

Deliverable 2.5
Process strategy optimization framework description

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.



Action number: #101138859

Action Acronym: DILAPRO
Action Name: Digital Laser Production

Deliverable Due date: 30th June 2025
Deliverable hand-in date: 31st July 2025

Revision	Date	Author/Organization	Description
1st	13/01/2025	Juliana Fonseca Paim/PEPITe	Editing
2e	22/04/2025	Valentin Kaulmann/PEPITe	Editing
3e	30/04/2025	Marc Monié /PEPITe	Editing
4e	15/05/2025	Valentin Kaulmann/PEPITe	Editing
5e	13/06/2025	Valentin Kaulmann/PEPITe	Editing
6e	17/07/2025	Sankhya Mohanty/DTU	Editing
7e	24/07/2025	Valentin Kaulmann	Editing
8e	30/07/2025	Valentin Kaulmann	Editing

Acronyms and abbreviations

AM: Additive Manufacturing

CSV: Comma-separated values file format

DED: Directed Energy Deposition

DT: Digital Twin

ICME: Integrated Computational Materials Engineering methods

IoT: Internet of Things

LPBF: Laser Power Bed Fusion

ML: Machine Learning

MODBUS: client/server data communication Protocol

MQTT: Message Queuing Telemetry Transport standard messaging protocol for IoT

OPC-UA: Open Platform Communication - Unified Architecture for data exchange open source

RDBMS: Relational Database Management System

SQL: Structured Query Language standard language store, manipulate and retrieve data

OS: Operating System

Table of contents

1. Introduction 7

2. Methodology: Incremental development approach 8

2.1. Key considerations for effort estimation 9

2.2. Benefits of the incremental strategy 9

2.3. Summary of methodological choices 10

3. Platform architecture overview 10

3.1. High-level system structure 10

3.2. Unified data storage and management 10

3.3. Model execution engine 10

3.4. Visualization and user interaction 11

3.5. Workflow and data pipeline design 11

3.6. Core technologies and tools 13

4. End-to-end workflow architecture 16

4.1. Step 1: File Upload & Metadata capture 16

4.2. Step 2: Use Case Configuration 16

4.3. Step 3: Model Container Launch 18

4.4. Step 4: Model Execution 18

4.5. Step 5: Result Storage 18

4.6. Step 6: Result Visualization 18

4.7. Step 7: Feedback & Iteration 19

5. Scenario specific workflows and deployment architecture 19

5.1. Scenario 1: Non-Time-Series CSV processing 20

5.1.1. File Upload and Metadata Management 20

5.1.2. Use Case Configuration 21

5.1.3. Model Launch in Docker 21

5.1.4. Model Execution in the Docker Container 21

5.1.5. Error Handling 21
5.1.6. Output Management 22
5.1.7. Result Visualization 23
5.2. Scenario 2: Time-Series CSV processing 24
5.2.1. File Upload and Metadata Management 24
5.2.2. Use Case Configuration 25
5.2.3. Model Launch in Docker 25
5.2.4. Model Execution in the Docker Container 25
5.2.5. Error Handling 26
5.2.6. Output Management 26
5.2.7. Result Visualization 26
5.3. Scenario 3: Image Batch classification 27
5.3.1. File Upload and Metadata Management 27
5.3.2. Use Case Configuration 28
5.3.3. Model Launch in Docker 28
5.3.4. Model Execution in the Docker Container 28
5.3.5. Error Handling 29
5.3.6. Output Management 29
5.3.7. Result Visualization 29
6. Architecture benefits 30
6.1. Modularity and Extensibility 30
6.2. Scalability and Performance 30
6.3. Security, Compliance, and Data Protection 30
6.4. Reproducibility and Quality Assurance 31
6.5. Usability and Stakeholder Engagement 31
6.6. Advanced Analytics and Decision Support 31
7. Hybrid modelling strategy: combining Multiphysics and ML models 32
8. Conclusion 37
A1. Test environment images of the Dilafact platform 38

Table of figures

Table 1 : Core Tools and Responsibilities..... 16
Table 2 : Scenario 1 : Non-Time- Series CSV processing..... 20
Table 3 : Scenario 2: Time-Series CSV processing..... 24
Table 4 : Scenario 3: Image Batch classification 27
Table 5 : Deformation models for the optimization framework showing the original geometry, the deformed shape and the distortion compensated geometry – the above figure is taken from an example simulation case available with the ANSYS Additive software..... 33
Table 6 : Comparison between Classification and Segmentation models in terms of data format, annotation, architecture, hardware, and inference speed 34
Table 7 : Surface Generation model specifications 35
Table 8 : Grayscale height maps showcasing the surface topographies emerging from PBF of 316L. The figure on the extreme left is produced by the generative AI tool. 35
Table 9 : Power vs scan speed plot showing the expected zone with low likelihood of defect (blue) and the zones where balling or excessive humping (red) are expected. The triangles and plus symbols correspond to data used to extract the model, while the circles..... 36
Table 10 : Homepage of the platform 38
Table 11 : Page for selecting inputs and running the model..... 39
Table 12 : View of Apache Airflow for model orchestration 39
Table 13 : Example of results for density prediction..... 40

Executive Summary

Deliverable 2.5 of the DILAPRO project presents a comprehensive description of the process strategy optimization framework. The document highlights the framework and workflows for integrating Multiphysics simulation models and data-driven machine learning approaches within a common platform. The central objective of this deliverable is to outline the detailed architecture, methodology, and implementation scenarios of the DILAFACT platform, emphasizing modularity, scalability, and practical usability.

The DILAFACT platform employs an incremental development strategy, systematically addressing three primary scenarios: structured CSV data processing, time-series data management, and image batch classification. Each scenario progressively introduces greater complexity, building upon a unified and reusable architectural foundation composed of robust data ingestion, storage systems, Docker-based model execution, and advanced visualization components.

Key technical highlights include:

- **Robust Data Infrastructure:** Utilizing DATAmaestro and PostgreSQL for structured and unstructured data storage, ensuring seamless integration, real-time analytics, and historical data management.
- **Orchestration and Automation:** Leveraging Apache Airflow and Docker to provide reproducibility, scalability, and reliable workflow automation.
- **Advanced Visualization:** Offering customized and standardized dashboards for intuitive user interaction and effective decision support.

The deliverable further emphasizes the hybrid modelling strategy adopted within DILAPRO. Four implementation examples are showcased where advanced Multiphysics simulations, machine learning algorithms, statistical methods and/or a combination of these are deployed to predict critical manufacturing quality aspects. In each case, a short description of the model characteristics is supplemented by a brief text regarding the deployment scenario (i.e. which workflow to utilize and how to use the model for process optimization).

In conclusion, this document not only defines the current capabilities and architecture of the DILAFACT platform but also lays out a clear pathway for future improvements, scalability opportunities, and broader industrial applicability. The strategies and tools detailed herein provide stakeholders with a framework to significantly improve efficiency, reliability, and innovation in manufacturing processes.

1. Introduction

The DILAPRO project aims to advance digital twins and process optimization for laser-based manufacturing processes, including but not limited to Additive Manufacturing (AM) techniques such as Direct Energy Deposition (DED), Laser Powder Bed Fusion (LPBF), and Laser Texturing. At the heart of this effort lies the DILAFACT platform, a unified digital ecosystem that integrates Multiphysics simulation, data-driven analytics, and decision-support tools.

DILAFACT provides a robust infrastructure composed of interconnected databases, a data lake, and a data warehouse. It is capable of managing both structured and unstructured data across diverse industrial contexts. Its capabilities span data collection, transformation, storage, cleaning, processing, and visualization, all supported by scalable cloud technologies.

The platform is designed to help users select and optimize laser-based manufacturing strategies based on component properties, industry standards, and real-time operational data. It supports decision-making not only for selecting the most suitable technology (DED, LPBF, or laser texturing) but also for optimizing post-processing, ensuring quality, and achieving certification compliance.

A key innovation of DILAFACT is its deployment of digital twins for each supported process. These digital twins enable real-time monitoring, predictive quality control, and adaptive parameter optimization. By combining high-fidelity Multiphysics models with advanced machine learning algorithms, the system can predict a wide range of properties such as density, hardness, elongation, surface quality, and fatigue strength.

The development of the platform follows an incremental methodology where each new functionality is delivered iteratively and validated through close collaboration with stakeholders. At the foundation lies DATAmaestro (DM), our proprietary data platform, which is integrated with open-source tools: Apache Airflow for workflow orchestration, Docker for container execution, PostgreSQL for storage, Rust for high-performance back-end services, and Leptos for a modern front-end interface. All these components are deployed on a secure Ubuntu environment.

The integrated decision-support framework guides users through complex manufacturing decisions, supported by a combination of Multiphysics models, reduced order or statistical response models and machine learning models trained on both simulated and real industrial data. Collaboration with project partners ensures the contribution of domain knowledge and experimental data essential for model validation and continuous improvement.

In summary, DILAFACT delivers a modular, scalable, and intelligent decision-support platform for laser-based manufacturing. It bridges the gap between physics-based simulations and data-

driven insights, supporting both process innovation and industrial adoption. The following sections present the methodology, system architecture, and deployment strategies underpinning this vision, building on the foundation set by earlier deliverables (D2.1 to D2.4) and paving the way for upcoming hybrid modelling and deployment phases.

The deliverable was delayed 1 month due to the unforeseen personnel changes at DTU in the month before deadline, who were required to provide significant and crucial input for this deliverable.

2. Methodology: Incremental development approach

To manage complexity and ensure continuous value delivery, the DILAFACT platform follows an incremental development methodology. This approach structures the evolution of the platform into a sequence of well-defined and validated steps, where each iteration introduces new functionalities while reinforcing the existing technical foundation.

The development strategy began with three initial deployment scenarios; each chosen to exercise core architectural components and extend system capabilities progressively.

The first scenario focused on building a robust foundation by implementing essential infrastructure components. This included tools for data ingestion and management, model execution environments, and visualization interfaces. Since these elements form the basis for all future workflows, significant effort was invested in setting up reliable data storage layers, container orchestration logic, and user interaction mechanisms.

Once this foundation was in place, functionality was gradually expanded through targeted application scenarios. The first deployment enabled the processing of structured tabular data (e.g., CSV files) to produce numerical outputs, such as predictions or classifications. This established critical features such as file upload mechanisms, containerized model execution, and result visualization **(*more details in section 5.1*)**.

The second deployment extended the system to handle time-series data, such as sensor streams or machine logs. This scenario added complexity by integrating external data sources while reusing many components developed in the first scenario, such as data validation pipelines and result dashboards **(*more details in section 5.2*)**.

The third deployment addressed the processing of image data. It introduced domain-specific logic for image classification and interpretation, while still relying on shared components like file handling, model containers, and storage routines **(*more details in section 5.3*)**.

A key principle of this methodology is reusability. Components such as the API layer, web interface, and orchestration tools are designed to be generic and modular, allowing them to support multiple types of data and model logic with minimal duplication. For example, the file upload system developed during the first scenario was adapted to handle images in the third, requiring only minor modifications. This approach significantly reduces development effort and accelerates delivery timelines for new capabilities.

By continuously integrating feedback and iterating on functional blocks, the DILAFACT platform remains scalable, adaptable, and aligned with both technical goals and user needs.

2.1. Key considerations for effort estimation

Effort estimation within the DILAFACT development process is driven by the complexity of each deployment scenario, as well as the degree of reusability and testing required.

The first scenario, focused on processing structured tabular data (CSV files), represents the least complex implementation. It involves direct file uploads, straightforward numerical outputs, and limited integration with external systems. The second scenario introduces greater complexity by integrating time-series data from external sources, requiring connections to partner systems and more advanced data synchronization and handling logic. The third scenario is the most demanding, involving image processing workflows and classification logic, which necessitate more intensive computing, storage management, and tailored visualization.

Another key factor in effort estimation is the platform's emphasis on shared components and reusability. Core functionalities such as the API layer, database configuration, orchestration pipeline, and web interface were designed from the outset to be modular and extensible. Once developed, these components are reused across scenarios with minimal adaptation, enabling a significant reduction in development time and resource allocation for subsequent features.

Each scenario is subject to comprehensive testing procedures before deployment. Particular attention is given to error handling, including validation of malformed files, management of interrupted data flows, and resilience to user disconnection. This rigorous approach ensures the platform remains robust and reliable as its scope and complexity grow.

2.2. Benefits of the incremental strategy

The incremental development strategy provides several strategic advantages for both the technical team and end users. First, it enables continuous improvement through frequent, manageable releases. These allow feedback from stakeholders and users to be quickly integrated into the development cycle, ensuring that the platform evolves in response to real-world needs and operational constraints. Second, the approach promotes scalability and adaptability. By validating each new functionality in isolation and layering features iteratively, the system

architecture remains stable, maintainable, and forward-compatible. This structure also facilitates the incorporation of emerging technologies, future deployment scenarios, and new partner contributions with minimal disruption.

2.3. Summary of methodological choices

DILAFACT's adoption of an incremental development methodology ensures that the platform evolves in a controlled, efficient, and scalable manner. Each new feature or scenario is introduced as part of a structured workflow, fully integrated into the existing ecosystem and validated through rigorous testing and stakeholder engagement. This strategy underpins the platform's technical architecture and ensures readiness for deployment in real-world industrial environments.

3. Platform architecture overview

The DILAFACT platform is designed to provide a modular, scalable, and unified environment for data management, advanced analytics, and additive manufacturing process optimization. The design integrates industrial data sources, robust storage solutions, a flexible model execution engine, and modern visualization tools, all orchestrated through a common, incremental development methodology.

3.1. High-level system structure

At the core of the DILAFACT architecture is a **central Linux server hosted on Hetzner Cloud**, responsible for orchestrating data collection, storage, model execution, and visualization. Data from various **partner data sources**, including files, OPC-UA streams, CSVs, SQL databases, MQTT, and MODBUS protocols, is ingested either automatically (using DATAmaestro Collectors) or manually by users via the web interface.

A dedicated **API layer** serves as the bridge between data ingestion, orchestration, and visualization components. Automated data flows are organized by an **Apache Airflow** service, which ensures data consistency and structured input organization for downstream processing.

3.2. Unified data storage and management

All ingested data and metadata are centrally stored in a robust **data warehouse** backed by a PostgreSQL database for metadata and a file system for storing unstructured data (e.g., images, waveforms). A dedicated **DATAmaestro Lake** component enables efficient storage and retrieval of large-scale time-series datasets.

3.3. Model execution engine

The DILAFACT platform supports a **model engine** that orchestrates the execution of scenarios (such as regression/classification on tabular data, time-series analysis, and image batch

classification). Each scenario may have multiple model versions, managed and executed as isolated **Docker containers** for maximum reproducibility and scalability.

- The orchestrator (Apache Airflow) coordinates pipeline execution, launching containers with all required environment variables and data paths.
- Containers execute the selected model, perform computation (e.g., prediction, forecasting, classification), and write results back to the data warehouse, with job metadata and status tracked through

3.4. Visualization and user interaction

Visualization is a core architectural component, enabling both real-time monitoring and post-process analysis. The platform offers multiple visualization tools adapted to different user needs:

- **Apache Airflow**: Provides native views for monitoring workflow execution, task status, and job scheduling.
- **DATAmaestro Visualization**: Delivers advanced dashboards for detailed analytics and operational insights across manufacturing processes.
- **Custom Python Visualizations**: Tailored Python-based dashboards and scripts support specialized plots, metrics tracking, and domain-specific visual outputs.

Users interact with the platform through:

- Manual data entry interfaces,
- Visualization dashboards,
- Job configuration and orchestration tools.

3.5. Workflow and data pipeline design

The workflow follows a structured pipeline that ensures consistency, modularity, and easy integration into industrial environments:

1. **File Upload & Metadata Capture**: Files can be **manually uploaded** by the user or **automatically collected** via a collector (e.g., DATAmaestro Collector). Metadata is extracted or entered during ingestion.

2. **Use Case Configuration**: The user selects a model and defines processing parameters. Input files are automatically assigned based on the model's requirements.

3. **Launching and Managing Docker Containers with Apache Airflow**:

- **Launch the Docker container**: The **Apache Airflow** orchestrator starts the Docker container corresponding to the selected model.
- **Task Management**: Through **Airflow**, the user can monitor task execution, schedule recurring executions, restart failed tasks, or cancel an ongoing execution.
- **Monitoring and Control**: The user can interact with the Airflow interface to view execution logs, check errors, and manage dependencies between different workflow stages (e.g., data validation, model execution, or result transfer).

4. **Model Execution**: The model runs in its isolated environment, processes the data, and generates the results.

5. **Result Storage**: Outputs are stored either as:
   - **Raw files** (e.g., images, CSVs, logs),
   - or in an **SQL database**, depending on the expected format.

6. **Result Visualization**: Results can be viewed via:
   - **DATAmaestro dashboards** for standardized visualization,
   - or **custom Python visualizations**, provided by the consortium partners.

3.6. Core technologies and tools

To build a scalable and flexible data-science platform, DILAFACT relies on a judicious mix of open-source and commercial technologies. Open-source tools offer several systemic advantages: they are free to download and use, avoiding licence fees and are highly customisable since the source code can be modified to fit specific needs. Projects are backed by large, active communities that provide support, frequent updates and transparent code audits, and the use of open standards makes these tools interoperable and reduces the risk of vendor lock-in. Critically, open-source software gives users full control over their technology stack, allowing them to switch vendors or customise features without being tied to a proprietary ecosystem. On the other hand, commercial software remains important in areas where stability, reliable customer support and a polished, user-friendly product are paramount; proprietary solutions are generally highly reliable, offer accessible customer service and provide feature-rich, purpose-built environments. By combining open-source frameworks for orchestration, containerisation, storage and development with commercial platforms like DATAmaestro for industrial data management, DILAFACT benefits from the flexibility and cost-efficiency of open source while retaining the dedicated support and specialised functionality of commercial offerings.

Open-Source Technologies

- **Apache Airflow**: Workflow orchestration and job scheduling.
  Apache Airflow is an open-source platform for orchestrating complex workflows and managing data pipelines. It allows users to define, schedule, and monitor workflows using Directed Acyclic Graphs (DAGs). Key features include task scheduling, dynamic workflow management, task monitoring, and scalability to handle complex workflows.
    - **Why open source vs. commercial schedulers?** No license fees, a very large ecosystem of providers/operators, and full code transparency avoid vendor lock-in and make it easy to embed Airflow into continuous integration and continuous deployment workflows and heterogeneous, multi-cloud stacks.
- **Docker**: Containerized model execution for reproducibility.
  Docker provides isolated environments for executing models in containers, ensuring reproducibility across different stages of model development and deployment. This also simplifies the integration of external dependencies and enhances portability.
    - **Why open source vs. proprietary runtimes?** It relies on OCI (Open Container Initiative) standards, is universally supported by cloud/Kubernetes platforms, and avoids dependence on closed, vendor-specific packaging/execution formats,

drastically reducing switching costs.

- **PostgreSQL**: Centralized metadata and results storage. PostgreSQL is an open-source relational database management system (RDBMS) used to store metadata and model results. Its robust feature set and reliability make it ideal for managing large datasets and complex queries.
    - **Why open source vs. commercial RDBMS?** Enterprise-grade features (JSONB, window functions, extensions like PostGIS/TimescaleDB) without per-core licensing, plus portability and extensibility that match, or exceed, many paid alternatives.

- **Rust/Leptos**: High-performance backend and interactive front-end framework. Rust is a systems programming language known for its speed and safety, while Leptos is a framework built on Rust to provide a reactive, high-performance front-end for web applications. This combination ensures both fast backend processing and seamless user interaction.
    - **Why open source vs. proprietary frameworks/stacks?** Memory safety without a garbage collector, zero-cost abstractions, and no runtime licensing deliver performance and security comparable to high-end commercial stacks, with full auditability of the code.

- **Ubuntu LTS**: Stable, secure operating environment. Ubuntu LTS (Long Term Support) provides a secure and stable Linux operating system environment, offering ongoing security patches and updates, making it a reliable foundation for DILAFACT's infrastructure.

    - **Why open source vs. commercial Operating Systems (OS)?** Predictable 5-year support windows, no per-node subscription by default, and broad community & enterprise adoption lower total cost of ownership while ensuring compatibility with the entire open-source data/ML toolchain.

Commercial Technologies

- **DATAmaestro**: A data management solution that includes a data warehouse and a data lake for the storage and analysis of industrial data. It also enables result visualization and analysis of large datasets.

- **DATAmaestro Collectors:** Components responsible for collecting data from various industrial sources. These collectors can retrieve data via protocols like OPC-UA or specific interfaces (HTTP/S, MQTT, Modbus TCP) and send it to **DATAmaestro** for storage and analysis. They operate in two modes: historical (for retrieving archived data) and subscription (for real-time data). A **Bridge** can be used to convert non-OPC-UA protocols into OPC-UA.

Summary Table: Core Tools and Responsibilities

Layer	Tool/Platform	Role(s)
Orchestration	Apache Airflow	Workflow automation, scheduling, monitoring
Containerization	Docker	Model/package isolation, reproducibility, scaling
Data Storage	PostgreSQL	Structured metadata/results
OS/Foundation	Ubuntu LTS	Stable, secure operating system
Backend	Rust	High-performance APIs, job management
Frontend	Leptos	Interactive dashboards, user interface
Core	DATAmaestro	Central coordination hub with integrated components (DM Collector, DM Lake, DM Analytics, DM Dashboarding) for time series data flow orchestration, ML model management, and system integration

API & Integration	REST, OpenAPI	Programmatic access, system integration

Table 1 : Core Tools and Responsibilities

Benefits of the Chosen Stack

- **Modularity:** Each component can evolve or be replaced independently.
- **Reproducibility & Traceability:** Containerization, versioned workflows, and rich metadata guarantee robust, auditable pipelines.
- **Scalability:** From lab to production, the platform is cloud-native and horizontally scalable.
- **Security & Compliance:** Open-source tools and commercial modules are configured to meet industrial security standards (RBAC, encryption, audit trails).
- **User Empowerment:** Advanced analytics, interactive dashboards, and programmatic APIs make the platform accessible to engineers, data scientists, and business users.

4. End-to-end workflow architecture

The DILAFACT platform's seven-step pipeline is a robust, modular workflow designed to automate and streamline the journey from data acquisition to actionable results for industrial additive manufacturing use cases. Each stage is designed for traceability, scalability, and extensibility, ensuring the system can evolve with future requirements and new use cases.

4.1. Step 1: File Upload & Metadata capture

Automatic and Manual Data Acquisition

- **Manual**: Users upload one or more files (CSVs, images, log files) through the web interface LeptOS.
- **Automatic**: DATAmaestro Collectors retrieve data continuously or in historical batches via OPC-UA, MQTT, MODBUS, SQL, etc.
- **Validation**: Incoming files are checked for format, size, structure, encoding, and schema compliance.
- **Metadata Extraction**: Filename, source, upload timestamp, file size, data schema, sampling rate, units, and quality indicators are extracted and stored in PostgreSQL.

4.2. Step 2: Use Case Configuration

Use Case Configuration via Apache Airflow

- Airflow DAGs fully define the use case (e.g. tabular regression, time-series forecasting, image classification).
- Each DAG specifies:
    - The **model version** (the Docker image to run),
    - The **execution parameters** (input columns, forecast horizon, anomaly thresholds, hyperparameters, output formats and naming).
- Variables for each run are entered or modified in the Airflow UI or via its API, and all settings are recorded in PostgreSQL for auditability and reproducibility.

4.3. Step 3: Model Container Launch

Orchestration and Docker Container Launch

- Airflow resolves DAG dependencies and triggers Docker to:
    - Mount input/output volumes,
    - Inject secure environment variables (credentials, paths, execution IDs),
    - Initialize the model container.
- From the Airflow UI, operators can monitor task status, retry failed containers, cancel or schedule runs, view logs, and inspect task graphs.

4.4. Step 4: Model Execution

Model Execution

- The container loads and parses input data according to the configured schema.
- Preprocessing (normalization, feature extraction), core computation (regression, classification, forecasting, anomaly detection, or image processing), and post-processing (aggregation, filtering, formatting) are executed.
- All actions and exceptions are logged exhaustively for diagnostics.

4.5. Step 5: Result Storage

Result Persistence

- **PostgreSQL**: Raw output values (numeric or string) and labels/annotations are written into tables, linked to the job ID.
- **DATAmaestro Lake:** Time-series results are ingested for historical analysis and SQL querying.
- **Filesystem**: Large or binary artifacts (images, raw logs, segmentation masks) are stored with metadata pointers for retrieval.

4.6. Step 6: Result Visualization

Result Visualization and Access

- **REST API**: Provides programmatic access to job statuses, output data, error reports, and execution metadata.
- **DATAmaestro Dashboards**: Standardized views (charts, tables, download links).
- **Custom Python Visualizations**: Consortium-provided scripts and reports for specialized analyses.
- Real-time monitoring and historical exploration via the web interface.

4.7. Step 7: Feedback & Iteration

Continuous Learning and System Improvement

- **User Feedback:** Users can comment on results, flag issues, and suggest enhancements directly in the UI, feeding into the product backlog.
- **Automated Monitoring:** System health and performance metrics are collected and analysed, guiding operational improvements and scaling decisions.

This step-by-step pipeline ensures a consistent, transparent, and flexible workflow combining automated orchestration with user control to support industrial-scale data processing and model-driven insights.

5. Scenario specific workflows and deployment architecture

DILAFACT's seven-step pipeline offers a consistent framework for turning raw inputs into actionable insights, but its implementation varies by use case. Different data modalities, text, tabular data, time-series readings, images, or multimodal streams, require distinct ingestion mechanisms, preprocessing routines and model architectures. An AI pipeline therefore comprises multiple stages, data collection, validation, cleaning, storage, model execution, post-processing, logging and visualisation and the design of each stage must reflect the demands of the specific problem.

In the sections that follow, we will walk through three representative scenarios that illustrate how DILAFACT adapts this general pipeline to different domains. For non-time-series CSV processing, we will see how a static tabular dataset uploaded by a user flow through file validation, Docker-based model execution, and storage of a numerical prediction. In time-series CSV processing, we will examine how automated collectors ingest continuous streams from industrial protocols (OPC-UA, MQTT, Modbus, etc.), pass JSON-formatted readings into a containerised model, and visualize normalized outputs in the DATAmaestro dashboard. Finally, in image batch classification, we will explore how uploaded images are validated and processed by a pre-trained computer-vision model to produce categorical labels. For each use case we will highlight the key decisions around data handling, model selection, and result presentation, showing how the seven-step backbone is tailored to suit the unique characteristics of each data type and business objective.

5.1. Scenario 1: Non-Time-Series CSV processing

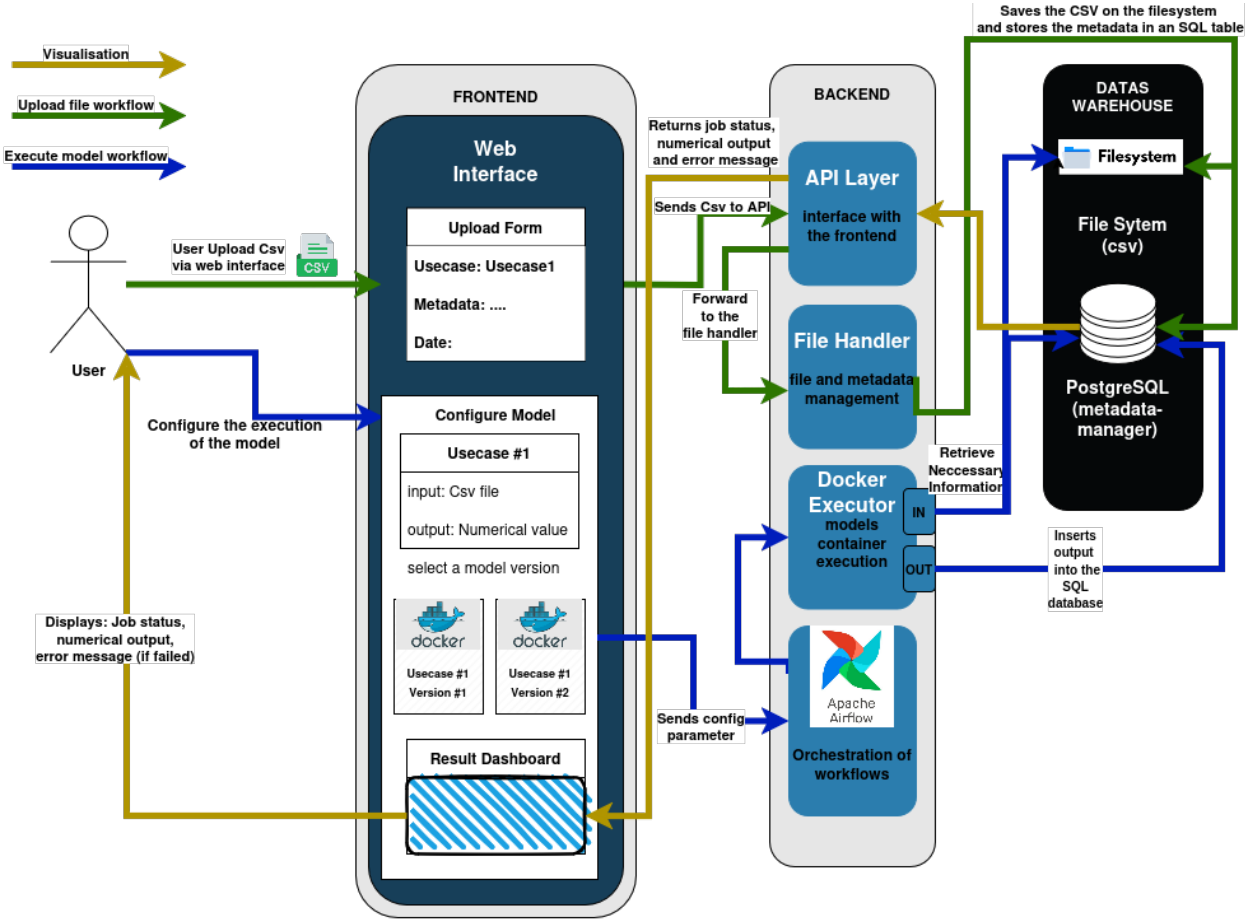


Table 2 : Scenario 1: Non-Time- Series CSV processing

5.1.1. File Upload and Metadata Management

User Action:

- The user uploads a CSV file via a web interface.

API Layer:

- Receives the CSV file along with its metadata (file name, size, etc.) and forwards them to the File Handler.

File Handler:

- Validates the CSV file (checks the number of columns, column titles, values, etc.).
- Stores the metadata in an SQL database (e.g., columns like file name, file size, upload date).
- Saves the CSV file to the filesystem at an accessible location (e.g., /data/usecase1/20250301-abcde.csv).

5.1.2. Use Case Configuration

User Action:

- Via the web interface, the user configures the execution by selecting the file name, the output name, and the model version to use (e.g., "Version 1" or "Version 2"), each corresponding to a Docker image stored on the server.

5.1.3. API Layer

- Sends the configuration parameters to Airflow. A DAG is triggered for the job execution.

5.1.3. Model Launch in Docker

Job Manager:

- Requests the Docker Executor to launch a Docker container specific to the use case and the selected model version (e.g., Docker image tagged as usecase1: version1).

Docker Executor:

- Launches the Docker container and provides it with the necessary information:
    - Path to the CSV file on the filesystem (e.g., /data/usecase1/20250301-abcde.csv).
    - Environment variables for SQL database connection (e.g., DB_HOST, DB_USER, DB_PASSWORD).

5.1.4. Model Execution in the Docker Container

Docker Container:

- Retrieves the CSV file from the filesystem using the provided path.
- Runs the pre-trained model on the CSV data.
- Generates a numerical output (e.g., a prediction or score).
- Inserts the output directly into the SQL database (e.g., into a model_outputs table with columns like execution_id, output_value, timestamp), using the connection details provided via environment variables.

5.1.5. Error Handling

Errors in the Container:

- If an error occurs (e.g., malformed CSV file, model failure), the container updates the job status in the jobs table (e.g., status = "failed") and logs an error message in a column like error message.

SQL Connection Errors:

- The container can implement retry logic for temporary connection failures or log the error in SQL if it persists.

User Disconnection:

- If the user disconnects during processing, it does not affect the execution since it is asynchronous. The user can later check the job status using the execution_id.

5.1.6. Output Management

The numerical output generated by the model is stored in the SQL database, associated with the execution_id (e.g., INSERT INTO model_outputs (execution_id, output_value) VALUES ('12345-abcde', 0.95)).

Job Manager:

- Updates the job status in the jobs table once the processing is complete (e.g., status = "completed").

5.1.7. Result Visualization

Web Interface:

- The user can query the job status and results via the API Layer by providing the execution_id.

API Layer:

- Retrieves data from the jobs table (for status) and model outputs table (for results).

Display:

- Results are displayed on a dashboard or dedicated page in the web interface, showing:
    - Job status ("in progress", "completed", "failed").
    - Numerical output (e.g., "Prediction: 0.95").
    - Any error message if the job failed.

5.2. Scenario 2: Time-Series CSV processing

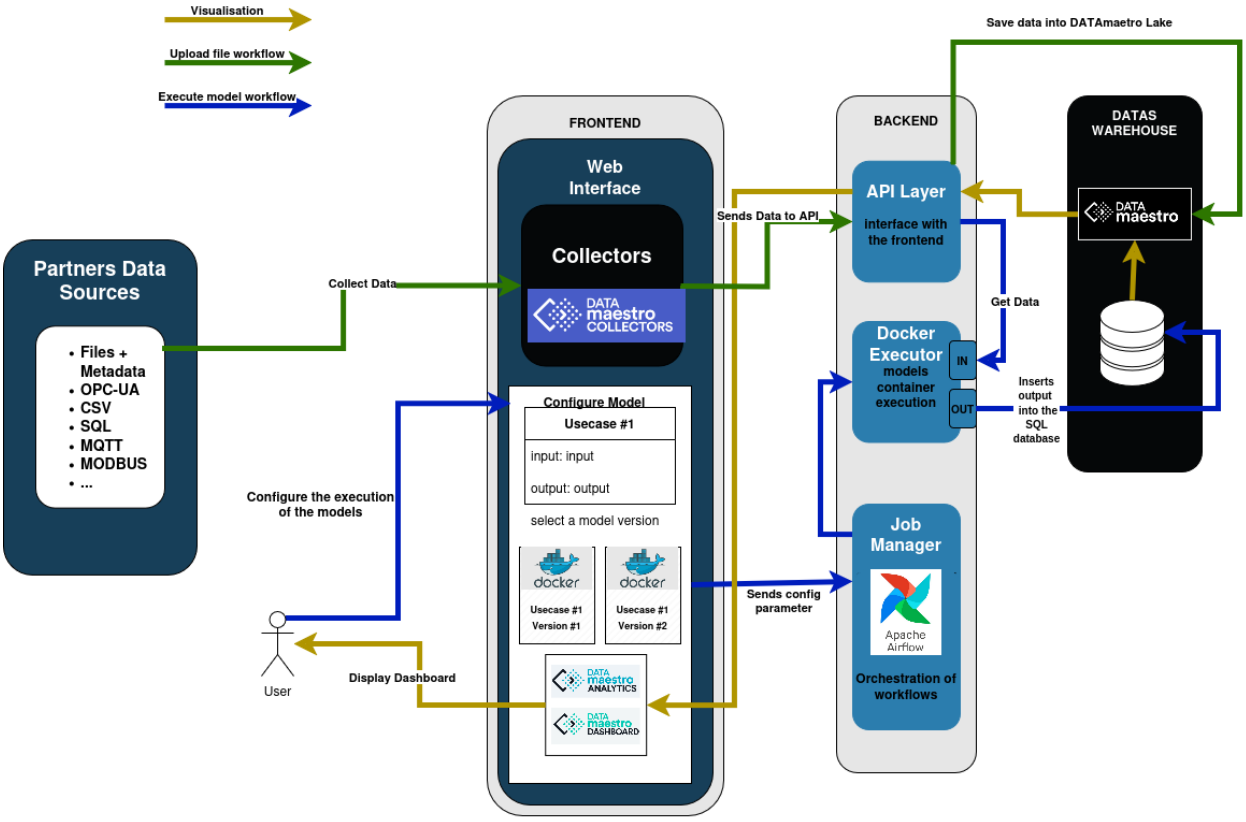


Table 3 : Scenario 2: Time-Series CSV processing

5.2.1. File Upload and Metadata Management

Collectors (DATAmaestro Collectors):

- Connects to external **Partner Data Sources** (e.g., OPC-UA, MQTT, SQL, MODBUS) to automatically collect time series data (e.g., sensor readings with timestamps).
- Receives the time series data along with its metadata (e.g., source identifier, timestamp range, data type) and forwards them to the **Data maestro Lake** vis **API.**

5.2.2. Use Case Configuration

User Action:

- Via the web interface, the user configures the execution by selecting the file name, the output name, and the model version to use (e.g., "Version 1" or "Version 2"), each corresponding to a Docker image stored on the server.

API Layer:

- Sends the configuration parameters to the Job Manager.

5.2.3. Model Launch in Docker

Job Manager:

- Generates a unique execution_id (e.g., 12345-abcde) to track the job execution.
- Requests the Docker Executor to launch a Docker container specific to the use case and the selected model version (e.g., Docker image tagged as usecase1: version1).

Docker Executor:

- Launches the Docker container and provides it with the necessary information:
    - Data in JSON format supplied through the Data Maestro API.
    - Environment variables for SQL database connection (e.g., DB_HOST, DB_USER, DB_PASSWORD).
    - The execution_id (e.g., EXECUTION_ID=12345-abcde).

5.2.4. Model Execution in the Docker Container

Docker Container:

- Fetches the input data in JSON format from the DATAmaestro via an API.
- Executes the pre-trained model on the input data.
- Produces a normalized output.
- Stores the output directly in the SQL database (e.g., in a model_outputs table with columns such as execution_id, output_category, and timestamp), using the connection details provided through environment variables.

5.2.5. Error Handling

Errors in the Container:

- If an error occurs
- The container updates the job status in the jobs table (e.g., status = "failed") and logs an error message in a column like error_message.

SQL Connection Errors:

- The container can implement retry logic for temporary connection failures or log the error in SQL if it persists.

User Disconnection:

- If the user disconnects during processing, it does not affect the execution since it is asynchronous. The user can later check the job status using the execution_id.

5.2.6. Output Management

SQL Database:

- The data output generated by the model is stored in the SQL database, associated with the execution_id.

Job Manager:

- Updates the job status in the jobs table once the processing is complete (e.g., status = "completed").

5.2.7. Result Visualization

Display:

- DATAmaestro gathers data from an SQL database to populate the data lake with the model outputs.
- Results are presented on the DATAmaestro Dashboard.
- Results can be analysed using DATAmaestro Analytics.

5.3. Scenario 3: Image Batch classification

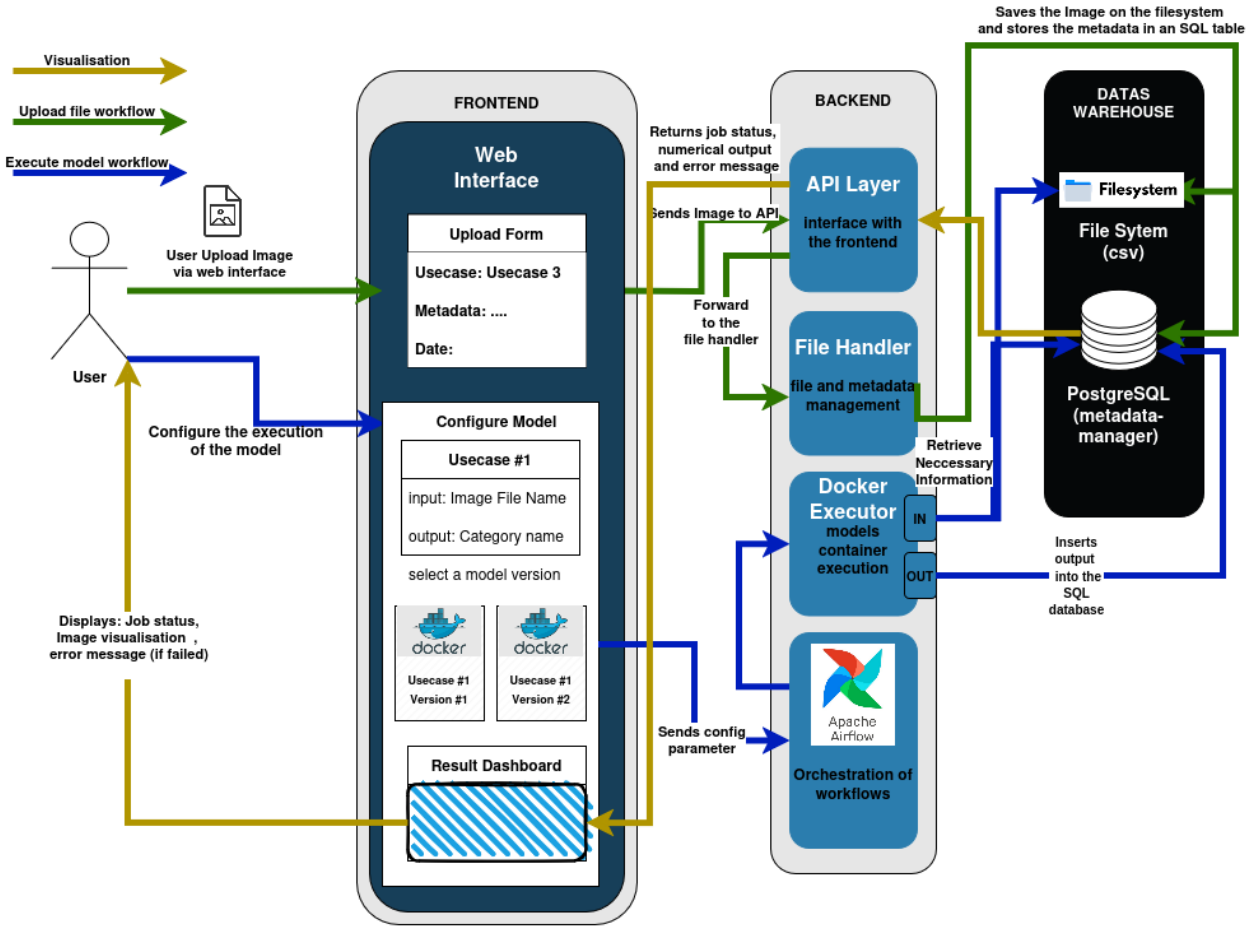


Table 4 : Scenario 3: Image Batch classification

5.3.1. File Upload and Metadata Management

User Action:

- The user uploads an image file via a web interface.

API Layer:

- Receives the image file along with its metadata (file name, size, etc.) and forwards them to the File Handler.

File Handler:

- Validates the image file (checks the format, size, etc).
- Stores the metadata in an SQL database (e.g., columns like file_name, file_size, upload_date).
- Saves the image file to the filesystem at an accessible location (e.g., /data/usecase1/20250301-abcde.jpg).

5.3.2. Use Case Configuration

User Action:

- Via the web interface, the user configures the execution by selecting the file name, the output name, and the model version to use (e.g., "Version 1" or "Version 2"), each corresponding to a Docker image stored on the server.

API Layer:

- Sends the configuration parameters to the Job Manager.

5.3.3. Model Launch in Docker

Job Manager:

- Generates a unique execution_id (e.g., 12345-abcde) to track the job execution.
- Requests the Docker Executor to launch a Docker container specific to the use case and the selected model version (e.g., Docker image tagged as usecase1: version1).

Docker Executor:

- Launches the Docker container and provides it with the necessary information:
    - Path to the image file on the filesystem (e.g., /data/usecase1/20250301-abcde.jpg).
    - Environment variables for SQL database connection (e.g., DB_HOST, DB_USER, DB_PASSWORD).
    - The execution_id (e.g., EXECUTION_ID=12345-abcde).

5.3.4. Model Execution in the Docker Container

Docker Container:

- Retrieves the image file from the filesystem using the provided path.
- Runs the pre-trained model on the image data.
- Generates a categorical output (e.g., a prediction or score).

- Inserts the output directly into the SQL database (e.g., into a model_outputs table with columns like execution_id, output_category, timestamp), using the connection details provided via environment variables.

5.3.5. Error Handling

Errors in the Container:

- If an error occurs (e.g., corrupted image file, model failure)
- The container updates the job status in the jobs table (e.g., status = "failed") and logs an error message in a column like error_message.

SQL Connection Errors:

- The container can implement retry logic for temporary connection failures or log the error in SQL if it persists.

User Disconnection:

- If the user disconnects during processing, it does not affect the execution since it is asynchronous. The user can later check the job status using the execution_id.

5.3.6. Output Management

- The categorical output generated by the model is stored in the SQL database, associated with the execution_id (e.g., INSERT INTO model_outputs (execution_id, output_category) VALUES ('12345-abcde', 'category A')).

Job Manager:

- Updates the job status in the jobs table once the processing is complete (e.g., status = "completed").

5.3.7. Result Visualization

Web Interface:

- The user can query the job status and results via the API Layer by providing the execution_id.

API Layer:

- Retrieves data from the jobs table (for status) and model_outputs table (for results).

Display:

- Results are displayed on a dashboard or dedicated page in the web interface, showing:
    - Job status ("in progress", "completed", "failed").
    - Categorical output (e.g., "Category A").
    - Any error message if the job failed

6. Architecture benefits

The DILAFACT architecture has been carefully designed to meet the diverse, demanding needs of modern additive manufacturing environments, enabling robust data-driven decision-making, rapid innovation, and industrial scalability. Below, we summarize the principal benefits provided by the architecture:

6.1. Modularity and Extensibility

- **Plug-and-play components:** The platform's modular design based on containerization and microservices allows new models, data sources, use cases, or analytics modules to be integrated with minimal effort and no impact on existing functionality.

- **Future-proofing:** As technologies and industrial needs evolve, the architecture supports seamless upgrades and extension, avoiding vendor lock-in and maximizing the useful lifetime of the platform.

6.2. Scalability and Performance

- **Cloud-native deployment:** Leveraging cloud infrastructure (Hetzner, Ubuntu LTS) and container orchestration, DILAFACT can scale horizontally to accommodate increasing data volumes, more users, and additional compute-intensive workloads.

- **Efficient resource allocation:** Orchestrated execution (Apache Airflow, Docker) ensures that compute resources are used optimally, and workloads are isolated, leading to predictable performance and efficient scaling from proof-of-concept to industrial deployment.

6.3. Security, Compliance, and Data Protection

- **Industrial security standards:** The platform employs role-based access control (RBAC), encrypted communication (TLS/SSL), and robust authentication (OAuth2, SSO) to safeguard sensitive data and user activity.

- **Data integrity and backup:** Automated backup and restore mechanisms ensure that data remains consistent and recoverable in the event of failure.
- **Audit and compliance:** All access and changes are logged, supporting audit trails and compliance with industry standards and regulations.

6.4. Reproducibility and Quality Assurance

- **Containerized workflows:** All models and processing pipelines run in reproducible Docker containers, ensuring that results are consistent across environments and over time.

6.5. Usability and Stakeholder Engagement

- **Unified user experience:** Interactive dashboards and configuration UIs (built with Leptos and DATAmaestro) provide consistent, intuitive access for engineers, data scientists, and decision-makers alike.
- **Accessibility:** RESTful APIs, batch and manual data upload, and visualization make the platform accessible for both programmatic and human-centric workflows.

6.6. Advanced Analytics and Decision Support

- **Integrated machine learning:** The architecture supports advanced analytics, statistical modelling, and AI/ML workflows, enabling predictive insights and process optimization.
- **Digital twin support:** By supporting both physics-based and data-driven digital twins, the platform enables real-time monitoring, predictive maintenance, and adaptive control of manufacturing processes.

7. Hybrid modelling strategy: combining Multiphysics and ML models

The hybrid modelling strategy for DILAFACT will be enabled by the process optimization framework described in this deliverable so far. The Multiphysics modelling activities are aimed towards predicting the temperature fields, the representative microstructure and the process-induced deformations during PBF and DED process, and the thermal fields during laser surface treatments. Since Multiphysics models have a large computational time, within DILAPRO several reduced models are being developed for enabling optimization. The reduced models span from physics-based models to machine learning based models to advanced statistical models. The optimized parameters achieved through these reduced models can then be verified through high-fidelity simulations, if needed.

Four categories of these reduced models will be briefly described, focusing on how the optimization framework enables their deployment – the details of these reduced models can instead be found in Deliverable 2.4.

Deformation Prediction Workflow

In this category of modelling for process optimization, the idea is to use inherent strain based reduced models to predict the expected deformation during AM production, as well as allow for distortion compensation simulations if relevant. As DILAPRO involves multiple production lines, a corresponding inherent strain material model will be required for each material at each production line. The inherent strain material model is generated by producing a set of calibration parts, measuring the distortions and then calibrating the “strain scaling factor” in successive simulations till the prediction error is within tolerance bounds. The calibrated model can be run in “thermal strain” mode or “assumed strain mode”, with the former generating the strain map based on the thermal simulation of the scan pattern while the latter utilizes uses the scan pattern to directly assign inherent strains. An example of the deformation prediction simulation and distortion compensation can be seen in Fig 7.1, where the picture on the left shows the outline of the original design and the deformed geometry while the picture on the right shows the distortion compensated geometry to be printed to achieve the correct shape post-production.

In the deployment scenario, the optimization framework can be used to select/determine process parameter sets and subsequently launch distortion prediction simulations using the inherent strain material model corresponding to the production line of interest. The displayable outputs will correspond to .vtk files containing predicted distortion values and deformed shape. The current version of this modelling workflow makes use of the commercial ANSYS Additive

tools, thus requiring either a stand-alone installation of ANSYS Additive on a virtual machine OR a Docker Image with the appropriate network/floating license information. The trained models will be packaged within Docker Images, which can be deployed using the workflow described above in Section 5.1. The current ANSYS Additive tool assumes that the part is produced using PBF technology for iterative distortion compensation – however the same procedure can be applied for DED production lines as well using single distortion compensation.

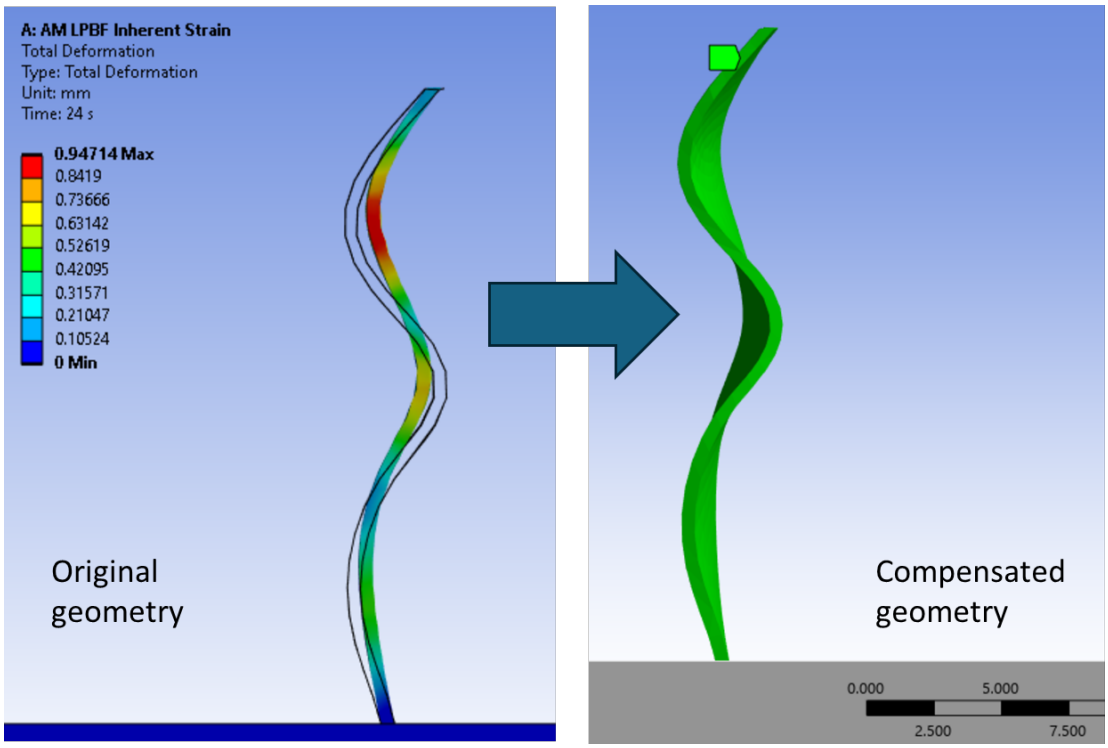


Table 5 : Deformation models for the optimization framework showing the original geometry, the deformed shape and the distortion compensated geometry – the above figure is taken from an example simulation case available with the ANSYS Additive software

Microstructure Classification & Segmentation

In this category of modelling for process optimization, the idea is to use microstructure data (i.e. LOM, EBSD, SEM/TEM data) generated through experiments and physical simulations as ground-truth for developing machine learning models for microstructure classification. For a wide range of process parameters sets for each material, the machine learning model learns the corresponding features that define an acceptable microstructure as opposed to a poor quality microstructure. The semi-supervised learning is enabled through labelled images highlighting desirable and undesired structures within micrographs and EBSD images. Further, experimental data of the mechanical properties (tensile strength/nano-hardness/fatigue properties) achieved using the process parameters sets corresponding to the labelled microstructures will act as the criteria for determining whether the microstructure is acceptable i.e. the classification category.

During the DILAPRO project, these microstructure classification models will be continuously improved as the physical simulation and experimental data for the different production lines are generated.

For the tasks of microstructure image classification and segmentation, the current approach relies on training task-specific CNN-based models. In particular, YOLO models are employed for classification, while DeepLab models are used for segmentation. The relevant details can be found in the table below.

	Classification	**Segmentation**
Data format	Image	Image
Annotation	Bounding boxes on the images with class information	Polygons on the images with class information
Model	YOLO	DeepLab
Inference Results	Bounding boxes	Polygons
Training Device	GPU	GPU
Inferencing Device	GPU/CPU	GPU
Inference speed	Fast (<0.5s)	Slow (>5s)

Table 6 : Comparison between Classification and Segmentation models in terms of data format, annotation, architecture, hardware, and inference speed

In the deployment scenario, the trained machine learning model will evaluate 2D microstructure images, either collected experimentally or generated by simulations, for a given process parameter set and material, to determine whether these could be used for producing the part. The trained models will be packaged within Docker Images, which can be deployed using the workflow described above in Section 5.3.

Representative Surface Prediction

In this category of modelling for process optimization, the idea is to use generative AI tools to predict the expected surface characteristics for the AM and laser surface treatment processes. The deep learning-based conditional denoising diffusion probabilistic model developed for this

purpose is trained on real surfaces produced by different process parameters and material combination. The training dataset could/will be expanded in the future using a combination of real experiments and high-fidelity CFD simulations of single layer production with PBF and/or DED. The relevant details for the generative AI tool are summarized in the table below.

	Surface generation
Data format	Image
Condition	2d: samples/locations
Model	U-Net + Diffuser
Inference Results	Generated new surfaces
Training Device	GPU
Inferencing Device	GPU
Inference Speed	Slow (>15s)

Table 7 : Surface Generation model specifications

In the deployment scenario, the model can be used for process optimization by predicting the expected representative surface topographies and thereby allowing for planning of the successive post-processing operations (such as machining and/or finishing). Fig 7.2 shows a set of real surface topographies as well as those predicted by the generative AI tool. This tool would be used in the process optimization framework following the workflow described in Section 5.1.

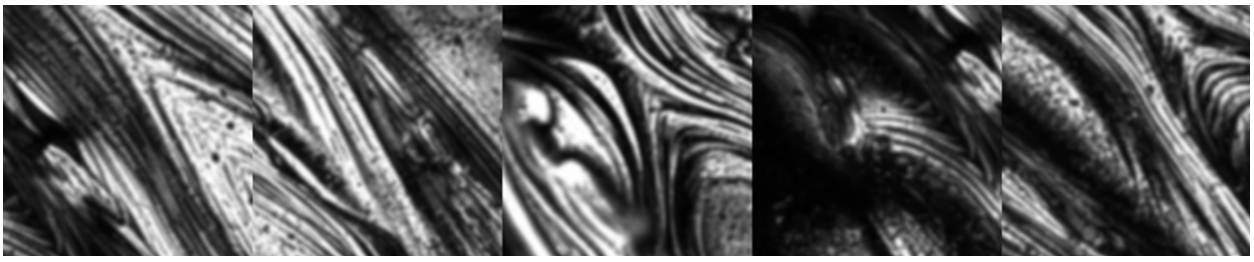


Table 8 : Grayscale height maps showcasing the surface topographies emerging from PBF of 316L. The figure on the extreme left is produced by the generative AI tool.

Process parameter selection

In this category of modelling for process optimization, the idea is to use the Gaussian process models developed for process parameter selection. The currently developed GP models focuses on the prediction of likelihood of balling defects in PBF and the likelihood of process success in DED. The GP models have been developed initially on data collected from literature for PBF as

well as through dedicated experimental campaigns for process window identification at one of the DED production lines. The dataset is supplemented by high-fidelity CFD simulations at selected process parameter set values with high prediction uncertainty. Fig 7.3 visualizes the power-scan speed parameter space and the corresponding Gaussian Mixture model for defect prediction (balling and humping likelihood).

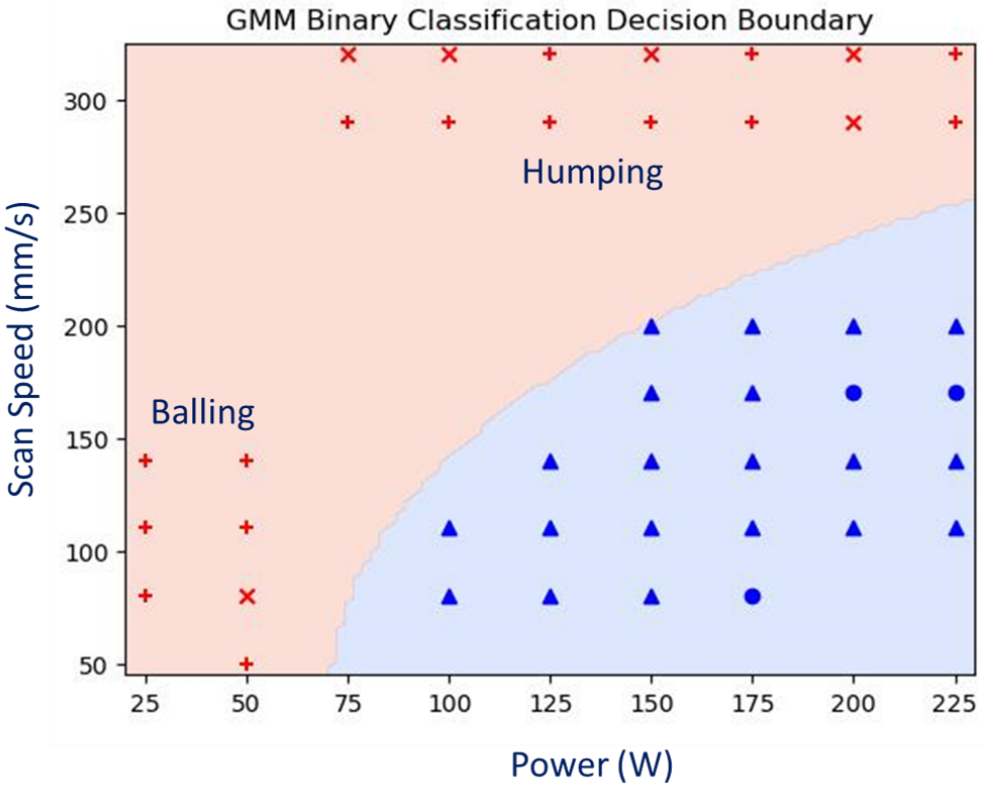


Table 9 : Power vs scan speed plot showing the expected zone with low likelihood of defect (blue) and the zones where balling or excessive humping (red) are expected. The triangles and plus symbols correspond to data used to extract the model, while the circles

In the deployment scenario, the different GP models can be directly implemented using the workflow established in Section 5.1 for non-time series processing. In addition to the predicted output of the GP, the models will also provide an estimation of uncertainty for the prediction, and the users will have the option to launch a dedicated corresponding numerical simulation to get a better evaluation. An example of such a fast GP model implementation can be found at https://laserweld.streamlit.app for porosity prediction during laser-welding.

8. Conclusion

In summary, Deliverable 2.5 establishes a comprehensive and strategic foundation for process optimization within the laser-based manufacturing context. The described DILAFACT platform provides robust, scalable, and modular architecture designed to meet diverse industrial demands effectively. By combining state-of-the-art Multiphysics simulations with powerful machine learning tools, it empowers stakeholders to make informed decisions, optimize manufacturing parameters, and ensure high-quality production outcomes.

Looking forward, the continued evolution of the DILAFACT platform will focus on further enhancing its scalability and adaptability to accommodate emerging industrial needs and technologies. Future developments will emphasize increased automation, integration of new data modalities (e.g. live sensor/monitoring data), and the continuous improvement of predictive capabilities.

Appendix A

A1. Test environment images of the Dilafact platform

- **Homepage of the platform**

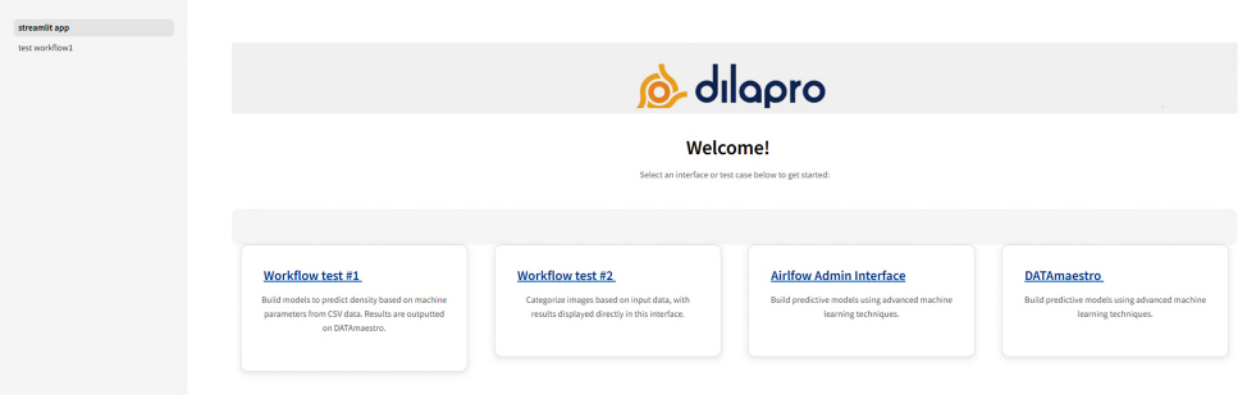


Table 10 : Homepage of the platform

- **Page for selecting inputs and running the model**

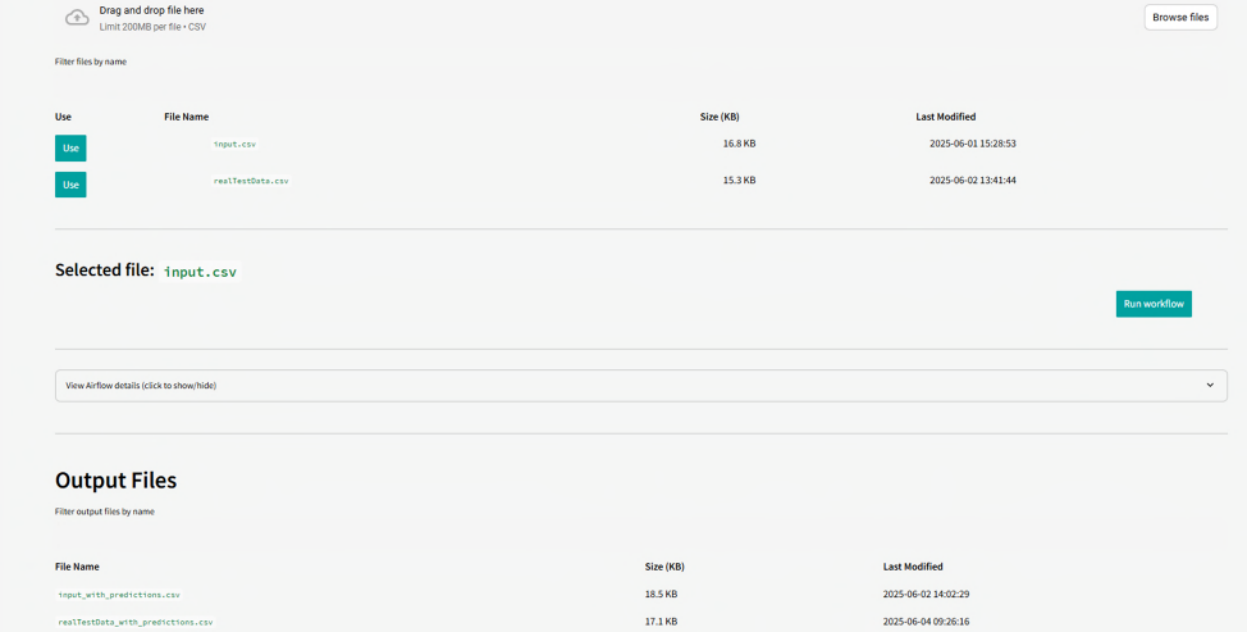


Table 11 : Page for selecting inputs and running the model

- **<u>View of Apache Airflow for model orchestration</u>**

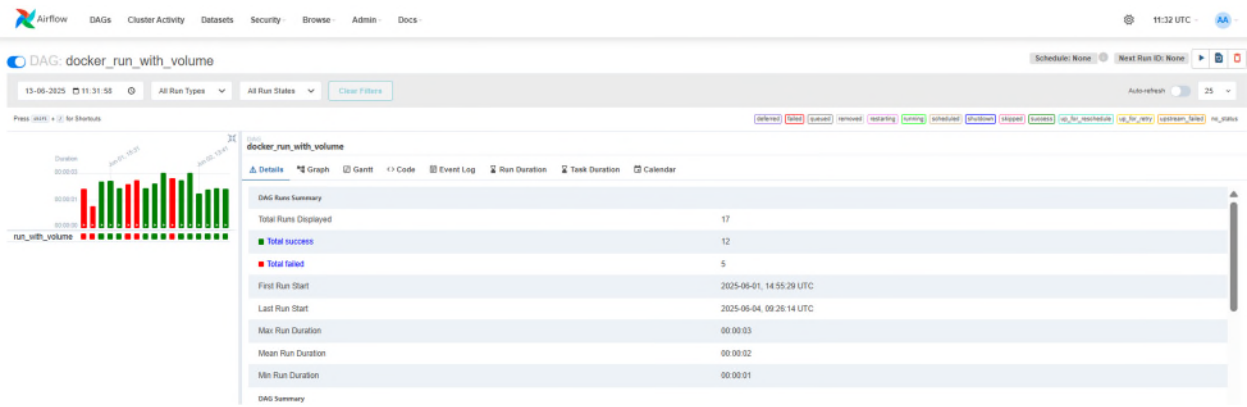


Table 12 : View of Apache Airflow for model orchestration

- **<u>Example of results for density prediction</u>**

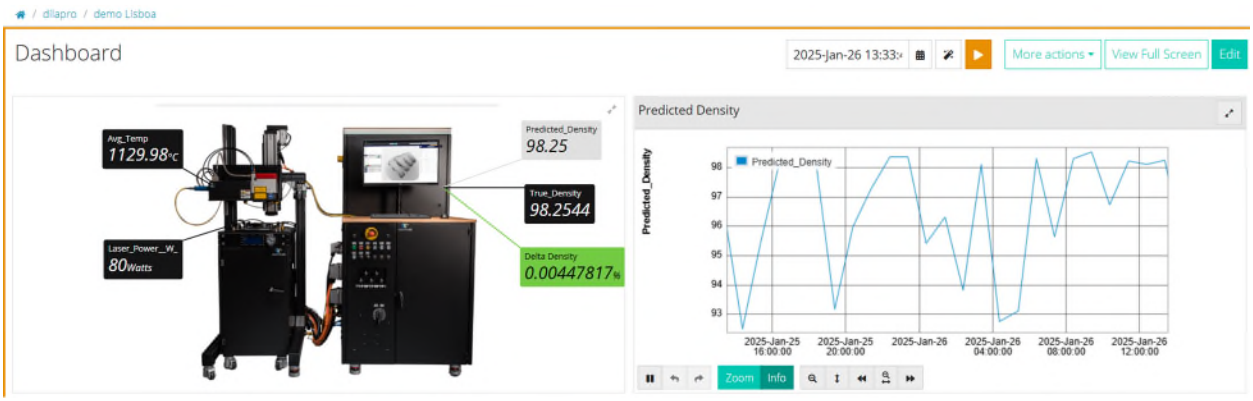


Table 13 : Example of results for density prediction