figure 6.2: Presentation of the paper *Towards Generalizable Machine Learning Pipelines in Complex Industrial Scenarios* at DistInSys 2025, Bologna, Italy.

## 6.4 Personal Contributions

In addition to the technical development of *MultiFlow*, this dissertation contributed to the following achievements:

- **Scientific impact:** Co-authorship of six publications, including conference papers and a journal article.
- **International exposure:** Participation in major events such as ESAIM 2024 (Greece), ISAMI 2024 (Spain), and DistInSys 2025 (Italy).
- **Open science:** Development of an open-source tool with freely available source code and documentation.
- **Industry collaboration:** Active partnership with Muvu Technologies, enabling the validation of real-world use cases.

## Summary

In summary, the development of *MultiFlow*, the collaboration with industrial partners, and the resulting scientific publications demonstrate the relevance of this work both in academic and applied contexts. These results validate not only the tool itself but also its role as a catalyst for further research in real-time AI in industrial environments.

# Chapter 7

# Conclusion

This dissertation set out to address a clear gap in the technological landscape: the lack of tools that enable the rapid prototyping and testing of machine learning and data analysis algorithms under realistic streaming conditions. While many existing tools support either batch-oriented pipelines or production-level orchestration, few are designed to combine simulation, experimentation, and evaluation in one integrated environment.

To fill this gap, the *MultiFlow* tool was developed. *MultiFlow* provides an environment for simulating real-time industrial data streams, deploying processing applications, and visualizing results through interactive dashboards. Its modular architecture, spanning Kafka for streaming, Faust for real-time processing, InfluxDB and Grafana for visualization, Node.js for orchestration, and React.js for the user interface, allows users to configure and test algorithms in a controlled but realistic setting.

The validation work, carried out through use cases inspired by industrial partners such as Muvu Technologies, demonstrates the tool's applicability. Applications for anomaly detection and concept drift, among others, confirmed that *MultiFlow* can support experimentation with challenges that frequently arise in industrial contexts. By allowing users to create projects, streams, apps, and instances, the tool facilitates iterative testing and highlights its value as a bridge between research prototypes and operational environments.

This work contributes both a practical artifact and a conceptual framework for approaching industrial MF-Workflow experimentation. It shows that research-oriented tools can be engineered to align with industrial needs, enabling controlled exploration of adaptive algorithms while reducing the risks and costs of testing on live production systems.

At the same time, the tool has limitations. *MultiFlow* has not yet been tested on a scale or in production environments, and usability studies remain pending. Future work should focus on extending the tool to support more advanced streaming parameters and customization options, enhancing the performance to handle larger and more complex data streams with lower latency, and conducting systematic evaluations with end-users.

In summary, *MultiFlow* demonstrates that it is possible to build an accessible and extensible tool for industrial MF-Workflow experimentation by combining open-source, containerized technologies. By enabling safe and repeatable testing of real-time algorithms, it supports the advancement of adaptive, frugal, and trustworthy MF-Workflow solutions in industrial domains.

# Chapter 8

# Bibliography


[1] M. Javaid, A. Haleem, R. P. Singh, and R. Suman, "Artificial intelligence applications for industry 4.0: A literature-based study," *Journal of Industrial Integration and Management*, vol. 7, no. 01, pp. 83–111, 2022.

[2] A. A. Alnaser and H. Elmousalami, "Benefits and challenges of ai-based digital twin integration in the saudi arabian construction industry: A correspondence analysis (ca) approach," *Applied Sciences*, vol. 15, no. 9, p. 4675, 2025.

[3] D. Jain, "Artificial intelligence in quality control systems: A cross-industry analysis of applications, benefits, and implementation frameworks," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 10, no. 6, pp. 1321–1333, 2024.

[4] A. E. H. Gabsi, "Integrating artificial intelligence in industry 4.0: insights, challenges, and future prospects–a literature review," *Annals of Operations Research*, pp. 1–28, 2024.

[5] A. Almeida, S. Brás, S. Sargento, and F. C. Pinto, "Time series big data: a survey on data stream frameworks, analysis and algorithms," *Journal of Big Data*, vol. 10, no. 1, p. 83, 2023.

[6] A. Elhanashi, P. Dini, S. Saponara, and Q. Zheng, "Integration of deep learning into the iot: A survey of techniques and challenges for real-world applications," *Electronics*, vol. 12, no. 24, p. 4925, 2023.

[7] T. Burns, J. Cosgrove, and F. Doyle, "A review of interoperability standards for industry 4.0.," *Procedia Manufacturing*, vol. 38, pp. 646–653, 2019.

[8] J. M. Tien, "Internet of things, real-time decision making, and artificial intelligence," *Annals of Data Science*, vol. 4, no. 2, pp. 149–178, 2017.

[9] M. Bahri, A. Bifet, J. Gama, H. M. Gomes, and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 11, no. 3, p. e1405, 2021.

[10] L. Faubel, K. Schmid, and H. Eichelberger, "Mlops challenges in industry 4.0," *SN Computer Science*, vol. 4, no. 6, p. 828, 2023.

[11] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos, "A survey on the evolution of stream processing systems," *The VLDB Journal*, vol. 33, no. 2, pp. 507–541, 2024.

[12] M. Pacella, A. Papa, G. Papadia, and E. Fedeli, "A scalable framework for sensor data ingestion and real-time processing in cloud manufacturing," *Algorithms*, vol. 18, no. 1, p. 22, 2025.

[13] M. Straat, K. Koster, N. Goet, and K. Bunte, "An industry 4.0 example: Real-time quality control for steel-based mass production using machine learning on non-invasive sensor data," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 01–08, IEEE, 2022.

[14] A. Ashfahani, M. Pratama, E. Lughofer, and E. Y. K. Yee, "Autonomous deep quality monitoring in streaming environments," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2021.

[15] S. F. Chevtchenko, E. D. S. Rocha, M. C. M. Dos Santos, R. L. Mota, D. M. Vieira, E. C. De Andrade, and D. R. B. De Araújo, "Anomaly detection in industrial machinery using iot devices and machine learning: a systematic mapping," *IEEE Access*, vol. 11, pp. 128288–128305, 2023.

[16] M. A. Panza, M. Pota, and M. Esposito, "Anomaly detection methods for industrial applications: A comparative study," *Electronics*, vol. 12, no. 18, p. 3971, 2023.

[17] E. Peixoto, D. Torres, D. Carneiro, B. Silva, and R. Marques, "Reusing ml models in dynamic data environments: Data similarity-based approach for efficient mlops," *Big Data and Cognitive Computing*, vol. 9, no. 2, p. 47, 2025.

[18] S. Valliappan, P. Bagavathi Sivakumar, and V. Ananthanarayanan, "Efficient real-time decision making using streaming data analytics in iot environment," in *International Conference on Advanced Computing Networking and Informatics: ICANI-2018*, pp. 165–173, Springer, 2018.

[19] T. V. Iyelolu, E. E. Agu, C. Idemudia, and T. I. Ijomah, "Driving sme innovation with ai solutions: overcoming adoption barriers and future growth opportunities," *International Journal of Science and Technology Research Archive*, vol. 7, no. 1, pp. 036–054, 2024.

[20] A. NiFi, "Apache nifi overview," 2020.

[21] K. Wnkek and P. Borylo, "A data processing and distribution system based on apache nifi," in *Photonics*, vol. 10, p. 210, MDPI, 2023.

[22] H. Isah and F. Zulkernine, "A scalable and robust framework for data stream ingestion," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2900–2905, IEEE, 2018.

[23] S. Dwivedi, P. Kasliwal, and S. Soni, "Comprehensive study of data analytics tools (rapidminer, weka, r tool, knime)," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pp. 1–8, IEEE, 2016.

[24] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "A comparison on scalability for batch big data processing on apache spark and apache flink," *Big Data Analytics*, vol. 2, no. 1, p. 1, 2017.

[25] E. Nazari, M. H. Shahriari, and H. Tabesh, "Bigdata analysis in healthcare: apache hadoop, apache spark and apache flink," *Frontiers in Health Informatics*, vol. 8, no. 1, p. 14, 2019.

[26] A. Bodor, M. Hnida, and D. Najima, "From development to deployment: An approach to mlops monitoring for machine learning model operationalization," in *2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pp. 1–7, IEEE, 2023.

[27] F. Möller, I. Jussen, V. Springer, A. Gieß, J. C. Schweihoff, J. Gelhaar, T. Guggenberger, and B. Otto, "Industrial data ecosystems and data spaces," *Electronic Markets*, vol. 34, no. 1, p. 41, 2024.

[28] J. Nilsson and F. Sandin, "Semantic interoperability in industry 4.0: Survey of recent developments and outlook," in *2018 IEEE 16th international conference on industrial informatics (INDIN)*, pp. 127–132, IEEE, 2018.

[29] K. Lakshmanna, R. Kaluri, N. Gundluru, Z. S. Alzamil, D. S. Rajput, A. A. Khan, M. A. Haq, and A. Alhussen, "A review on deep learning techniques for iot data," *Electronics*, vol. 11, no. 10, p. 1604, 2022.

[30] E. Peixoto, D. Torres, D. Carneiro, B. Silva, and P. Novais, "Efficient mlops: Meta-learning meets frugal ai," in *European Symposium on Artificial Intelligence in Manufacturing*, pp. 253–261, Springer, 2024.

[31] M. Javaid, A. Haleem, R. P. Singh, R. Suman, and E. S. Gonzalez, "Understanding the adoption of industry 4.0 technologies in improving environmental sustainability," *Sustainable operations and computers*, vol. 3, pp. 203–217, 2022.

[32] S. Khare and M. Totaro, "Big data in iot," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–7, IEEE, 2019.

[33] D. P. Möller, H. Vakilzadian, and R. E. Haas, "From industry 4.0 towards industry 5.0," in *2022 IEEE international conference on electro information technology (eIT)*, pp. 61–68, IEEE, 2022.

[34] I. G. A. Premananda, A. Tjahyanto, and A. Mukhlason, "Design science research methodology and its application to developing a new timetabling algorithm," in *2022 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, pp. 433–438, IEEE, 2022.

[35] J. Antony, M. Sony, B. Lameijer, S. Bhat, R. Jayaraman, and L. Gutierrez, "Towards a design science research (dsr) methodology for operational excellence (opex) initiatives," *The TQM Journal*, vol. 36, no. 8, pp. 2383–2397, 2024.

[36] F. Wortmann and K. Flüchter, "Internet of things: technology and value added," *Business & information systems engineering*, vol. 57, no. 3, pp. 221–224, 2015.

[37] A. Čolaković and M. Hadžialić, "Internet of things (iot): A review of enabling technologies, challenges, and open research issues," *Computer networks*, vol. 144, pp. 17–39, 2018.

[38] S. H. Shah and I. Yaqoob, "A survey: Internet of things (iot) technologies, applications and challenges," *2016 IEEE Smart Energy Grid Engineering (SEGE)*, pp. 381–385, 2016.

[39] A. Khanna and S. Kaur, "Internet of things (iot), applications and challenges: a comprehensive review," *Wireless Personal Communications*, vol. 114, pp. 1687–1762, 2020.

[40] K. Marinova-Kostova and I. Kostov, "Application of internet of things in industry 4.0," *Economics Ecology Socium*, vol. 5, no. 2, pp. 49–58, 2021.

[41] M. R. Faheem, T. Anees, M. Hussain, A. Ditta, H. Alquhayz, and M. A. Khan, "Indexing in wot to locate indoor things," *IEEE Access*, vol. 11, pp. 53497–53517, 2023.

[42] N. Martinez-Martin, Z. Luo, A. Kaushal, E. Adeli, A. Haque, S. S. Kelly, S. Wieten, M. K. Cho, D. Magnus, L. Fei-Fei, *et al.*, "Ethical issues in using ambient intelligence in health-care settings," *The lancet digital health*, vol. 3, no. 2, pp. e115–e123, 2021.

[43] H. Guo, H. Li, Q. Ren, and W. Wang, "Concept drift type identification based on multi-sliding windows," *Information Sciences*, vol. 585, pp. 1–23, 2022.

[44] A. L. Suárez-Cetrulo, D. Quintana, and A. Cervantes, "A survey on machine learning for recurring concept drifting data streams," *Expert Systems with Applications*, vol. 213, p. 118934, 2023.

[45] W. Roth, G. Schindler, B. Klein, R. Peharz, S. Tschiatschek, H. Fröning, F. Pernkopf, and Z. Ghahramani, "Resource-efficient neural networks for embedded systems," *Journal of Machine Learning Research*, vol. 25, no. 50, pp. 1–51, 2024.

[46] V. Bol'on-Canedo, L. Mor'an-Fern'andez, B. Cancela, and A. Alonso-Betanzos, "A review of green artificial intelligence: Towards a more sustainable future," *Neurocomputing*, p. 128096, 2024.

[47] R. Subramanya, S. Sierla, and V. Vyatkin, "From devops to mlops: Overview and application to electricity market forecasting," *Applied Sciences*, vol. 12, no. 19, p. 9851, 2022.

[48] A. Bastos, M. L. S. C. De Andrade, R. T. Yoshino, and M. M. D. Santos, "Industry 4.0 readiness assessment method based on rami 4.0 standards," *IEEE Access*, vol. 9, pp. 119778–119799, 2021.

[49] Z. Lv, "Digital twins in industry 5.0," *Research*, vol. 6, p. 0071, 2023.

[50] P. Stavropoulos and D. Mourtzis, "Digital twins in industry 4.0," in *Design and operation of production networks for mass personalization in the era of cloud technology*, pp. 277–316, Elsevier, 2022.

[51] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (mlops): Overview, definition, and architecture," *IEEE access*, vol. 11, pp. 31866–31879, 2023.

[52] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. F. Garcia, J. T. Oliva, and A. C. P. L. F. de Carvalho, "Mfe: Towards reproducible meta-feature extraction," *Journal of Machine Learning Research*, vol. 21, no. 111, pp. 1–5, 2020.

[53] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, and A. C. de Carvalho, "Meta-features for meta-learning," *Knowledge-Based Systems*, vol. 240, p. 108101, 2022.

[54] A. Manta-Costa, S. O. Araújo, R. S. Peres, and J. Barata, "Machine learning applications in manufacturing-challenges, trends, and future directions," *IEEE Open Journal of the Industrial Electronics Society*, 2024.