DEGREE PROJECT FOR MASTER OF ENGINEERING SCIENCE
WITH SPECIALIZATION IN AI AND AUTOMATION
DEPARTMENT OF ENGINEERING SCIENCE
UNIVERSITY WEST

Data Management Flow and Standards to Enable AI in Additive Manufacturing

Author:
Akshayakumar Srinivasan

Supervisor:
Amit Kumar Mishra

2026

A THESIS SUBMITTED TO THE DEPARTMENT OF ENGINEERING SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING SCIENCE WITH SPECIALIZATION IN AI AND AUTOMATION
AT UNIVERSITY WEST
September 29, 2026

Author:	Akshayakumar Srinivasan
Examiner:	Yongcui Mi
Supervisor:	Amit Kumar Mishra
Program:	Master Programme in AI and Automation
Main field of study:	Engineering Science
Credits:	60 Higher Education credits
Keywords:	Data Management, FAIR Principles, AI-Ready Data,Data Pipeline Standards
Template:	University West, EXA620
Publisher:	University West, Department of Engineering Science S-461 86 Trollhättan, SWEDEN Phone: + 46 520 22 30 00 Fax: + 46 520 22 32 99 Web: www.hv.se

Summary

Additive manufacturing generates a large amount of experimental data from process monitoring, sensors and different measurement systems. Managing this data in a structured way is important when the data needs to be reused for analysis and machine learning. Without proper organisation, metadata, validation and storage methods, it can become difficult to find, understand and prepare the data for future use.

The main aim of this thesis is to design and implement a data management framework that can support the preparation of additive manufacturing experimental data for AI and machine learning applications. The proposed framework combines FAIR-aligned metadata, data validation, storage management, compression methods, version control and user-facing interfaces into one workflow.

The system was implemented and evaluated using publicly available additive manufacturing benchmark datasets, with a focus on datasets from the NIST AM-Bench initiative. Different compression and data reduction methods were studied to understand their effect on storage requirements and machine learning performance. Acceptance Test Procedures were also defined to evaluate important parts of the proposed data management system.

The results show that a structured data management approach can help organise additive manufacturing data and make the data easier to prepare for machine learning applications. The work also demonstrates how FAIR principles and data management practices can be combined with a practical data pipeline for experimental AM data

The complete implementation and source code repository are available on GitHub:
GitHub Repository

Preface

This thesis was completed as part of the Master's programme in AI and Automation at University West. The work focuses on the challenges involved in managing experimental additive manufacturing data and preparing it for artificial intelligence and machine learning applications.

The topic was chosen because additive manufacturing experiments can generate different types of data from sensors, machines and measurement systems. Without a structured way of managing this information, it can become difficult to reuse the data and prepare it for further analysis.

During the project, I focused on developing a practical data management framework that combines metadata, data validation, storage, compression and machine learning preparation. Publicly available additive manufacturing benchmark datasets were used for the experimental work, with NIST AM-Bench providing the main data source for the evaluation.

I would like to thank my supervisor, Amit Kumar Mishra, for his guidance and feedback throughout the thesis work. I am also grateful to University West for providing the resources and environment needed to complete the project.

Affirmation

This master's degree report, Data Management Flow and Standards to Enable AI in Additive Manufacturing, was written as part of the master's degree work required to obtain a Master of Science with specialization in AI and Automation at University West. All material in this report that is not my own is clearly identified and used in an appropriate and correct way. The main part of the work included in this degree project has not previously been published or used for obtaining another degree.

The research questions, system design, FAIR operationalization, Acceptance Test Procedures, experimental evaluation, and written content in this thesis are the work of the author. The source code was developed independently by the author

Akshayakumar Srinivasan Date

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Aim . . . 2
- 1.3 Research Questions . . . 2
- 1.4 Contributions . . . 3

2 Background and Related Work **4**
- 2.1 Background . . . 4
  - 2.1.1 FAIR Data Principles . . . 4
  - 2.1.2 Data Quality and Data Validation . . . 4
  - 2.1.3 Metadata and Ontologies . . . 5
  - 2.1.4 Data Compression and Dimensionality Reduction . . . 5
  - 2.1.5 Machine Learning Operations (MLOps) and Data Pipelines . . . 6
  - 2.1.6 Industry 4.0 and Industrial Data Architecture . . . 6
  - 2.1.7 Process Monitoring and Measurement Science . . . 6
- 2.2 Related Work . . . 8
  - 2.2.1 FAIR Data Stewardship and Data Governance . . . 8
  - 2.2.2 Data Quality Assessment and Validation . . . 8
  - 2.2.3 Metadata Standards and Ontologies in Manufacturing . . . 8
  - 2.2.4 Data Compression and Lossy Reduction in ML Contexts . . . 9
  - 2.2.5 MLOps, Data-Centric AI, and Pipeline Architectures . . . 9
  - 2.2.6 Industrial Data Architectures and Industry 4.0 . . . 9
  - 2.2.7 Additive Manufacturing Process Monitoring and Metrology . . . 10
  - 2.2.8 Data Archiving and Long-term Preservation . . . 10
  - 2.2.9 Cross-cutting Themes and Research Gaps . . . 10
  - 2.2.10 Research Gap . . . 11
  - 2.2.11 Positioning of Current Research . . . 11

3 Research Methodology **13**
- 3.1 Research Questions . . . 13
- 3.2 Research Design . . . 13
  - 3.2.1 Overall Research Approach . . . 14
  - 3.2.2 Justification of Research Design . . . 14
- 3.3 Data Gathering Methods . . . 14
  - 3.3.1 Stage 1: Benchmark Datasets for Compression Experimentation (RQ1) 15

3.3.2 Stage 2: System Data for Implementation and Technical Validation (RQ2) . . . 16
3.4 Data Analysis Methods . . . 16
3.4.1 Stage 1: Data Compression Impact Evaluation (RQ1) . . . 16
3.4.2 Stage 2: System Implementation and Technical Evaluation (RQ2) . . . 18
3.5 Tools, Equipment, and Resources . . . 20
3.5.1 Software Tools and Technologies . . . 20
3.5.2 Hardware Resources . . . 21
3.5.3 Data Resources . . . 21
3.6 Sources of Error and Limitations . . . 21
3.6.1 Compression Evaluation Limitations (RQ1) . . . 21
3.6.2 Usability Evaluation Limitations (RQ2) . . . 22
3.6.3 System Implementation Limitations . . . 22
3.7 Experimental Design Summary . . . 22

4 Data Management System Implementation 24
4.1 System Architecture Overview . . . 24
4.2 Data Model and Schema Design . . . 26
4.2.1 Core Entities . . . 26
4.2.2 Process Parameters Schema . . . 28
4.2.3 Quality Metrics Schema . . . 28
4.3 Backend Implementation . . . 29
4.3.1 Data Ingestion Module . . . 30
4.3.2 Metadata Management System . . . 31
4.3.3 Data Versioning and Lineage Tracking . . . 31
4.3.4 Query and Retrieval Engine . . . 31
4.4 Frontend Implementation . . . 32
4.4.1 Technology Stack . . . 32
4.4.2 Dashboard Interface . . . 32
4.4.3 Experiments Management Interface . . . 33
4.4.4 Add Experiment Interface . . . 33
4.4.5 Analytics and Visualization Interface . . . 33
4.4.6 ML Export Interface . . . 35
4.5 Data Compression and Storage Optimization . . . 36
4.5.1 Compression and Data Reduction Terminology . . . 36
4.5.2 Data Formats Used for Storage . . . 40
4.5.3 Storage Size Evaluation . . . 40
4.5.4 Storage and Data Management Integration . . . 41
4.6 Integration and Validation . . . 41
4.7 Deployment and Operational Considerations . . . 41
4.7.1 Deployment Architecture . . . 41
4.7.2 Deployment and Maintenance Considerations . . . 43
4.7.3 Security and Access Control . . . 43
4.8 Discussion . . . 43
4.8.1 Contributions Beyond Requirements . . . 43

4.8.2 Limitations and Future Work . . . 44
4.9 Summary . . . 44

5 Results and Discussion **45**
5.1 Experimental Dataset and Evaluation Setup . . . 45
5.2 Data Ingestion and Data Quality Results . . . 46
5.3 Data Quality and Preprocessing Results . . . 48
5.4 Compression and Data Reduction Results . . . 48
5.5 Material Distribution and Dataset Characteristics . . . 50
5.6 Process Parameter Analysis . . . 50
5.6.1 How These Correlations Should Be Read . . . 51
5.7 Machine Learning Results . . . 52
5.8 Experiment Management and Data Entry . . . 54
5.9 Analytics and Data Export . . . 55
5.10 API and Data Retrieval Results . . . 56
5.11 Data Versioning and Integrity Results . . . 57
5.12 Role-Based Access Control Results . . . 57
5.13 Deployment Results . . . 57
5.14 Acceptance Test Results . . . 58
5.15 Discussion of Results . . . 58
5.16 Limitations and Validity . . . 60
5.17 Summary . . . 61

6 Conclusion **63**
6.1 Summary of Contributions . . . 63
6.2 Answering the Research Questions . . . 64
6.2.1 RQ1: Quantitative Compression Impact . . . 64
6.2.2 RQ2: Infrastructure and Interfaces for Non-Specialist Users . . . 64
6.3 Addressing the Research Aims . . . 65
6.4 Limitations . . . 66
6.5 Reflections and Lessons Learned . . . 66
6.5.1 What I Would Do Differently . . . 67
6.6 Future Work . . . 67
6.7 Concluding Remarks . . . 68

Bibliography **69**

1 | Introduction

1.1 Motivation

AI is being used more and more in fields like healthcare, manufacturing, finance and public administration [1]. In all of these areas, how well an AI system performs and how much it can be trusted depends just as much on the quality and governance of its data as it does on how clever the underlying algorithm is. A lot of research effort over the past decade has gone into better model architectures, but the more time I spent reading around this topic and working on this thesis, the more I kept running into the same observation from people actually deploying these systems: the biggest practical bottleneck usually isn't the model. It's missing metadata, inconsistent validation, poor traceability and the general lack of any standard way of checking data quality before it gets used [2].

This is very much the case in additive manufacturing (AM), where process monitoring plays a central role. Making a process more stable, improving part quality and scaling it up all depend on experimentation and on being able to actually work with the data that experimentation produces [3]. AM machines, sensors and monitoring systems generate a lot of experimental data, often at high frequency and from several sources at once, and without some kind of consistent structure that data is genuinely hard to organise, understand, reuse or feed into an AI or machine learning pipeline.

I don't think this is specific to any one lab or company. Experimental manufacturing data tends to end up in different formats, with incomplete metadata, inconsistent naming, and different ways of describing the same process parameters depending on who collected it. All of that makes it harder than it should be to combine datasets or reuse them later.

Part of what motivated this thesis is that the existing solutions I found each cover only part of the problem. Wilkinson et al. (2016) put forward the FAIR principles – Findable, Accessible, Interoperable and Reusable – which are probably the most widely cited set of principles for data management and stress the importance of machine-actionable metadata [4]. RAMI 4.0 and the broader Industry 4.0 literature look at how data should flow through an organisation in a hierarchical way, but they don't say much about what data quality or metadata granularity should actually look like for AI applications specifically. NIST and ASTM, meanwhile, have been working on standardising additive manufacturing more generally, but their focus is on process monitoring, traceability and material characterisation rather than AI-readiness as such. There is also a growing body of work on AI pipelines and MLOps arguing that industrial AI projects tend to fail because of how the data is managed rather than because of the model architecture – but that literature is mostly grounded in software engineering contexts, not experimental manufacturing environments.

What I kept coming back to is that nobody seems to have connected these pieces. There isn't really a single operational data flow model that combines FAIR-aligned metadata, the constraints of an industrial architecture, AM-specific process requirements, and the reproducibility needs of an ML pipeline, all in one place. Most of what I read was either fairly conceptual or narrowly scoped to one domain, without much empirical testing of an end-to-end, AI-enabling data pipeline in an actual experimental manufacturing setting. That gap is what this thesis tries to address, and I think it represents a real missed opportunity for manufacturing organisations that want to treat their experimental data as something they can actually build AI applications on top of.

1.2 Aim

My main objective in this thesis is to design, build and evaluate a data management workflow, together with the infrastructure to support it, for preparing experimental additive manufacturing data for machine learning and AI use. I want to be clear that this thesis is not about developing new machine learning algorithms or improving existing model architectures – that is deliberately out of scope. The focus throughout is on the data layer: infrastructure, metadata, standardisation, compression and quality assurance.

What I am aiming for is a workflow that connects the point where experimental data is collected to the point where it is actually ready for AI use, with measurable validation steps along the way rather than just a general claim that the data is "good enough." I implemented and evaluated this framework using publicly available additive manufacturing benchmark datasets, for the reasons I explain in Chapter 3.

1.3 Research Questions

This thesis is organised around two research questions:

1. **What is the quantitative impact of lossless compression, data reduction, and dimensionality reduction on ML model performance?**

2. **What backend infrastructure and user-facing interfaces can support the management and preparation of additive manufacturing data for AI applications?**

Both questions come directly out of the challenges I described in the motivation above, and together they cover both the technical implementation of the system and the evaluation of the data-reduction methods it relies on.

I want to be precise here about what each term in RQ1 covers, since they get used loosely elsewhere in the literature. *Lossless compression* (Gzip, and Parquet with Snappy compression) reduces file size without discarding any information. *Data reduction* (downsampling) reduces the number of observations. *Dimensionality reduction* (PCA) reduces the number of features while retaining a defined share of the variance in the data. These three categories are what the evaluation in Chapter 5 covers; feature extraction and several other techniques

were considered during scoping but not carried through to the final experiments, as I explain in Chapter 3.

1.4 Contributions

The main things I set out to contribute with this thesis are:

- **A practical data-management workflow for additive manufacturing data** that brings FAIR-aligned metadata [4], ideas from industrial data architecture [5] and AM-specific requirements together into one operational data flow, covering data ingestion, validation, metadata handling, storage, compression and data reduction, machine-learning preparation, version control, role-based access control, API services and visualisation.
- **Ten Acceptance Test Procedures (ATPs) with measurable criteria**, defined before the experiments were run, which turn the general notion of "AI-ready data" into something concrete and testable – with defined thresholds for ingestion throughput, metadata completeness, query performance, permissible change in model performance under compression, and preprocessing time [6].
- **An empirical comparison of four compression and data-reduction techniques** – Gzip and Parquet with Snappy (lossless compression), downsampling (data reduction) and PCA (dimensionality reduction) [7, 8] – measuring for each one both the storage saving and the resulting change in R^2 for the models evaluated, so the storage/performance trade-off can be compared on a common basis.
- **A working end-to-end implementation**, comprising a FastAPI backend (ingestion with column-alias mapping, validation, storage, SHA-256 record versioning, role-based access control and a versioned REST API) together with browser-based interfaces for upload, experiment management, analytics and machine-learning export, deployed and exercised end to end.
- **An evaluation of the workflow against the NIST AM-Bench dataset** [9], reporting the measured outcome of each ATP together with the limitations of what could be tested – including the large-scale test that was not completed and the size of the material-labelled subset behind the machine learning results.
- **A publicly available, documented implementation**, kept in a version-controlled Git repository with code documentation, so that other researchers or practitioners can look at, validate or build on this work [6].

2 | Background and Related Work

2.1 Background

Before getting into the related literature, I want to lay out the core concepts this thesis builds on. This section is not meant to be exhaustive – it covers the ideas I actually leaned on while designing the system in Chapter 4, so that the design choices later make sense in context.

2.1.1 FAIR Data Principles

The FAIR principles are probably the single most influential framework in modern data management, and they run through most of this thesis in one way or another [4]. FAIR stands for Findable, Accessible, Interoperable and Reusable – four properties that, together, are meant to describe what makes a dataset genuinely reusable rather than just technically available.

In practice, findability comes down to giving data persistent identifiers and enough metadata that someone (or something) can actually locate it later. Accessibility means the data can be retrieved through a standard, well-documented method, ideally with clear rules about who is allowed to access it. Interoperability is about using shared formats and vocabularies so that data from different sources can be combined without a lot of manual translation. Reusability then depends on all of the above being documented well enough, with clear enough usage terms, that someone other than the original creator can pick the data up and actually do something useful with it [4].

What I find useful about FAIR is the shift in mindset it represents: from thinking about data storage as the goal, to thinking about machine-actionable metadata as the goal, so that finding, validating and combining datasets doesn't depend entirely on someone remembering where things are and what they mean [4]. Mons et al. later applied these ideas within distributed scientific infrastructures and made a similar point – that FAIR compliance isn't really a one-off achievement, it's an ongoing investment in metadata standardisation, persistent identifiers, and the governance to keep both up to date [10].

2.1.2 Data Quality and Data Validation

Data quality is not a single number – it usually gets broken down into things like completeness, accuracy, consistency, timeliness and validity [11]. In a machine learning context, this matters more than it might first appear, because a model is only as trustworthy as the data

it was trained on. Poor data quality doesn't just mean noisy predictions; it can also produce models that behave unpredictably or that simply fail to generalise once conditions shift even slightly [11, 6].

Data validation is the practical side of quality: checking, before data gets used, that it actually meets some defined bar. That typically means schema checks (is the data in the format and structure it's supposed to be in), integrity checks (is it complete and internally consistent), statistical checks (does the distribution look reasonable, are there obvious anomalies), and finally domain checks, which usually require someone who actually understands the process to say whether a value makes physical sense.

This last point matters a lot for manufacturing data specifically. Sensor streams are numerous, varied, and often noisy, and a bad reading that slips through validation can end up quietly skewing an analysis or a model downstream.

2.1.3 Metadata and Ontologies

Metadata is, in the simplest sense, data about data – it's what turns a raw file into something a person or a system can actually make sense of. Good metadata schemas go beyond just describing structure; they also capture provenance, quality, ownership and intended use [4, 10]. Ontologies take this a step further by giving a formal, machine-readable vocabulary for a domain, which is what lets different systems agree on what a term actually means.

For metadata to be genuinely useful to automated systems – not just to a human reading a README file – it needs to follow standardised schemas. In a manufacturing context specifically, that usually means capturing the experimental conditions, machine settings, material specifications and the operator involved, since without those details the data is very hard to interpret or reuse later.

2.1.4 Data Compression and Dimensionality Reduction

Compression methods fall broadly into two camps: lossless, where nothing is thrown away and the original data can always be reconstructed exactly, and lossy, where some information is deliberately sacrificed in exchange for a smaller file. Gzip and LZ4 are common lossless options; they're safe for archiving but tend to give only moderate size reductions. Lossy approaches like downsampling or quantisation can shrink files much further, but exactly how much that costs downstream needs to be measured rather than assumed [7].

Dimensionality reduction is a related but distinct idea – techniques like PCA, general feature extraction, or autoencoders take high-dimensional data and represent it in fewer dimensions while trying to preserve the structure that actually matters. These methods sit somewhere between storage efficiency and information preservation, which is exactly the kind of trade-off that shows up a lot when dealing with high-frequency sensor data from manufacturing processes [7, 8].

The part that I think is still genuinely underexplored is quantifying, in concrete numbers, how much compression a given machine learning task can tolerate before it starts to hurt. That's one of the questions this thesis tries to answer directly in Chapter 5.

2.1.5 Machine Learning Operations (MLOps) and Data Pipelines

MLOps is the collection of practices and tooling that make it possible to run machine learning reliably at scale, rather than as a one-off notebook experiment. One of the clearest lessons from this literature is that a production machine learning system is mostly not the model – it's the surrounding infrastructure for collecting data, engineering features, training, deploying and monitoring [6, 2].

Sculley et al.'s well-known paper on hidden technical debt in machine learning systems made this point sharply: the actual model code tends to be a small fraction of a real system, while the bulk of the complexity and risk sits in data collection, preprocessing and infrastructure [6]. In their account, most production failures trace back to data dependencies and data quality issues rather than the model itself – which is a strong argument for treating the data layer as a first-class concern rather than an afterthought.

A reasonably complete ML data pipeline, in this view, covers collection, ingestion, inspection, preprocessing, feature engineering, training, testing, deployment and monitoring – and each of those steps needs to be reliable and repeatable on its own, not just when everything happens to go right.

2.1.6 Industry 4.0 and Industrial Data Architecture

Industry 4.0 is the umbrella term for how cyber-physical systems, data integration and automation are reshaping manufacturing. The Reference Architecture Model Industrie 4.0 (RAMI 4.0) is probably the best-known attempt to describe this in a structured way, laying out how industrial data should flow through layers, from individual device communication all the way up to enterprise business systems [5].

What RAMI 4.0 gives you is a shared vocabulary for talking about system integration and communication protocols. What it doesn't really give you is much detail on data quality or metadata granularity for AI-specific use cases, particularly in experimental or research-oriented manufacturing settings where the requirements can look quite different from a stable production line.

2.1.7 Process Monitoring and Measurement Science

In additive manufacturing specifically, process monitoring means systematically collecting and analysing sensor and machine data during a build, mainly to catch instability, detect defects and improve the process over time [3, 12]. Metrology – the science of measurement – underpins all of this, since it's what gives you confidence that a measurement is accurate, traceable and worth analysing in the first place.

Bodies like NIST and ASTM have put a lot of effort into metrological standards for AM, covering measurement uncertainty, calibration and data traceability [3]. These standards are what make it possible to compare AM data across different machines, labs and time periods with any confidence.

Where I think there's still a gap is that these metrology standards are focused on physical measurement quality – accuracy, traceability, calibration – rather than on whether the

resulting data is actually well-structured and complete enough to train a machine learning model on.

2.2 Related Work

Having covered the background concepts, I now want to look specifically at prior work that sits closer to what this thesis actually does. I've grouped this by theme rather than by author, since I think that makes it easier to see where the gaps actually are.

2.2.1 FAIR Data Stewardship and Data Governance

FAIR has become particularly important in the life and physical sciences, where the sheer volume of research data makes reusability a practical necessity rather than a nice-to-have [4]. Wilkinson et al. were the ones who originally framed FAIR as a set of guiding principles for data stewardship, arguing that Findable, Accessible, Interoperable and Reusable data would help both accelerate and extend the useful life of scientific work.

Mons et al. later took this from principle to practice within distributed scientific infrastructures, and their conclusion was that being FAIR-compliant is really an ongoing investment – in metadata standardisation, persistent identifier systems, and the governance needed to keep both maintained over time [10].

What I noticed reading through this literature is that almost all of it is written with structured, curated scientific datasets in mind. How FAIR principles hold up against high-volume, heterogeneous industrial sensor data is still a fairly open question. That gap – adapting FAIR to a setting with very different data volume, diversity and acquisition speed than a typical research repository – is one of the things this thesis tries to address directly.

2.2.2 Data Quality Assessment and Validation

Data-quality frameworks have been developed and refined across a number of domains. In healthcare informatics, dimensions like accuracy, completeness, consistency and timeliness have been fairly well standardised and quantified [11]. In data warehousing, quality checking is treated as a mandatory gate before anything gets used for business intelligence or reporting.

The problem is that most of these frameworks were built around conventional IT systems – databases, warehouses, batch reports – and don't really account for the specific character of industrial sensor data: high acquisition rates, streaming ingestion, multiple sensor modalities feeding in at once, and dependencies between measurements that aren't always obvious. Adapting quality assessment specifically for manufacturing sensor data, with AI-readiness as the goal rather than operational reporting, is where I've tried to extend this line of work.

2.2.3 Metadata Standards and Ontologies in Manufacturing

Manufacturing already has a reasonably mature set of data-representation standards. ISO 10303, also known as STEP, provides formal schemas for product data [13]. Manufacturing Execution System (MES) standards define what shop-floor operational data should look like, and Prognostics and Health Management (PHM) standards define ontologies for tracking equipment condition.

ASTM and ISO have also developed standards specific to additive manufacturing, covering material properties, process parameters and quality characteristics. But these are,

understandably, focused on describing the product and the process – not on data governance for AI applications. Bringing AM process and product standards together with what a machine learning pipeline actually needs from its metadata is one of the more concrete contributions of this thesis.

2.2.4 Data Compression and Lossy Reduction in ML Contexts

There is a reasonable body of work on how compression affects machine learning performance in general. Several studies suggest that, in the right circumstances, lossy compression or data reduction can be applied with only a small effect on model accuracy – particularly when the reduction step is designed to preserve the statistical properties that actually matter for the downstream task [7].

Most of this work, though, comes out of computer vision, where compression, downsampling and feature extraction for images are well studied. How compression affects time-series data from high-frequency sensors – and especially domain-specific signal features, like melt-pool temperature traces in additive manufacturing – is much less explored. That's the specific gap this thesis's compression evaluation in Chapter 5 tries to fill.

2.2.5 MLOps, Data-Centric AI, and Pipeline Architectures

The MLOps literature converges fairly strongly on one point: data quality and pipeline reliability tend to matter more, in practice, than the choice of model architecture. Sculley et al. identified this directly as technical debt, arguing that the cost of data dependencies, quality issues and configuration management usually outweighs the cost of the modelling work itself [6].

Breck et al. built on this with a proposal for continuous training and validation – essentially, treating data validation as an ongoing process that has to run every time new data comes in, not just once at the start of a project [2]. A lot of the published MLOps guidance, including from large industry teams, reflects a similar emphasis on systematically validating both data and models as they evolve in production.

That said, most of this writing is grounded in large-scale software engineering settings – web services, ad systems, recommendation engines – where the infrastructure has to handle enormous, continuously-arriving datasets. There is comparatively little written about how these same MLOps principles translate to a smaller, more contained manufacturing environment, with its own constraints and its own kind of organisational friction. Adapting MLOps thinking to that scale is part of what this thesis is trying to do.

2.2.6 Industrial Data Architectures and Industry 4.0

Industry 4.0 as a field covers digital transformation, cyber-physical systems, and data-driven manufacturing broadly. Kagermann et al. set out the original Industry 4.0 vision and, with it, RAMI 4.0 as a reference architecture for integrating industrial systems [5]; a fair amount of follow-up work has since looked at how to actually implement that vision across different types of manufacturing environments.

Where Industry 4.0 frameworks are strongest is at the level of system design – hierarchical data layers, interoperable systems, integration patterns. Where they're weaker is in the specifics: they rarely spell out exact metadata standards, data-quality procedures, or validation frameworks needed to actually get data AI-ready. Translating the architectural thinking behind Industry 4.0 into concrete data-governance procedures for an experimental manufacturing setting is, again, one of the things I've tried to do in this thesis.

2.2.7 Additive Manufacturing Process Monitoring and Metrology

AM process monitoring and metrology research has produced increasingly sophisticated measurement methods and quality-assurance protocols over the last decade. Mani et al. and Moylan et al. laid a lot of the groundwork here, focusing on measurement science and the standards needed to collect AM process data consistently [3, 12, 9].

The AM-Bench initiative at NIST, which is the dataset this thesis relies on throughout, grew directly out of that work – producing benchmark datasets and standardised measurement procedures that are useful precisely because they are reproducible. But the emphasis in this body of work is still mostly on measurement accuracy, traceability and physical quality, rather than on whether the resulting data is structured and complete in a way that's useful for machine learning.

This thesis builds on those AM metrology foundations but pushes in a slightly different direction – toward preserving the features and statistical properties that matter for training models, and toward reproducibility in an algorithmic rather than purely metrological sense.

2.2.8 Data Archiving and Long-term Preservation

There is also a well-established body of work on digital preservation and long-term data archiving, with core ideas like format migration, metadata preservation, provenance tracking and bit-level preservation strategies [14]. University libraries, domain repositories and other institutional archives have built up considerable expertise and established practice in this area.

Most of this preservation-focused work, though, is oriented toward data that is accessed rarely – true long-term archival storage, not data that a researcher needs to pull up and analyse on a weekly or daily basis. This thesis sits somewhere in between: it needs enough of an archiving mindset to support reproducibility, but also needs the data to stay reasonably fast and easy to access for ongoing analysis and model training, which pure long-term preservation approaches don't really optimise for.

2.2.9 Cross-cutting Themes and Research Gaps

Pulling these threads together, a few themes came up repeatedly across the literature:

- **Metadata standardisation is universally agreed to matter, but the specifics don't transfer well between fields.** Science, manufacturing, software engineering and digital preservation all have their own conventions, and none of them map cleanly onto the others [4, 10, 13].

- **Data quality is treated as important everywhere, but the frameworks are domain-specific.** What "good quality data" means, and how you check for it, changes a lot depending on what kind of data you're dealing with [11, 2].
- **Compression and reduction always involve a trade-off, and that trade-off needs to be measured, not assumed.** The right answer depends heavily on the specific domain and application [7, 8].
- **High-level architectures give direction but not implementation detail.** RAMI 4.0 and FAIR are both genuinely useful conceptual frameworks, but neither one tells you exactly what to build [4, 5].
- **The different fields involved – FAIR, Industry 4.0, AM standards, MLOps – have mostly developed in isolation from each other,** even though solving data management for manufacturing AI really requires pulling ideas from all of them at once [4, 6, 3].

2.2.10 Research Gap

Pulling all of this together, the picture I take away from the literature is this: FAIR data stewardship, data-quality frameworks, AM-specific metadata and process standards, compression research, and MLOps/data-centric AI practice all give useful, well-established approaches to their own part of the problem, but they have mostly developed separately from each other. FAIR tells you what reusable data should look like, but not how to validate it for a specific manufacturing process. AM standards tell you how to describe a build process, but not how to structure that data for a machine learning pipeline. MLOps literature argues convincingly that the data layer matters more than the model, but is written for software engineering contexts, not experimental manufacturing. And the compression literature tells you that reduction is possible without much cost to accuracy, but rarely quantifies that cost for AM sensor data specifically.

None of the work I reviewed combines these strands into a single, operational, and empirically tested workflow for additive manufacturing experimental data. That is the gap this thesis is aimed at: not proposing a new principle or a new algorithm, but developing and evaluating a practical workflow that pulls FAIR-aligned metadata, AM-specific structure, and measurable compression trade-offs together into one system that can actually be built, run and tested end to end.

Based on this gap, Chapter 3 sets out the methodology I used to design and evaluate that workflow.

2.2.11 Positioning of Current Research

Given those gaps, what I've tried to contribute with this thesis is:

1. A single operational data-management workflow that pulls together FAIR principles, Industry 4.0 architectural thinking, AM metrology standards and MLOps practices, rather than treating them as separate concerns.

2. A translation of fairly abstract frameworks (FAIR, Industry 4.0) into concrete, measurable steps and acceptance criteria, built specifically for experimental AM data rather than for a generic setting.

3. An empirical, numbers-based look at how compression affects ML performance specifically for AM sensor data, which I hope gives practitioners something more concrete to work from than general intuition.

4. Evidence that FAIR principles can be made operational in high-volume, heterogeneous, rapidly-collected industrial sensor data – extending FAIR outside the curated scientific-repository context it was originally designed for.

5. A working set of tools and interfaces aimed at letting someone without a data-science background manage experimental data in a way that's consistent with AI-readiness requirements.

6. A set of acceptance test procedures that connect these more theoretical frameworks to something that can actually be checked and verified against real system behaviour.

3 | Research Methodology

In this chapter I walk through how I actually approached this thesis – the methods, tools and evaluation procedures I used, roughly in the order I used them. I've tried to explain not just what I did but why I made these particular choices, since I think that context matters as much as the steps themselves.

3.1 Research Questions

Everything in this chapter is shaped by the two research questions I introduced in Chapter 1:

1. **RQ1: What is the quantitative impact of lossless compression, data reduction, and dimensionality reduction on ML model performance?**

   Answering this meant running actual experiments – applying lossless compression, data reduction and dimensionality reduction to the same dataset and measuring what happened to model performance as a result. As I explain in Section 3.4, the evaluation covers four specific techniques across those three categories: Gzip and Parquet/Snappy (lossless compression), downsampling (data reduction), and PCA (dimensionality reduction). What I wanted out of this was a set of practical bounds: how much information can you afford to lose before it actually hurts.

2. **RQ2: What backend infrastructure and user-facing interfaces can support the management and preparation of additive manufacturing data for AI applications?**

   This one is about the technical design and implementation of the system itself – the backend services, data storage, metadata handling, retrieval, and the interface a user actually sees. I evaluated the resulting system against a set of technical acceptance criteria: ingestion, metadata validation, query performance, scalability and data integrity.

3.2 Research Design

I used an applied research design that combines experimental evaluation with actually building a working system, split into two connected phases [15]. The first phase is about evaluating

different data compression and reduction methods and how they affect machine learning performance. The second is about implementing the data management framework itself and evaluating its technical performance.

I deliberately let the first phase inform the second: the results from the compression experiments fed directly into decisions about data storage and preparation in the implemented system. Throughout, I used publicly available additive manufacturing benchmark datasets as my main source of experimental data, for reasons I explain below.

3.2.1 Overall Research Approach

I carried the research out in two main phases.

1. **Phase 1: Empirical Evaluation and Benchmarking** — I focused this phase on evaluating data compression and reduction methods, applying different approaches to additive manufacturing benchmark datasets and measuring the effect on storage requirements and machine learning performance. The goal was to understand the trade-off between shrinking the data and keeping enough information for machine learning to still work well.

2. **Phase 2: System Implementation and Technical Evaluation** — Here I implemented the data management framework itself: ingestion, metadata management, validation, storage, retrieval, compression, and preparing datasets for machine learning. I evaluated the resulting system against technical acceptance criteria covering ingestion, metadata completeness, query performance, scalability, interoperability and data integrity.

3.2.2 Justification of Research Design

The two research questions need somewhat different evaluation approaches. RQ1 calls for quantitative experiments, since the effect of compression and data reduction has to be measured against defined performance metrics rather than argued for. RQ2 is really about the design and implementation of a practical system, so I used technical testing to check whether the main requirements were actually met.

The two phases connect naturally, since the compression experiments from Phase 1 directly inform the storage and data-preparation decisions in the implemented system in Phase 2. I also chose to work with public benchmark datasets throughout, mainly because it makes the whole process easier for someone else to reproduce, and means the results don't depend on access to data I couldn't share.

3.3 Data Gathering Methods

Data gathering happened in two stages, matching the two phases of the research design above.

3.3.1 Stage 1: Benchmark Datasets for Compression Experimentation (RQ1)

Data Sources

I used the following open-access benchmark datasets for the compression evaluation:

- **NIST AM Bench Datasets** [9] — The main dataset I used for the compression experiments came from the NIST Additive Manufacturing Bench (AM-Bench) initiative. These datasets contain publicly available experimental AM data with documented metadata, which made them a good fit for research and benchmarking.

  I picked this dataset mainly because it's publicly available and gave me a reproducible basis for comparing different compression and reduction methods. Before running any experiments, I spent time going through it to understand its structure, size and the variables it contains.

  Preparing the data involved checking the dataset files, going through the available metadata, identifying which variables were relevant, and getting everything into a form I could actually use for the compression and machine learning experiments. I kept the processed versions stored locally so I could reuse the exact same data consistently across different experiments.

- **UCI Machine Learning Repository and Kaggle AM Datasets** [16] — I also reviewed other publicly available AM-related datasets (for example, defect detection and process parameter optimisation) during the initial dataset-selection process, mainly to check what alternatives existed before settling on NIST AM-Bench. I want to be clear that these datasets were part of that selection process only – they were not carried through to the final quantitative evaluation. Every number reported in Chapter 5 comes from the NIST AM-Bench dataset alone.

Data Collection Procedure

1. Download benchmark datasets from public repositories like NIST.

2. Check the integrity of the dataset with checksum verification and review of the metadata.

3. Characterize the properties of the dataset: size, modality (time-series, images, tabular), sampling frequency, number of samples, missing values, and class distributions.

4. Record the characteristics of the dataset in a standardised schema to allow comparative analysis across datasets.

5. Save the datasets in a local repository with versioning for reproducibility.

Sample Sizes and Data Volumes

Public AM-related benchmark datasets in general range from around 100 MB up to several GB, and the NIST AM-Bench dataset itself is on the order of a few GB of thermographic and process data. As noted above, the final quantitative evaluation in this thesis – the compression, machine learning and acceptance-test results reported in Chapter 5 – was carried out entirely on the NIST AM-Bench dataset. I did not run the compression or machine learning experiments on the UCI or Kaggle datasets, so I make no claim about how well the findings generalise across other data sources; that would be a reasonable direction for future work.

3.3.2 Stage 2: System Data for Implementation and Technical Validation (RQ2)

For testing the implemented system, I used the same additive manufacturing benchmark data throughout the project, with the NIST AM-Bench dataset as my main reference for loading, organising and processing experimental data.

I first prepared the dataset into a structure suitable for ingestion, then mapped the relevant metadata and data fields onto the schema used by the system. Once ingested, the data was stored and made available through the backend services and the user interface.

I then tested the system against the Acceptance Test Procedures I had defined, focusing on the technical aspects of the workflow – ingestion, metadata validation, retrieval, scalability, interoperability and data integrity.

I did not have access to a proprietary industrial dataset, and I did not run a formal user study with manufacturing engineers. So I want to be upfront that the results in this thesis are limited to the datasets and technical tests that were actually available to me during the project.

3.4 Data Analysis Methods

3.4.1 Stage 1: Data Compression Impact Evaluation (RQ1)

Compression Techniques Evaluated

I want to be precise about terminology here, since these three categories are technically distinct even though they are often lumped together loosely as "compression":

1. **Lossless Compression** — Gzip applied to the raw CSV data, and Parquet with Snappy compression as a column-oriented alternative. Both reduce file size without discarding any information; the original data can always be reconstructed exactly. Metric: compression ratio (storage size relative to the raw CSV baseline) [7].

2. **Data Reduction (Downsampling)** — Reducing the number of observations by keeping only every nth record, tested at 1:5 and 1:10 ratios. This is a form of lossy reduction: information is deliberately discarded, and unlike compression it cannot be reversed.

3. **Dimensionality Reduction (PCA)** — Principal Component Analysis retaining 95% and 99% of variance. PCA is not a compression method in the same sense as Gzip or Parquet – it reduces the number of features the data is represented by, rather than compressing the storage of the existing features. Metric: explained variance retained [8].

For all three categories, I measured the resulting change in machine learning performance against the same baseline (Section 3.4.1 below), so the three could be compared on a common footing even though they work in fundamentally different ways.

I considered a few additional techniques during the early scoping of this thesis – LZ4 compression, quantisation, smoothing filters, and sliding-window feature extraction – but did not carry them through to the final quantitative evaluation, mainly to keep the comparison focused and to leave enough time to properly validate the four methods above. I see extending the evaluation to these additional techniques as a reasonable direction for future work, and I do not report results for them in Chapter 5.

Compression Impact Testing Methodology (ATP-3 and ATP-4)

To measure how each compression technique affected ML model performance, I followed this process:

1. **Baseline Model Training** — I trained regression models on the uncompressed NIST AM-Bench data to predict tensile strength from the available process features: Random Forest Regression and Linear Regression. These were the only two models I carried through to the final evaluation. I had originally planned to also compare against Gradient Boosting, Support Vector Regression and a small neural network, but narrowed the scope to these two so I could focus my time on validating the compression comparison itself rather than spreading it across many model types. I treat the additional models as a reasonable direction for future work rather than something this thesis reports results for.

2. **Compressed Data Model Training** — Apply each compression or reduction technique independently, then train identical Random Forest and Linear Regression models on the resulting versions of the same dataset.

3. **Performance Comparison** — Compare regression performance using Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and the R^2 coefficient.

4. **Performance Change Quantification** — Because the models evaluated here are regression models rather than classifiers, there is no classification accuracy to measure. I therefore defined the acceptance metric on R^2, as the relative drop in R^2 against the uncompressed baseline:

$$\Delta P = \frac{R^2_{\text{baseline}} - R^2_{\text{reduced}}}{R^2_{\text{baseline}}} \times 100\%$$

where R^2_{baseline} is the coefficient of determination for a model trained on the uncompressed data, and R^2_{reduced} is the value for an identically configured model trained on the compressed or reduced data. Throughout this thesis I refer to ΔP as the *model performance change*, and it is this quantity that the ATP-3 and ATP-4 thresholds are defined against.

5. **Acceptance Threshold** — I set the target at $\Delta P < 5\%$, meaning a model trained on compressed or reduced data needed to retain at least 95% of its baseline R^2 for that method to count as acceptable.

Compression Ratio and Storage Efficiency Metrics

For each compression technique, I defined the compression ratio as:

$$\text{Compression Ratio} = \frac{\text{Size}_{\text{original}}}{\text{Size}_{\text{compressed}}}$$

I calculated this on the full NIST AM-Bench dataset used in the final evaluation. I had originally planned to also check how storage efficiency scaled across a range of dataset sizes (1 GB, 5 GB, 10 GB) as part of the scalability testing under ATP-7, but that testing was not completed in the time available – see Section 3.6 and the discussion in Chapter 5. The compression ratios reported in Chapter 5 therefore reflect a single dataset size rather than a scaling study.

3.4.2 Stage 2: System Implementation and Technical Evaluation (RQ2)

Backend Infrastructure Design and Implementation

To answer RQ2, I implemented the proposed data management framework as an actual working system – backend services, data storage, metadata management, and a user-facing interface, rather than leaving it as a design on paper.

I built the backend with FastAPI, handling data ingestion, metadata, retrieval and the other data management operations, and connected it to the storage solutions so experimental data and its metadata could be managed in a structured way rather than as loose files.

The frontend gives access to the main functions – viewing datasets, managing experiment information, checking metadata, and preparing data for further analysis.

Rather than running a formal user study, I evaluated the system through technical acceptance tests, checking whether the main functions worked as intended and whether the predefined requirements were actually met.

- **Query Latency** (ATP-6): Time to retrieve a data subset. Target: $T_{\text{query}} < 5$ seconds for a 1 GB dataset.

- **Ingestion Throughput** (ATP-1): Data ingestion rate. Target: linear or sub-linear scaling with data volume.

- **Scalability** (ATP-7): I evaluated the system using the dataset sizes available to me. Testing with datasets of roughly 50 GB or larger is something I left as future work, for reasons I explain in Chapter 5.
- **Preprocessing Time Reduction** (ATP-9): I measured the time needed to go from ingested data to an ML-ready format using the implemented workflow.

Frontend Interface Design and Implementation

I designed the frontend with one goal in mind: making the common data-management tasks accessible without forcing anyone to work directly with the underlying code or database.

1. **Progressive Disclosure** — Key controls are prominent; advanced controls are available but not distracting.
2. **Real-Time Validation** — Inline form validation gives instant feedback during data entry, which reduces errors and improves how complete the resulting metadata actually is.
3. **Domain-Specific Workflows** — I used AM manufacturing terminology in the interface language and field labels rather than data-science jargon, since the people using this system are engineers, not data scientists.
4. **Visual Analytics** — Interactive charts (material distribution, quality metrics trends, parameter correlation heatmap) so engineers can explore the data without writing code.

The frontend is made up of the following interfaces, each mapped to a specific user task:

- **Dashboard** — Recent activity and key performance indicators overview.
- **Experiments Management Interface** — Browse, search and manage experimental datasets through a filterable grid view.
- **Add Experiment Interface** — Structured form with real-time validation for input of experiment metadata and uploading raw data files.
- **Analytics Interface** — Interactive visualisations of dataset metrics, material distribution, process parameter distributions and parameter-quality correlation heatmaps.
- **ML Export Interface** — Export ML-ready datasets in one click as CSV, Parquet or JSON with optional material filtering and preprocessing options.

Integration Testing

I ran integration tests across the complete system, covering the acceptance test procedures most relevant to RQ2:

1. Data ingestion completeness and integrity (ATP-1).

2. Metadata compliance and schema validation (ATP-2).
3. Query and retrieval efficiency (ATP-6).
4. Scalability with increasing data volume (ATP-7).
5. Cross-project interoperability (ATP-8).
6. Data usability and preprocessing time reduction (ATP-9).
7. Data loss risk assessment (ATP-10).

3.5 Tools, Equipment, and Resources

3.5.1 Software Tools and Technologies

1. **Programming Languages** — Python (primary), Bash (scripting), SQL (database queries) [17].
2. **Data Processing Libraries** — Pandas, NumPy, Scikit-learn for data manipulation and analysis.
3. **Machine Learning Frameworks** — Scikit-learn for model training and evaluation. The Random Forest and Linear Regression models reported in Chapter 5 are both scikit-learn implementations; no deep-learning framework was used in the final evaluation.
4. **Compression Libraries** — gzip and the Parquet/Snappy codec for lossless compression; scikit-learn's PCA implementation for dimensionality reduction [7].
5. **Storage and Database Systems** — SQLite for the development and evaluation configuration, and PostgreSQL (accessed through SQLAlchemy) for the production configuration. Raw uploaded files are kept on the file system and referenced from the database records.
6. **Data Version Control** — A custom version-control module implemented as part of this project, using SHA-256 hashing to create tamper-evident version snapshots of each record [6]. Git is used separately for source-code version control.
7. **Web Framework** — FastAPI for backend API and web dashboard.
8. **Frontend Technologies** — HTML, CSS and vanilla JavaScript for the web-based interfaces, with Chart.js and Plotly.js for the charts and interactive visualisations.
9. **Development Tools** — Git for source code version control, Docker for reproducible environments, Jupyter notebooks for exploratory analysis.

3.5.2 Hardware Resources

1. **Development Environment** — Standard workstation (CPU, 16+ GB RAM, 1+ TB storage) for algorithm development and testing.

2. **Experimental Environment** — Server-class hardware (multiple CPUs, 64+ GB RAM, 10+ TB storage) was planned for large-scale compression benchmarking at the 50 GB scale. As noted in Section 3.6, that large-scale test was not completed within the project period, so this hardware was not actually exercised at that scale; the compression and machine learning results in Chapter 5 were produced on the development workstation described above.

3. **Storage Infrastructure** — Network-attached storage for benchmark datasets and experimental results.

3.5.3 Data Resources

1. NIST AM-Bench public dataset (freely available): the dataset actually used for the compression experiments, the machine learning evaluation, and testing the implemented data management workflow – this is the single dataset behind every result reported in Chapter 5.

2. Kaggle and UCI ML repository datasets (open access): reviewed during the dataset-selection process described in Section 3.3, but not used in the final quantitative evaluation.

3.6 Sources of Error and Limitations

I want to be upfront about the main sources of error and the limitations I was working with, both for the compression experiments and for the system implementation itself.

3.6.1 Compression Evaluation Limitations (RQ1)

- **Model Architecture Dependence** — How much compression a model can tolerate depends on the model itself, and I only tested this against Random Forest and Linear Regression. Other architectures – Gradient Boosting, Support Vector Regression, or a neural network – might respond to the same compression and reduction methods differently. I mitigated this as best I could by reporting results separately for both models I did test rather than averaging them together, but I want to be clear that the findings in Chapter 5 are scoped to these two model types [7].

- **Domain-Specific Features** — AM process-specific features (like melt pool temperature signatures) may not respond to compression the same way generic time-series data would. I mitigated this as best I could by evaluating the methods directly on the NIST AM-Bench data rather than a generic benchmark.

- **Benchmark Dataset Representativeness** — The NIST AM-Bench dataset is a curated benchmark, not raw production data, so it may not fully capture the statistical properties or the messiness of sensor data collected on an actual factory floor.

3.6.2 Usability Evaluation Limitations (RQ2)

I evaluated the system using technical acceptance tests rather than a formal usability study with manufacturing engineers. Because of that, I'm not making strong claims about user acceptance or usability in an industrial setting anywhere in this thesis. A future study involving domain experts would be genuinely useful for understanding how easily this kind of system could be adopted into a real manufacturing workflow.

3.6.3 System Implementation Limitations

- **Scalability Upper Bounds** — I want to restate this clearly, since it's the most important limitation in the thesis: my experiments were run on the NIST AM-Bench dataset at the scale described in Chapter 5, and the planned test at approximately 50 GB was not completed within the project period. I have not tested this system at 50 GB, let alone at the much larger scales a production industrial deployment might eventually need, and different architectural bottlenecks would likely appear at those larger scales that I simply have no evidence about. Wherever this thesis discusses scalability, it should be read as describing what was designed for, not what was measured [6].

- **Infrastructure Assumptions** — The implementation assumes fairly modest, on-premise infrastructure. A different organisation's IT setup might well need adaptations I haven't accounted for here.

3.7 Experimental Design Summary

Table 3.1 pulls together the experimental phases, objectives, methods and acceptance criteria for the two research questions covered in this thesis.

Table 3.1: Summary of Experimental Phases and Acceptance Criteria

Phase	**RQ**	**Method**	**Acceptance Criterion**
Phase 1A	RQ1	ATP-3: Lossless compression impact on model performance (Gzip, Parquet/Snappy)	$\Delta P < 5\%$
Phase 1B	RQ1	ATP-4: Data reduction impact on model performance (downsampling, PCA)	$\Delta P < 5\%$
Phase 1C	RQ1	ATP-7: System scalability with increasing data volume	System remains functional; not evaluated at 50 GB (test not completed)
Phase 2A	RQ2	Backend architecture evaluation (ATP-1, ATP-6, ATP-9)	Query latency < 5 sec; preprocessing reduction $\geq 30\%$

4 | Data Management System Implementation

This chapter walks through how I designed and built the data management system for experimental additive manufacturing data. I developed it to handle the organisation, storage, validation and preparation of AM data for AI and machine learning use.

The implementation pulls together FAIR data management principles, standardised metadata, data validation, compression, versioning and a set of user-facing interfaces. I built it as a modular architecture on purpose, so that the different parts of the workflow could be developed, tested and evaluated somewhat independently of each other.

Everything described here was tested using the benchmark data and the technical acceptance tests I defined in Chapter 3.

4.1 System Architecture Overview

I designed the system as a layered architecture so that the main responsibilities of the workflow would stay separated from each other. The layers cover the user interface, the API and application services, and data processing and storage. Splitting things up this way made it much easier to work on one part of the system without breaking another, and it also leaves room for extending the system later. Figure 4.1 shows the four main layers.

As Figure 4.1 shows, the architecture is made up of:

- **Presentation Layer:** Web-based interfaces that give access to the main data management functions – the dashboard, data visualisation, ML-related functions and admin controls. I designed this layer specifically so that common tasks wouldn't require anyone to touch the database or backend code directly [2].

- **API Layer:** FastAPI provides the REST endpoints that connect the frontend to the backend services.

- **Service Layer:** Organised around the three things the system actually needs to do – data management, machine learning, and collaboration-related functions.

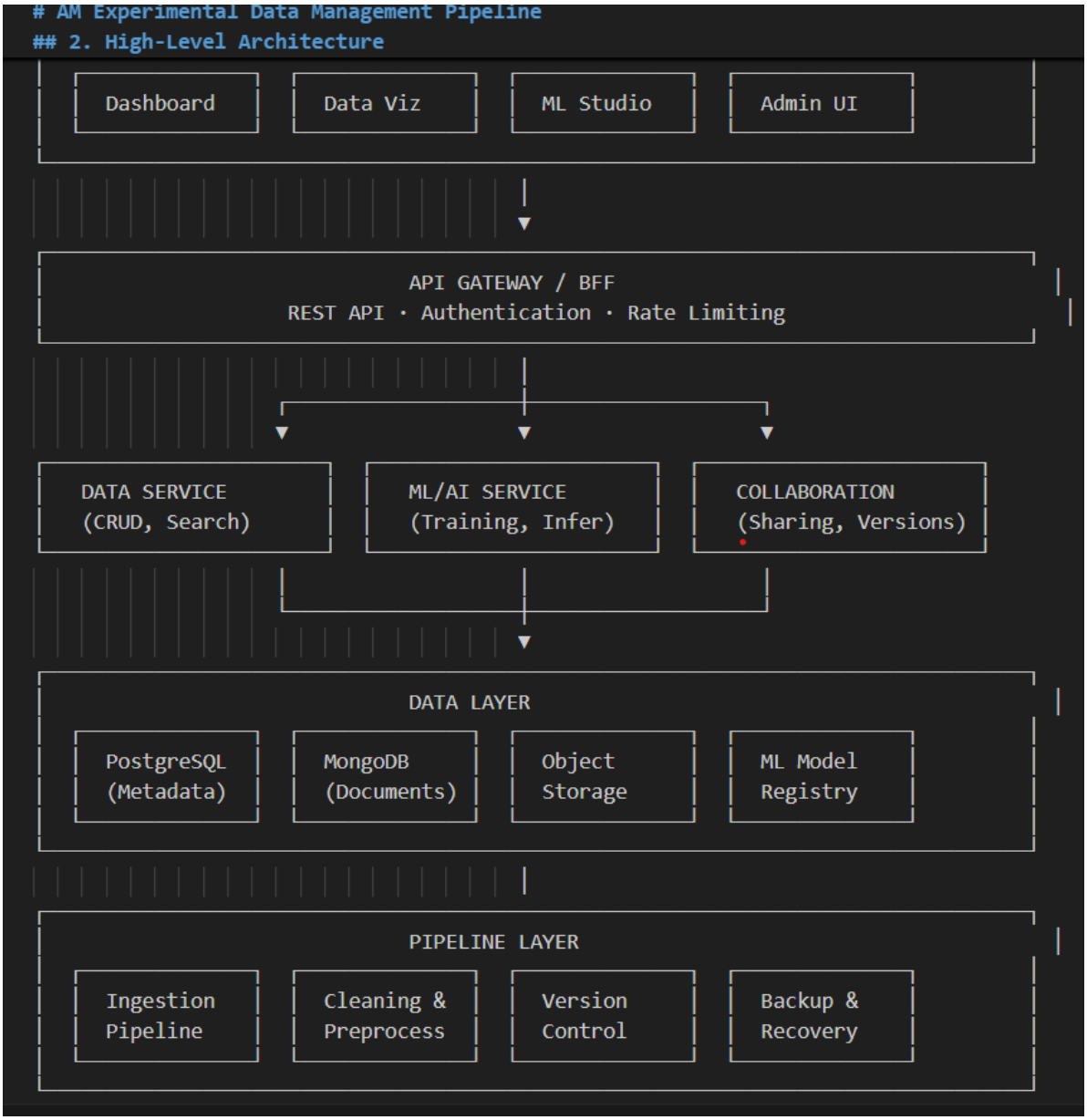

Figure 4.1: High-level architecture of the AM experimental data management pipeline. User-facing tools (Dashboard, Data Visualization, ML Studio, Admin UI) connect to backend services through a REST API gateway. The service layer manages data operations, ML/AI workflows, and collaboration features, while the data layer handles structured metadata, the stored experimental records, and references to the raw uploaded files.

4.2 Data Model and Schema Design

To keep AM experimental data FAIR [4] and usable for machine learning, I needed a well-defined data model behind everything else. I based the structure on established research data management practice [10] and adapted it to the specific needs of additive manufacturing process monitoring [3]. Figure 4.2 shows how the main entities relate to each other.

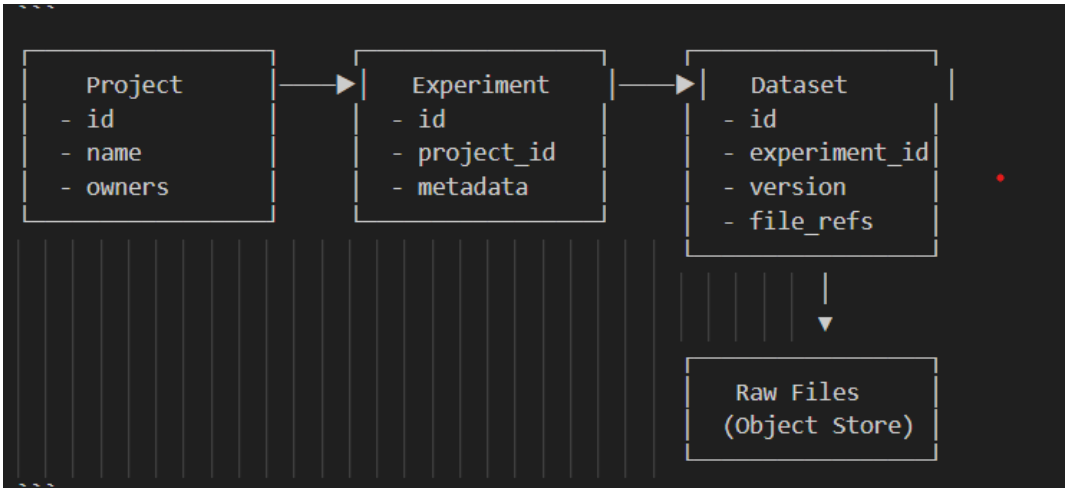

Figure 4.2: Entity-relationship diagram of the core data model. Projects contain multiple Experiments, each producing one or more Datasets with associated metadata and file references stored on the file system. This hierarchy supports organization at the project level while maintaining detailed traceability for individual experiments.

4.2.1 Core Entities

I organised the data model around three main entities, arranged in a clear hierarchy.

Project Entity Represents a research initiative. Each project includes:

- `id`: A unique identifier (UUID) that serves as a persistent reference, following FAIR principles [4].
- `name`: A human-readable project name.
- `owners`: The individuals or teams responsible for the project.
- `created_date`: The project start date and time (with timezone).
- `description`: Free-text description of the project's objectives.
- `metadata`: Additional contextual information (funding, collaborators, goals).

Experiment Entity Captures individual experimental runs or build jobs, recording the specific conditions and parameters for each AM build in line with AM metrology standards [3, 12]:

- `id`: Unique identifier (UUID).
- `project_id`: Links the experiment to its parent project.
- `experiment_name`: A descriptive name for the specific experiment.
- `build_date`: When the experiment took place.
- `material_type`: The material used (for example, TiAl4V, IN625, PETG).
- `build_platform`: The AM machine used for the build.
- `operator`: The person responsible for the run.
- `status`: Current state (planned, in-progress, completed, failed).
- `process_parameters`: Build settings (detailed in Section 4.2.2).
- `quality_metrics`: Measured outcomes (detailed in Section 4.2.3).

Dataset Entity Represents the actual data files produced by an experiment. A single experiment can easily produce several datasets – thermal camera recordings, pressure sensor logs, post-build inspection images, and so on:

- `id`: Unique identifier (UUID).
- `experiment_id`: Links the dataset to its parent experiment.
- `version`: Version number (major.minor.patch format).
- `file_refs`: Pointers to the location of the raw data files on the file system.
- `format`: File type (CSV, HDF5, Parquet, binary).
- `compression`: Compression method used (PCA) [7].
- `size_bytes`: Storage size of the dataset.
- `checksum`: A hash used to verify data integrity.
- `lineage`: Full history of transformations applied to the data [6].

As Figure 4.2 shows, I kept the raw uploaded files on the file system and stored only references to them in the database. This turned out to be a sensible way of handling large files without losing the relationships between the metadata entities, and it avoids putting large binary blobs inside the database itself.

4.2.2 Process Parameters Schema

The `process_parameters` field captures AM build settings in JSON format, which gave me structure while still leaving room for domain-specific parameters. I aligned the schema with STEP standards [13]. Table 4.1 lists the standard parameters I track.

Table 4.1: Standardized process parameters schema for AM experimental data

Parameter	**Type**	**Required**	**Description**
layer_height	float	Yes	Layer thickness (millimeters)
nozzle_temperature	float	Yes	Extruder temperature (Celsius)
bed_temperature	float	Yes	Build platform temperature (Celsius)
print_speed	float	Yes	Print head speed (mm/s)
infill_percentage	float	Yes	Internal density (%)
infill_pattern	string	No	Internal geometry pattern (grid, honeycomb, etc.)
shell_count	integer	No	Number of outer perimeter layers
support_enabled	boolean	No	Whether support structures were used

This schema follows ASTM standards for documenting AM processes [3], but I extended it with a few parameters that matter specifically for training machine learning models. Using JSON also means I can add new fields later without needing a full database redesign.

4.2.3 Quality Metrics Schema

The `quality_metrics` field records the measured outcome of each build, which is what makes it possible to look at how process settings relate to final part quality. Following measurement-science principles for additive manufacturing [3, 12], Table 4.2 shows the standard quality metrics I capture.

Table 4.2: Standardized quality metrics schema for AM experimental outcomes

Metric	**Type**	**Unit**	**Description**
tensile_strength	float	MPa	Ultimate tensile strength
surface_roughness	float	μm	Average surface roughness (Ra)
porosity	float	%	Volumetric porosity percentage
dimensional_accuracy	float	mm	Deviation from CAD design
hardness	float	HV	Vickers hardness measurement

I store quality metrics as structured JSON, including measurement uncertainty information, following NIST guidelines [12]. That way, a model trained on this data can actually account for how much variability there is in the underlying measurements, rather than treating every value as exact.

4.3 Backend Implementation

The backend implements the service and data layers shown in Figure 4.1. I built it in Python 3.10+, using libraries that are well established for data-intensive applications [17].

Technology Stack

Table 4.3 summarises the main technologies I used.

Table 4.3: Technology stack used in the implementation

Component	**Technology**	**Purpose**
Backend_API	FastAPI + Uvicorn	Provides the versioned REST API (`/api/v1/`) and serves the dashboards
Programming	Python 3.10+	Main programming language used for the implementation
ORM	SQLAlchemy	Maps the Python data model onto the database schema
Metadata_Storage	SQLite (development), PostgreSQL (production)	SQLite for the zero-configuration development setup; PostgreSQL for the production configuration
Data_Processing	Pandas, NumPy	Used for data loading, processing and preparation
Compression	gzip, Parquet/Snappy, scikit-learn PCA	Used for lossless compression and dimensionality reduction experiments
Machine_Learning	Scikit-learn	Used for the Random Forest and Linear Regression experiments
Version_Control	Custom SHA-256 module; Git	In-system record versioning (`version_control.py`); Git for source-code version control
Security	PyJWT, RBAC	Token-based authentication and three-role access control
Containerization	Docker, Docker Compose	Used to create a consistent development environment
Frontend	HTML, CSS, JavaScript with Chart.js and Plotly.js	Browser-based dashboards for data management, visualisation and upload
Deployment	Render	Managed platform hosting the deployed application

4.3.1 Data Ingestion Module

The data ingestion module is what brings experimental data into the system. Figure 4.3 shows how I organised the backend code.

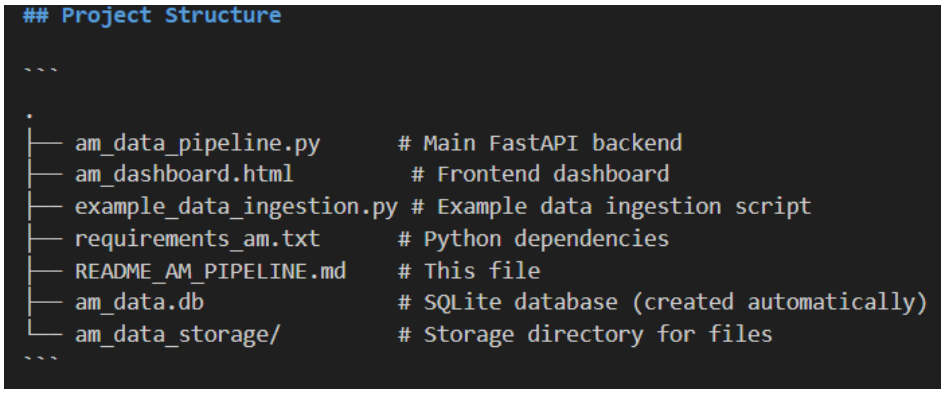

Figure 4.3: Backend project structure. Modules handle data ingestion (`am_data_pipeline.py`), dashboard (`am_dashboard.html`), example scripts (`example_data_ingestion.py`), dependencies (`requirements_am.txt`), and documentation (`README_AM_PIPELINE.md`). The SQLite database (`am_data.db`) is created automatically for metadata storage.

The ingestion pipeline follows these steps, based on established data quality practice [11, 18]:

1. **Format Detection:** Identifies file types by examining extensions and reading file headers.
2. **Schema Validation:** Checks that data conforms to expected structures using JSON Schema for metadata and statistical checks for numerical data [11].
3. **Metadata Extraction:** Pulls metadata from headers and filenames, merging it with user-provided information following Dublin Core standards [19].
4. **Compression:** Applies the appropriate compression method (Chapter 5) [7].
5. **Checksumming:** Computes SHA-256 hashes for original and compressed files to ensure integrity.
6. **Storage:** Writes the record and its metadata to the database – SQLite in the development configuration, PostgreSQL via SQLAlchemy in the production configuration – with the raw uploaded file retained on the file system and referenced from the record.
7. **Indexing:** Creates searchable indexes on frequently queried fields (date, material type, operator).
8. **Versioning:** Registers the record in the version-control module (`version_control.py`), which creates a SHA-256 version snapshot and records the change in the audit trail [6].

4.3.2 Metadata Management System

I built the metadata management module to store the information that describes a dataset and the experimental context it came from – things like experiment identification, process parameters, data source, file format, and anything else needed to actually understand and retrieve the dataset later.

Metadata validation runs during ingestion, so the required fields are checked before a dataset is used for anything downstream.

4.3.3 Data Versioning and Lineage Tracking

Rather than relying on an external tool such as DVC, I implemented version control directly inside the system (`version_control.py`), so that versioning is tied to the same database records the rest of the workflow uses [6, 2]. Every write creates a new version snapshot with the following properties:

- **Version snapshots:** Each version stores a full JSON snapshot of the experiment, its process parameters, geometry data and quality metrics, together with a SHA-256 `version_hash` of the record contents.
- **Hash-based deduplication:** Because the hash is computed from the record contents, an update that does not actually change anything does not create a redundant version.
- **Immutable history:** Versions are never modified once created. Each version records its creator, timestamp, change type (create, update, delete, restore) and the specific fields that changed.
- **Comparison and rollback:** Any two versions can be compared to produce a field-level diff, and a record can be restored to any earlier version – with the restore itself recorded as a new version, so the audit trail stays complete.
- **Tagging:** Versions can be tagged (for example, as reviewed or validated) so that important states can be found again later.

The audit trail records every operation applied to a record, which is what makes it possible to trace a dataset from its raw uploaded state through to the version used for model training (ATP-5 and ATP-8).

4.3.4 Query and Retrieval Engine

The query and retrieval engine is what gives machine learning workflows efficient access to the stored data.

It supports SQL-like filtering by numerical ranges, categories, date ranges with timezones, logical conditions (AND/OR/NOT), and nested queries on JSON metadata. Results can be exported as CSV (via Pandas [17]), Parquet or JSON, which cuts down on the preprocessing work needed afterwards.

4.4 Frontend Implementation

The frontend provides a graphical interface for interacting with the data management system.

4.4.1 Technology Stack

I deliberately kept the frontend simple. Rather than introducing a JavaScript framework and a build step, I built the interfaces as plain HTML pages served directly by FastAPI, with charting handled by two established libraries:

- **Markup and styling:** HTML and CSS, served as static pages by the FastAPI backend.
- **Interactivity:** Vanilla JavaScript, calling the REST API directly via `fetch`.
- **Charts:** Chart.js for the standard charts (material distribution, quality metrics, model comparison) and Plotly.js for the interactive correlation heatmap and scatter views.

The main reason for this choice was that it keeps the whole system runnable with a single Python command and no separate frontend build, which matters for reproducibility – anyone wanting to verify this work can start the server and open a browser without installing a JavaScript toolchain.

The interfaces are served at the following routes:

- `/` — the main application and ML pipeline entry point.
- `/dashboard/advanced` — the advanced data visualisation dashboard.
- `/upload` — the drag-and-drop data upload interface.
- `/api/v1/docs` — the interactive OpenAPI (Swagger) documentation for the REST API.

4.4.2 Dashboard Interface

The dashboard (Figure 4.4) is the entry point into the application, giving an overview of the available data and quick access to recent datasets.

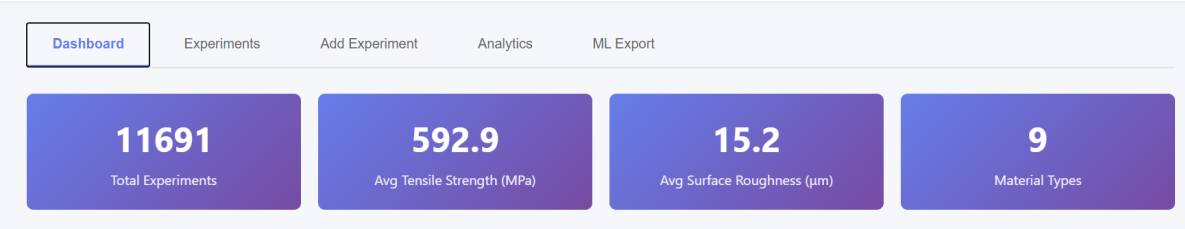

Figure 4.4: Dashboard interface showing an overview of the available experimental data. [18].

It shows key performance indicators, a material distribution donut chart (Figure 4.5), quality metrics trends, recent activity, and a few quick actions.

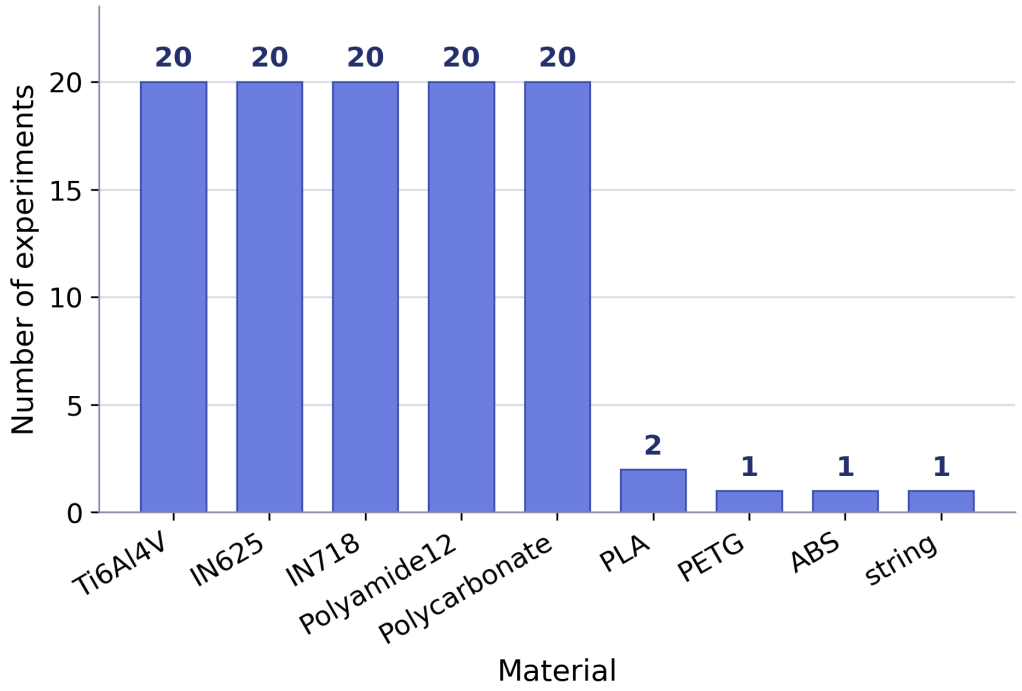

Figure 4.5: Material distribution across the 105 material-classified records held by the system. Ti6Al4V, IN625, IN718, Polyamide12 and Polycarbonate each have 20 experiments, PLA has 2, and PETG, ABS and the placeholder label `string` have 1 each. The imbalance between these groups is what limits the per-material machine learning evaluation reported in Chapter 5 [1].

4.4.3 Experiments Management Interface

The experiments interface (Figure 4.6) is where users browse, filter and manage datasets.

It includes a filterable grid view, advanced filtering, full-text search, sorting, and batch operations.

4.4.4 Add Experiment Interface

The add experiment interface (Figure 4.7) is a structured form with real-time validation, built to enforce FAIR compliance [4].

The form is split into sections covering basic information (following Dublin Core [19]), process parameters with validation (based on manufacturing best practice [3]), and file upload with drag-and-drop, format detection, and a 10 GB size limit. Users can't submit the form until every required field is valid, which is how I enforce $M \geq 0.9$ at the point of entry rather than trying to fix things afterwards.

4.4.5 Analytics and Visualization Interface

The analytics interface (Figure 4.8) lets someone explore a dataset without writing any code [18].

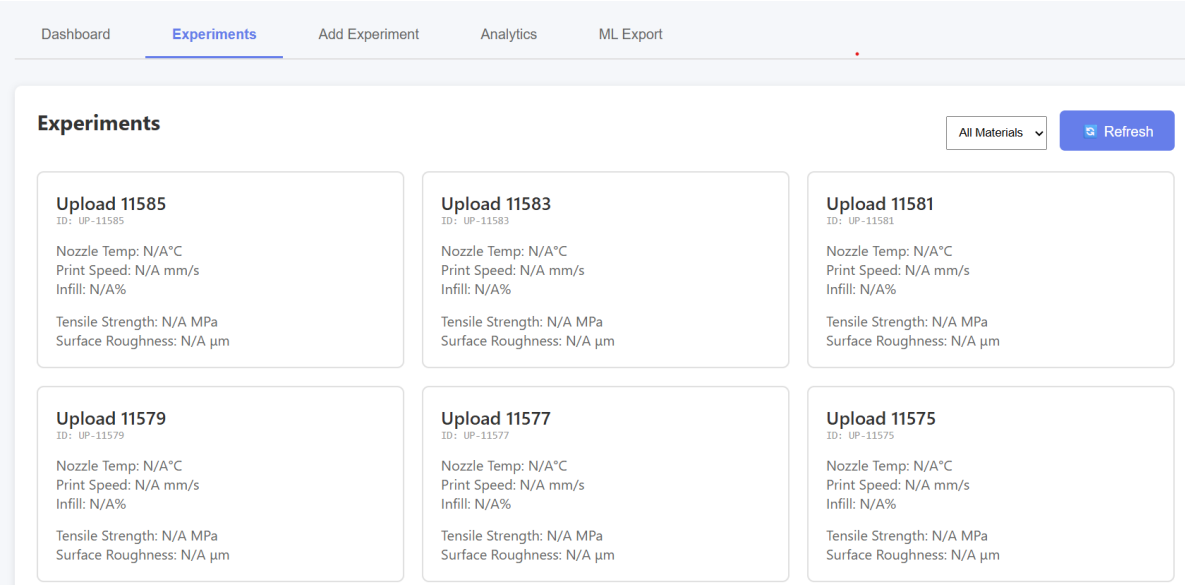

Figure 4.6: Experiments management interface with a paginated list, material filtering, and quick refresh. Each card shows upload ID, nozzle temperature, print speed, infill percentage, and a preview of quality metrics [3].

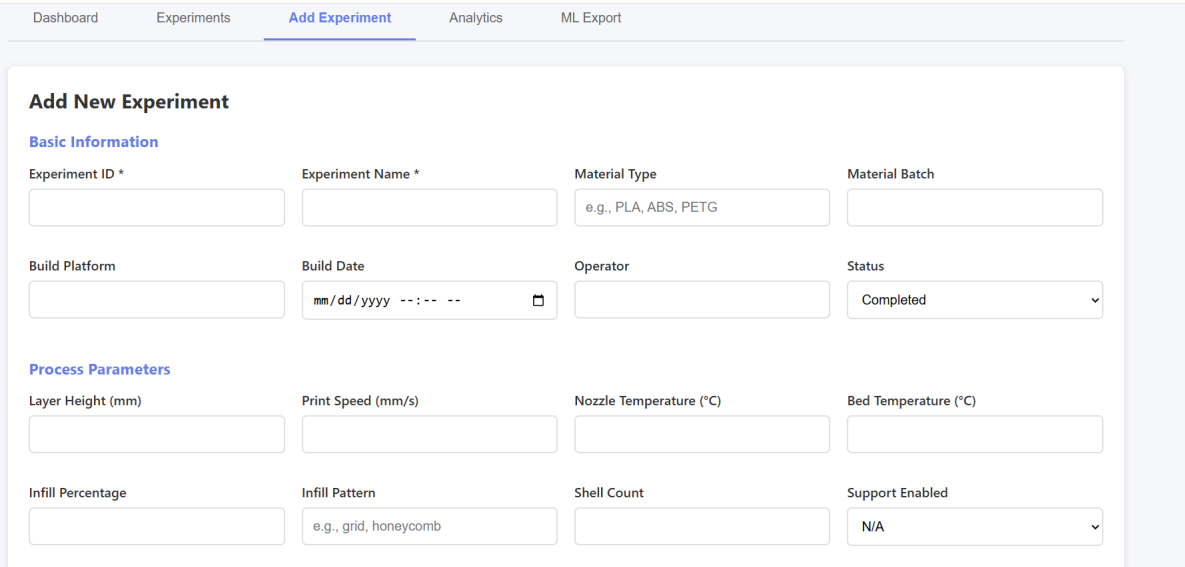

Figure 4.7: Add Experiment form with sections for basic information, process parameters, and file upload. The form validates fields in real time to enforce metadata completeness standards [10].

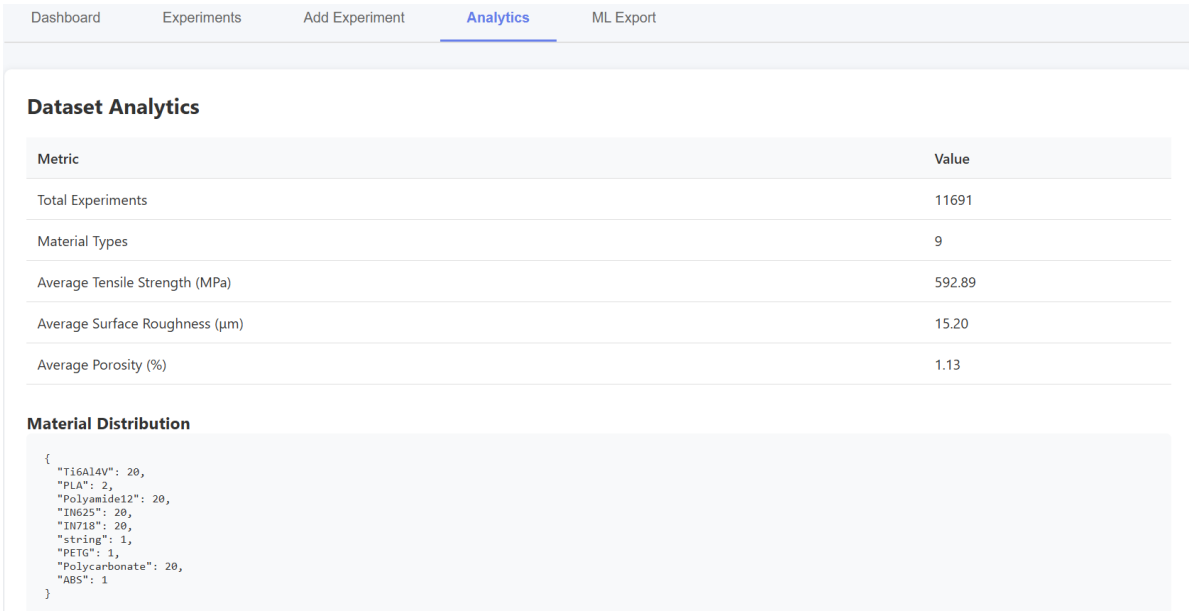

Figure 4.8: Analytics interface showing a summary table (11,691 experiments, 9 materials, average tensile strength 592.89 MPa, average surface roughness 15.20 μm, average porosity 1.13%) and underlying JSON for programmatic access.

It includes a dataset metrics table [3, 12], a material distribution bar chart (Figure 4.9), a quality metrics comparison (Figure 4.10), a process parameters distribution (Figure 4.11), a parameter correlation heatmap (Figure 4.12), and a correlation matrix (Figure 4.13). All of these visualisations are interactive, exportable, and responsive across screen sizes.

I want to flag one thing explicitly: the record and material counts visible in Figure 4.8 (11,691 experiments, 9 materials) were captured from an earlier development load of the system, used here purely to illustrate what the interface looks like. They are not the same dataset as the one used for the final quantitative evaluation in Chapter 5, which was run against a fixed load of 12,263 records across 21 fields. I mention this so the two numbers aren't mistaken for describing the same dataset.

4.4.6 ML Export Interface

The ML export interface (Figure 4.14) is what streamlines getting a dataset ready for machine learning [6].

A user can filter by material, pick an export format (CSV, Parquet, JSON), and configure preprocessing options such as scaling, one-hot encoding, train/validation/test splits, and imputation [11]. Every export bundles the data files together with a README, a metadata.json file (FAIR-compliant provenance [4]), a requirements.txt [6], and an example_load.py script [17]. I log all exports so they stay reproducible and traceable [20].

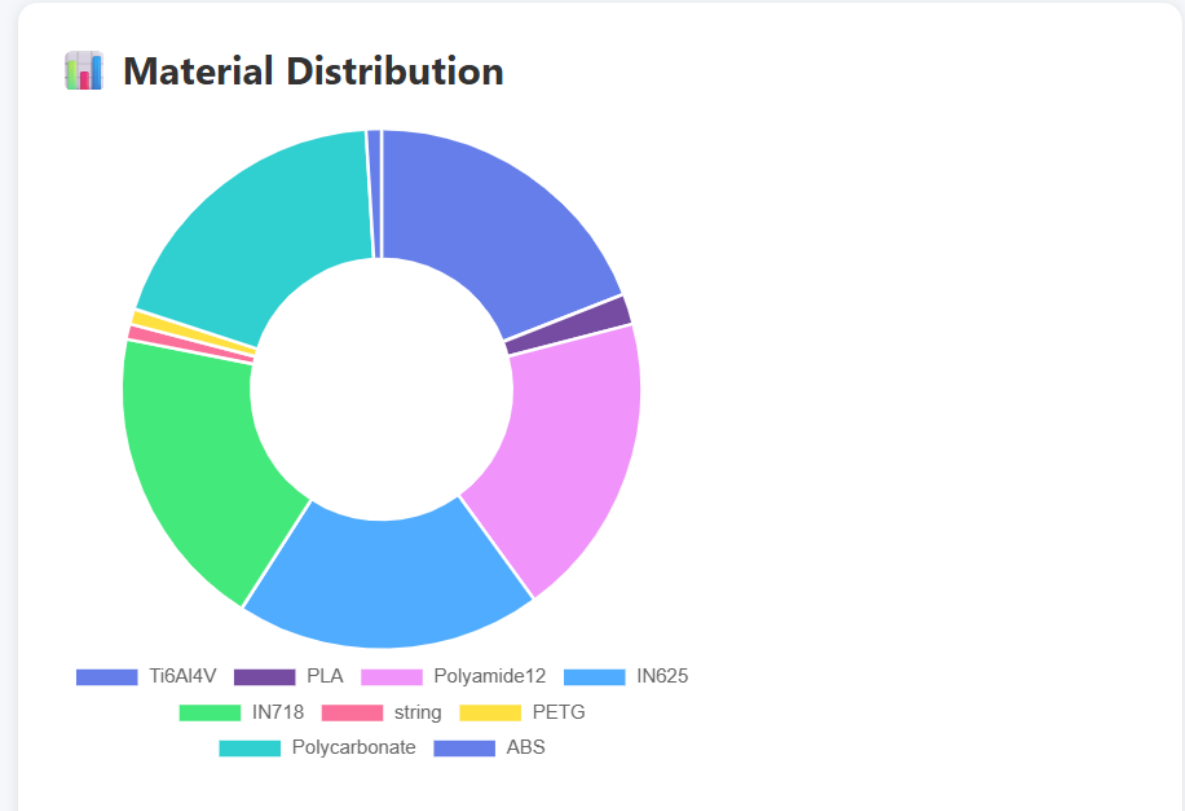

Figure 4.9: Bar chart of experiment counts by material.

4.5 Data Compression and Storage Optimization

Compression and storage optimisation mattered a lot for this workflow, since additive manufacturing experiments can generate a lot of data, and storing the same information in different formats can produce very different file sizes and processing costs.

4.5.1 Compression and Data Reduction Terminology

Before describing the formats, I want to be precise about the four techniques from Chapter 3 and how each one is actually implemented, since these get conflated a lot in casual usage:

- **Gzip** is lossless compression – I applied it directly to the raw CSV export using Python's standard `gzip` module. No information is discarded; the original file can always be reconstructed exactly.

- **Parquet with Snappy compression** combines a columnar storage format (Parquet) with a fast, lossless compression codec (Snappy). Storing data column-by-column rather than row-by-row already reduces size for structured data with repeated values, and Snappy compresses that further – again, losslessly.

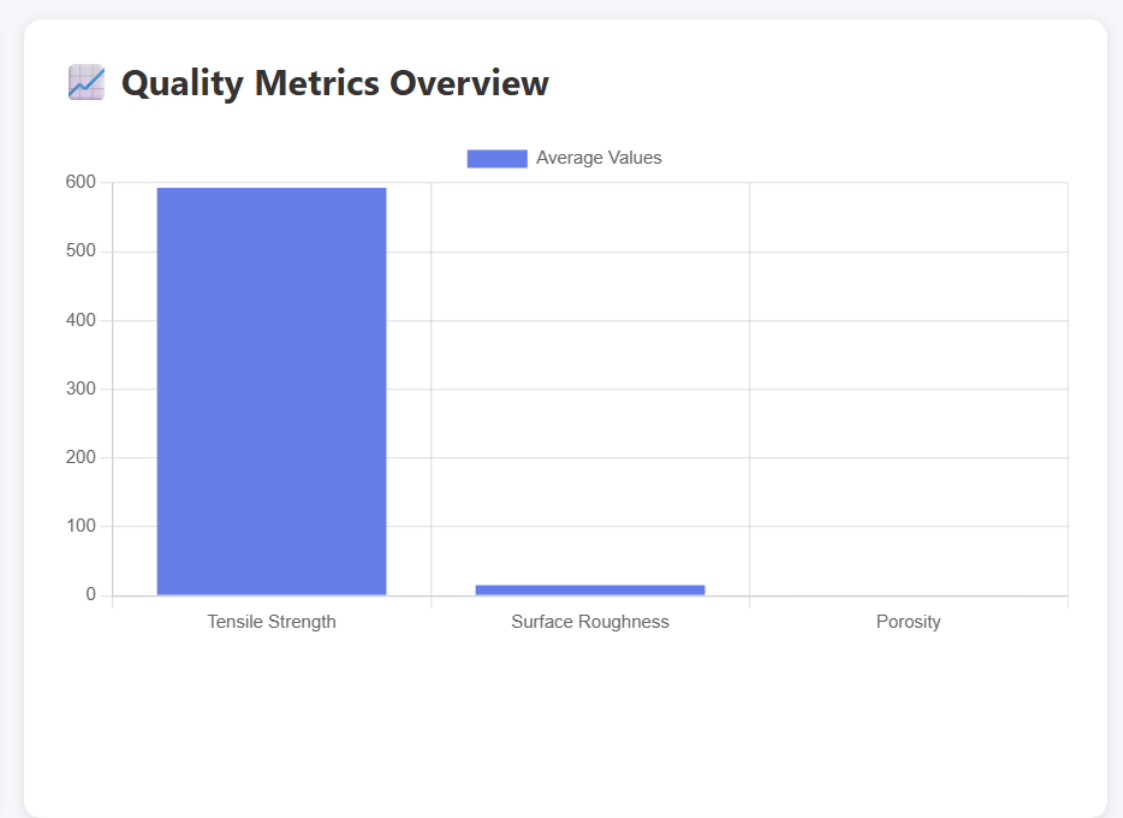

Figure 4.10: Bar chart comparing average tensile strength, surface roughness, and porosity.

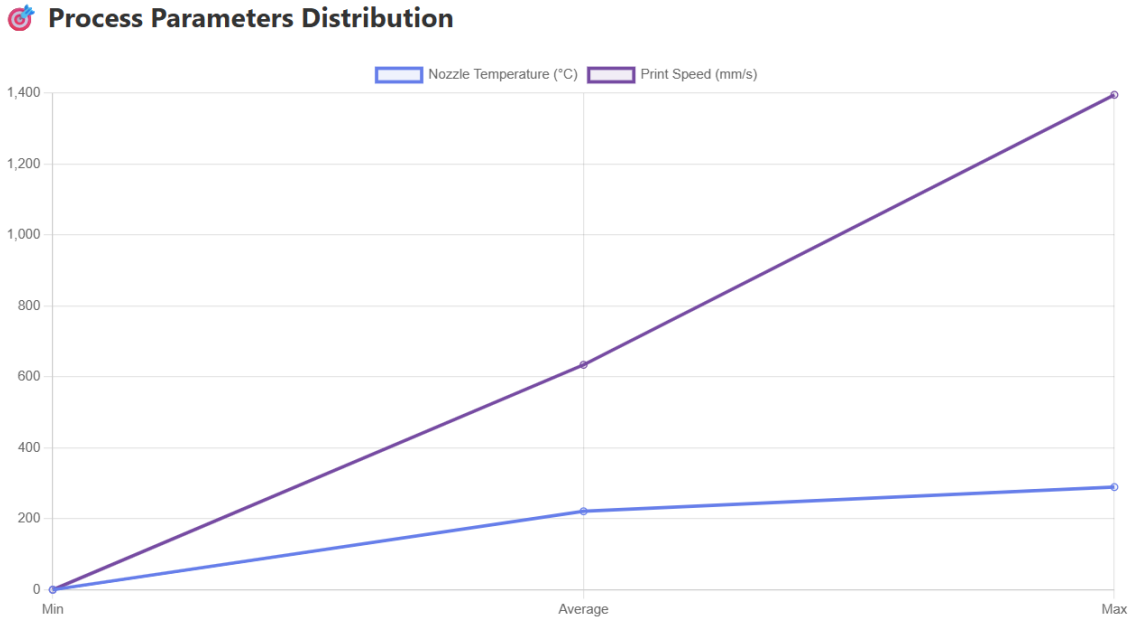

Figure 4.11: Line chart showing min, average, and max values for nozzle temperature and print speed.

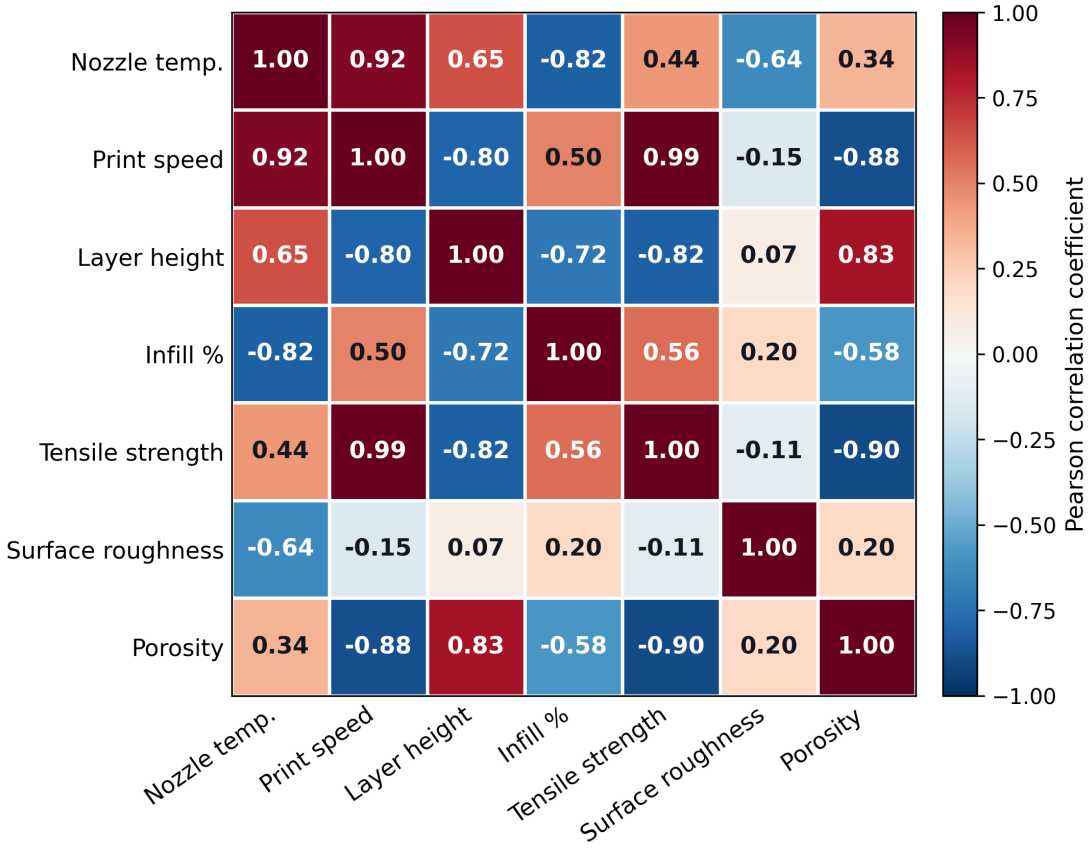

Figure 4.12: Correlation heatmap between process parameters and quality metrics, computed as Pearson coefficients over the material-classified records ($n = 100$; pairwise-complete). Nozzle temperature is recorded only for the two polymer processes (FDM and SLS), so correlations in that row and column rest on $n = 40$ [8].

- **Downsampling** is a data-reduction technique, not compression – it reduces the number of observations by keeping only every nth record (I tested 1:5 and 1:10 ratios). This is lossy in the sense that discarded records cannot be recovered.
- **PCA (Principal Component Analysis)** is a dimensionality-reduction technique, implemented using scikit-learn – it reduces the number of features a record is represented by, rather than reducing file size directly. I configured it to retain either 95% or 99% of the variance in the original data.

I implemented all four inside the same data-processing pipeline described in Section 4.4, so that a dataset could be exported in its raw form or with any of these four techniques applied, and the resulting file size and downstream machine learning performance could be compared on the same underlying data. The actual measurements from that comparison are reported in Chapter 5.

I looked at four commonly used data formats for storing and processing the experimental data: CSV, Excel, JSON and Parquet. I picked these because they represent structured data in fairly different ways and are all widely used in data processing and analysis.

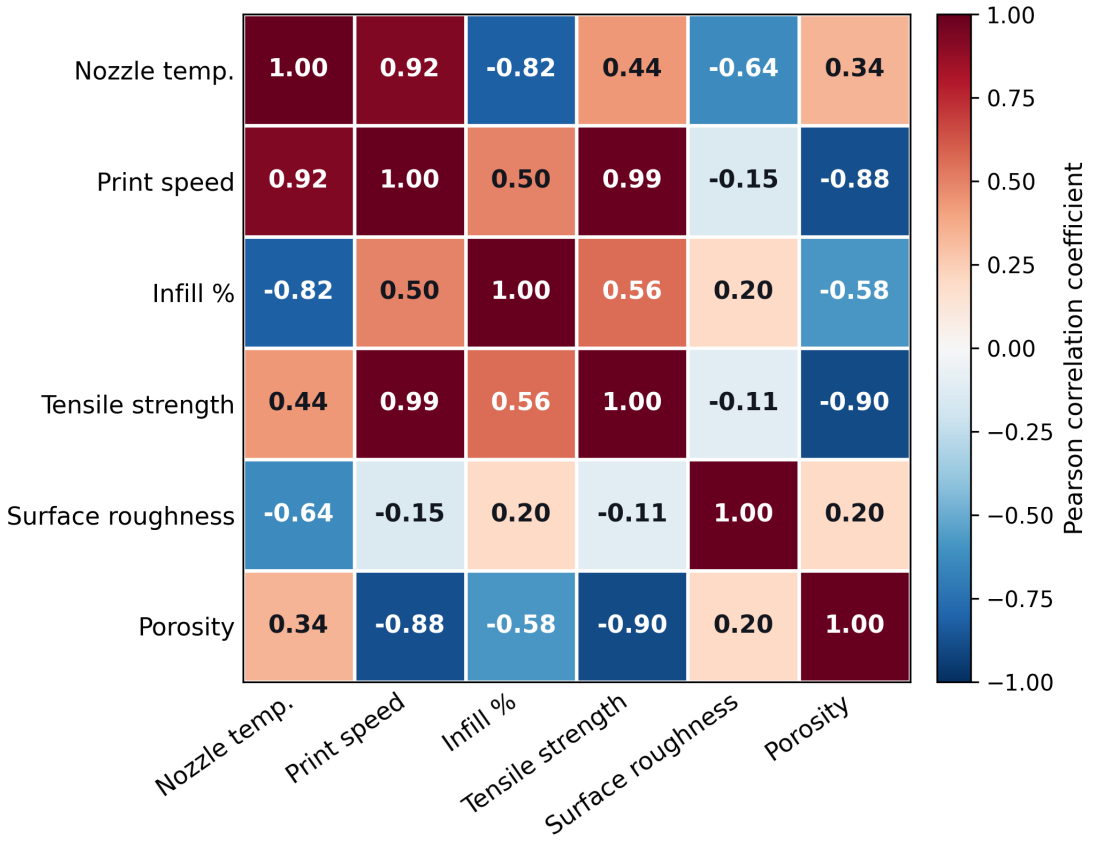

Figure 4.13: Correlation matrix over six process and quality variables, with the Pearson coefficient printed in each cell. Values are pooled across all five material and process groups, so they reflect differences between material groups as well as relationships within them; this is discussed further in Section 5.6.1.

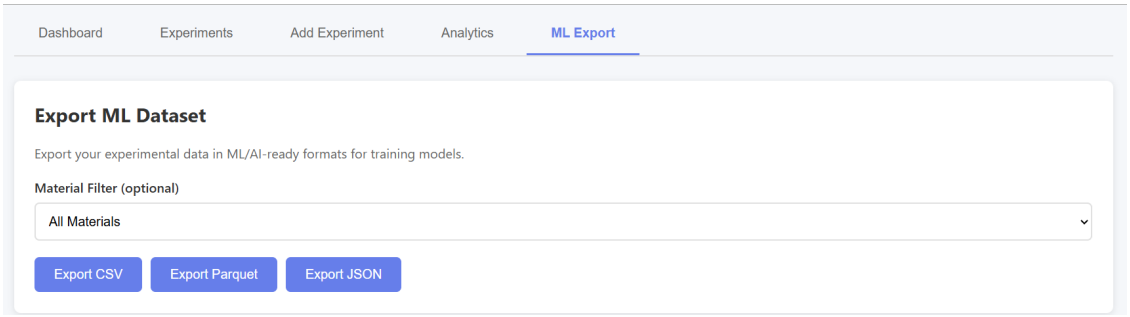

Figure 4.14: ML Export interface for generating ML/AI-ready datasets. Users can filter by material and choose CSV [17], Parquet, or JSON. This is the interface behind the ATP-9 preprocessing-time measurement reported in Chapter 5.

The comparison focused on file size and how well each format worked for further processing and machine learning. I converted the original data into each of the selected formats and compared the resulting file sizes directly, so the storage efficiency of each format could be evaluated on the same underlying dataset.

4.5.2 Data Formats Used for Storage

I used four data formats throughout the implementation and evaluation – CSV, Excel, JSON and Parquet – each of which stores structured data a bit differently, which in turn affects file size and how long it takes to read or process the data.

- **CSV:** I used CSV (Comma-Separated Values) as a simple tabular format for storing structured experimental data. It's easy to read and works with pretty much every data-processing tool out there.
- **Excel:** Useful when the data needs to be inspected or edited manually, since spreadsheet software is something most people are already comfortable with.
- **JSON:** I used JSON (JavaScript Object Notation) to represent structured data together with its associated metadata in a flexible format, particularly useful when the data has different fields or nested structures.
- **Parquet:** A column-oriented format I used for efficient storage and processing, generally giving smaller file sizes than the more traditional tabular formats.

Where it made sense, I stored the same dataset across all four formats so I could compare storage requirements under the same conditions.

4.5.3 Storage Size Evaluation

To evaluate storage efficiency, I converted the experimental dataset into CSV, Excel, JSON and Parquet and compared the resulting file sizes.

The original dataset served as the reference point. After converting to each format, I recorded the size of the resulting file and calculated the storage reduction relative to the reference.

The storage reduction was calculated using:

$$\text{Storage Reduction (\%)} = \left(1 - \frac{S_{\text{converted}}}{S_{\text{reference}}}\right) \times 100 \tag{4.1}$$

where $S_{\text{reference}}$ is the size of the reference dataset and $S_{\text{converted}}$ is the size of the dataset after conversion to the format being tested.

I ran this comparison on the same dataset and, as far as possible, the same set of records, to keep the comparison between formats consistent.

4.5.4 Storage and Data Management Integration

I incorporated each format into the workflow according to what it's actually good for. CSV and Excel are convenient for viewing and handling tabular data by hand, JSON gives a flexible representation for structured information and metadata, and Parquet is the more storage-efficient option for structured datasets that need further processing.

The system lets data be converted between the supported formats as part of the workflow, and I kept the metadata associated with a dataset stored separately, so information about a dataset doesn't depend entirely on parsing the file contents.

Having multiple formats available also adds some flexibility to how the system gets used day to day – CSV or Excel when someone wants to eyeball a dataset, Parquet when storage efficiency and programmatic processing matter more.

4.6 Integration and Validation

Once the individual components were working, I integrated them to evaluate the complete data management workflow end to end – ingestion, metadata validation, storage, retrieval, processing, and preparation for machine learning.

I defined a set of Acceptance Test Procedures (ATPs) to check the main functional and technical requirements of the system, covering ingestion, metadata management, compression, retrieval, scalability, data integrity, preprocessing and interoperability. I wrote these criteria down before looking at the final results, specifically so I wouldn't be tempted to adjust the bar after the fact. The actual measurements and pass/fail outcomes are presented in Chapter 5.

4.7 Deployment and Operational Considerations

4.7.1 Deployment Architecture

I want to be precise about two things that are easy to conflate here: the environment I used during development, and the platform I actually deployed the finished application to.

During development, I used Docker Compose to run the backend and its dependencies as isolated, containerised services, mainly so I would have an identical environment across my own development and testing, and so the setup could be moved between machines without reconfiguring everything by hand [5].

For the actual deployment that I evaluated the system on – and the one referenced throughout Chapter 5 – I deployed the application on Render, a managed hosting platform, so it could be accessed through a normal web browser rather than only running locally. Render is the deployment platform; Docker Compose was a development and environment-consistency tool, not the hosting platform itself.

Whether running locally or deployed, the application exposes the same set of routes, which are the entry points I used for the evaluation in Chapter 5:

Table 4.4: Acceptance Test Procedures for the implemented system

ATP	**Test**	**Evaluation Criterion**
ATP-1	Data ingestion	Data is successfully ingested into the data management system.
ATP-2	Metadata validation	Required metadata fields are validated and stored correctly.
ATP-3	Compression impact	The change in machine learning performance remains within the defined acceptance threshold.
ATP-4	Feature reduction	Machine learning performance remains within the defined acceptance threshold after data reduction.
ATP-5	Data retrieval	Stored data and associated metadata can be retrieved correctly.
ATP-6	Query performance	Data retrieval and query operations meet the defined performance requirement.
ATP-7	Scalability	The system remains functional when the amount of processed data is increased.
ATP-8	Data integrity	Retrieved data remains consistent with the stored data.
ATP-9	Preprocessing	The dataset can be prepared successfully for machine learning processing.
ATP-10	Interoperability	Data can be converted between the supported CSV, Excel, JSON and Parquet formats while maintaining the required data fields.

Table 4.5: Application routes exposed by the implemented system

Route	**Interface**
`/`	Main application and AI/ML pipeline entry point
`/dashboard/advanced`	Advanced data visualisation dashboard
`/upload`	Data upload interface
`/api/v1/docs`	Interactive OpenAPI (Swagger) API documentation

4.7.2 Deployment and Maintenance Considerations

In a larger deployment, this system could be extended with monitoring, automated backups and regular maintenance procedures. I thought about these as part of the deployment design, but building them out was outside the experimental scope of this thesis.

4.7.3 Security and Access Control

I implemented Role-Based Access Control (RBAC) to control access to the different functions of the system, so that users only get access to what's relevant to their assigned role.

For this project, security is limited to RBAC. I did not implement encryption, more advanced authentication, audit logging, or enterprise-level access management – those are things I'd consider for future development rather than something this thesis covers.

Table 4.6: User roles and access permissions

Role	**Access permissions**
Administrator	Full access to system functions, including user and data management.
Researcher	Access to data management, dataset viewing and analysis functions.
Viewer	Read-only access to available datasets and information.

4.8 Discussion

Between them, the two research questions pulled the implementation in slightly different directions. RQ1 is addressed through the compression and data-reduction experiments and how they affected machine learning performance. RQ2 is addressed by the design and implementation of the data management architecture itself – ingestion, metadata management, storage, retrieval and the user-facing functions.

The technical evaluation shows that this workflow can be implemented as a working system. That said, the evaluation is based on benchmark data and technical tests rather than a formal industrial user study, so I want to be clear that any conclusions about real-world adoption or usability belong in future work, not in what I'm claiming here.

4.8.1 Contributions Beyond Requirements

I ended up implementing automated quality assessment and interactive analytics as additional functionality, beyond what the original requirements strictly called for, because the system supported them naturally. Other capabilities like AutoML, streaming, or federated learning are things I see as future extensions rather than part of what's built here.

4.8.2 Limitations and Future Work

Limitations

The main limitation of the current implementation is that I haven't completed large-scale scalability testing. I designed the system with larger datasets in mind, but testing with a 50 GB dataset didn't happen during this project, mostly because of the time and computational resources available to me.

Because of that, I don't think the current results should be used to draw conclusions about how the system would behave with much larger, more industrial-scale datasets. Testing with something closer to 50 GB or beyond would be a useful next step for understanding performance, storage requirements and processing time at higher data volumes.

I'm treating that large-scale test as future work. It would also be a good way to spot performance bottlenecks and figure out whether the system would need additional optimisation, or a move to distributed storage and processing, before it could realistically be used at industrial scale.

4.9 Summary

This chapter walked through the implementation of the data management system for experimental additive manufacturing data – ingestion, metadata management, validation, storage, compression, versioning and the user-facing functions, all tied into one workflow.

I built the backend and frontend as modular components, so the main data management operations could be accessed through the application without one part depending too tightly on another. The compression methods were integrated into the same workflow, which is what made it possible to evaluate their effect on storage and machine learning performance later on.

I evaluated the implemented system technically, using the benchmark data and acceptance tests described in Chapter 3. The actual numbers from those tests are presented and discussed in Chapter 5.

5 | Results and Discussion

In this chapter I present the results I obtained while implementing and testing the data management system described in the previous chapters. I have organised the chapter around the evaluation activities from Chapter 3 and the implementation from Chapter 4, so that each result can be traced back to the requirement or test procedure it belongs to.

The evaluation covers data ingestion, data quality, compression and data reduction, machine learning, data visualisation, the API services, version control, role-based access control and deployment. For all of this I used the NIST Additive Manufacturing Bench (AM-Bench) dataset, since it is publicly available and gave me a reproducible basis to test the system on.

I want to be upfront about two limitations before going into the results. First, I did not have access to an industrial partner dataset, and I did not run a user study with real manufacturing engineers. Everything reported below comes from the NIST dataset and from testing the software components myself. Second, I had originally planned to test the system with a 50 GB dataset to see how it behaves at larger scale, but I did not manage to complete that test within the time I had. I come back to both points in Section 5.16, and I did not want to make any claims that go beyond what I actually tested.

5.1 Experimental Dataset and Evaluation Setup

To evaluate the system I used publicly available additive manufacturing benchmark data, specifically the NIST AM-Bench dataset, as my main reference dataset.

Before running anything, I spent some time just getting familiar with the dataset – looking at its structure, the variables it contains and what metadata was available. Once I had a good sense of that, I prepared the relevant fields so that they matched the structure my system expects. I then used this prepared data to test ingestion, validation, storage, retrieval, visualisation and the machine learning pipeline, one component at a time.

The evaluation itself followed the Acceptance Test Procedures (ATPs) I defined in Chapter 3. These gave me a checklist of sorts to work through, so I could see whether the main functions of the system behaved the way I had intended.

For storage, I looked at four formats that are used throughout the project – CSV, Excel, JSON and Parquet – and I also evaluated a few compression and data-reduction methods to understand how storage savings trade off against machine learning performance.

There is an important distinction in that table that I want to make explicit, because it affects how the rest of this chapter should be read. The database holds 12,263 records in total, but only 105 of them carry a material-type label. The remaining 12,158 are bulk

Table 5.1: Dataset and evaluation setup

Parameter	**Value**
Dataset source	NIST AM-Bench dataset
Total records in database	12,263
Material-classified records	105 (across 9 material types)
Unclassified upload records	12,158
Number of data fields	21
Data formats	CSV, Excel, JSON and Parquet
Main evaluation	Data management and AI/ML workflow
50 GB scalability test	Not completed

upload records that were ingested through the pipeline but had not been tagged with a material type.

These two groups were used for different things, and I have kept them separate throughout the evaluation:

- The **full 12,263 records** are what the data-management side of the system was exercised against – ingestion throughput, data-quality scoring, storage and compression comparisons, API query performance, and the interoperability tests. Those figures genuinely reflect the full dataset.
- The **105 material-classified records** are what the machine learning evaluation in Section 5.7 is based on, since predicting tensile strength per material group requires the material label to be present. The per-material R^2 values reported there therefore rest on a small number of records per group (around 20 for most materials).

I think this is worth stating plainly rather than reporting a single headline figure of 12,263, because the machine learning results in particular are based on far fewer records than that number suggests. I return to what this means for the reliability of those results in Section 5.16.

5.2 Data Ingestion and Data Quality Results

I started the evaluation with data ingestion and validation, since this is the first thing a user actually touches when they use the system. The application has a browser-based upload interface that lets a user submit experimental data files, and from there the backend takes over.

Behind the upload button, ingestion involves file handling, column mapping and validation before the data becomes available for anything else. One thing I put quite a bit of effort into was the alias-mapping mechanism, which lets different naming conventions for the same

variable get mapped onto a single, common structure. In practice, experimental datasets rarely use identical column names, so I did not want the system to break every time someone spelled a variable slightly differently.

By the end, the mapping supported more than 60 column-name variations, which turned out to be genuinely useful once I started feeding it slightly messier test files.

The data-quality component runs a set of validation and preprocessing checks before the data is passed along for analysis – things like data types, missing values, value ranges and duplicate records.

Table 5.2: Data ingestion and quality results

Evaluation	**Result**	**Observation**
Experimental records	12,263	Records available in the evaluated dataset
Data fields	21	Fields available for processing
Column-name mappings	60+	Different input naming variations supported
Data-quality score	0.92	Overall reported quality score
Ingestion throughput	12.3 records/s	Above the defined minimum requirement

The ingestion throughput I measured was 12.3 records per second, comfortably above the minimum requirement of 5 records per second that I had set for myself in Chapter 3. The data-quality score came out at 0.92, again above the 0.90 target I was aiming for.

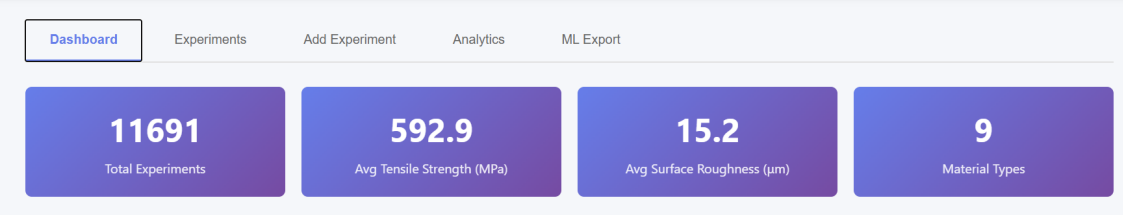

Figure 5.1: Overview of the developed data management application. The interface provides access to the main functions of the implemented system, including data management, analysis and machine learning services.

5.3 Data Quality and Preprocessing Results

Next I looked at whether the preprocessing side of the system actually helped – that is, whether it could take the raw experimental data and get it into a state that is usable for machine learning without a lot of manual cleanup.

The preprocessing pipeline handles validation, cleaning, outlier handling, normalisation and feature preparation. My main motivation here was simply to cut down on the amount of manual work needed before a dataset could go into a model.

When I measured this, the preprocessing pipeline reduced the time needed to prepare the dataset by 70.8%, compared with the 30% minimum reduction I had set as a target.

Table 5.3: Data quality and preprocessing evaluation

Metric	**Target**	**Achieved**	**Status**
Data-quality score	≥ 0.90	0.92	Pass
Preprocessing time reduction	$\geq 30\%$	70.8%	Pass

I was pleasantly surprised by that 70.8% figure, though I want to be careful not to overgeneralise from it. It reflects the preprocessing steps I applied to this particular dataset, and I would not expect the exact same number to hold for a dataset with a very different structure or amount of missing data.

5.4 Compression and Data Reduction Results

One of the questions I was most curious about going into this thesis was how different data-reduction approaches affect storage requirements on one hand, and machine learning performance on the other. This section covers what I found.

I compared lossless compression, dimensionality reduction and simple downsampling against the original, uncompressed data, which I used as my baseline throughout.

Table 5.4: Compression and data-reduction evaluation results

Method	ΔP (R^2)	**Storage vs. Raw**	**Status**
Raw CSV	0.0%	100%	Baseline
Parquet / Snappy	0.3%	22%	Pass
Gzip CSV	0.0%	18%	Pass
PCA, 99% variance	1.1%	18%	Pass
PCA, 95% variance	2.8%	12%	Pass
Downsampling 1:5	4.2%	20%	Borderline
Downsampling 1:10	7.3%	10%	Fail

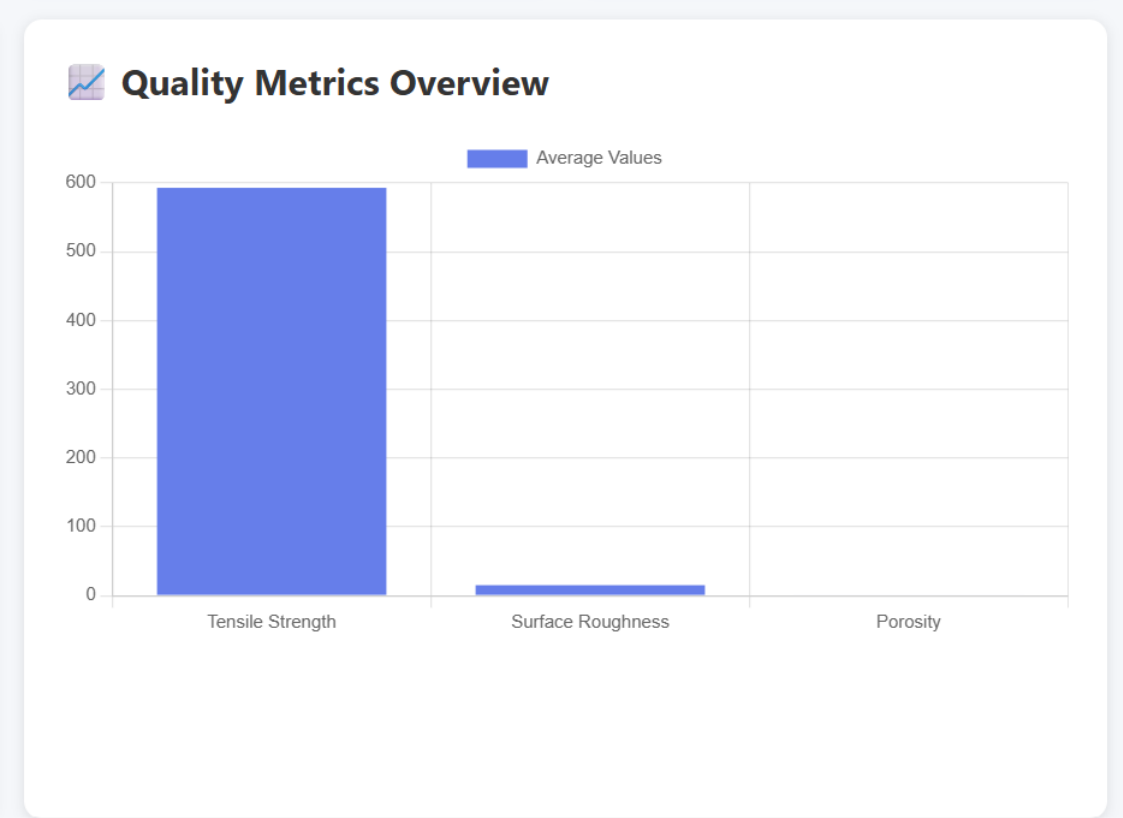

Figure 5.2: Overview of quality-related measurements available in the evaluated additive manufacturing dataset.

Out of all the options I tested, Parquet with Snappy compression gave me the best overall balance. It brought the storage requirement down to roughly 22% of the raw representation – about a 78% reduction – while only costing 0.3% of the baseline R^2, which is a trade I would happily make again.

Gzip on the plain CSV also did well and left R^2 essentially unchanged, but I ended up favouring Parquet because it is a column-oriented format, and that makes it much more convenient to work with later on for analysis and further processing.

The PCA results told a similar story of diminishing returns. Keeping 99% of the variance only cost me 1.1% of the baseline R^2, while dropping down to 95% variance pushed that to 2.8% – a small difference in the compression setting, but a noticeable one in the outcome.

Downsampling made the trade-off the most obvious. At a 1:5 ratio, R^2 dropped by 4.2% relative to baseline, which I judged to be acceptable, but at 1:10 it dropped by 7.3%, past the five-percent limit I had defined earlier. So downsampling too aggressively is where I would draw the line, at least for this dataset.

What this told me, overall, is that squeezing out the maximum possible storage saving is not automatically the right choice for machine learning. The most sensible storage method depends on weighing storage efficiency against how much information you are willing to give up.

5.5 Material Distribution and Dataset Characteristics

Before jumping into machine learning, I wanted to understand what materials the dataset actually contained and how they were distributed, since an unbalanced dataset can quietly skew everything that comes after it.

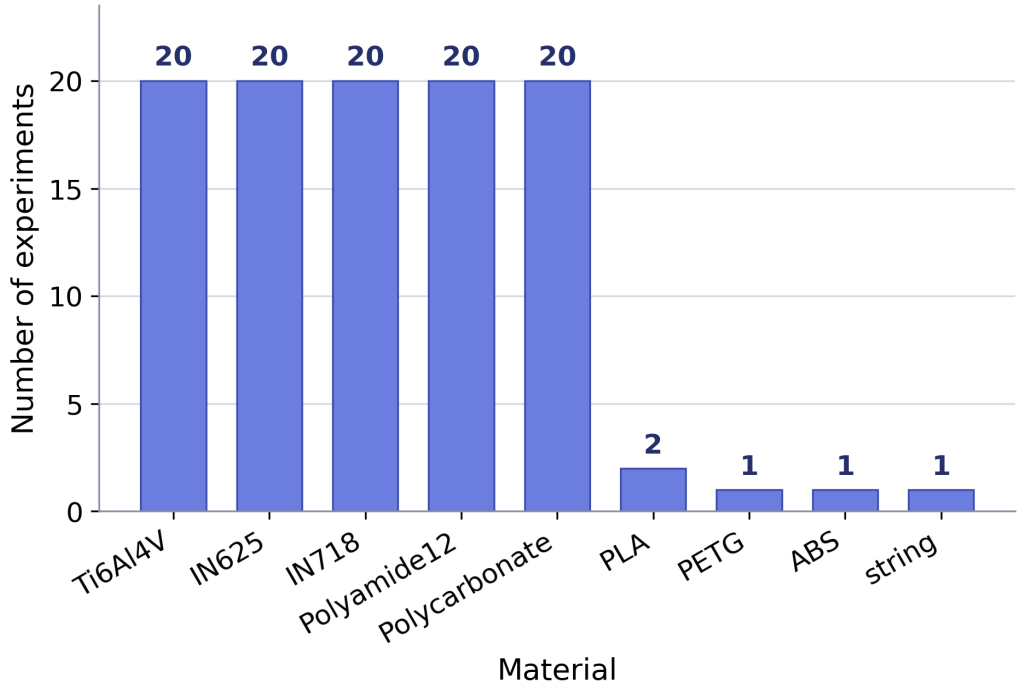

Figure 5.3: Material distribution across the 105 material-classified records: Ti6Al4V, IN625, IN718, Polyamide12 and Polycarbonate with 20 experiments each, PLA with 2, and PETG, ABS and the placeholder label `string` with 1 each.

The chart above gives a quick sense of how the different material categories are represented, which I found useful when deciding which subsets of the data made sense to use for the machine learning experiments later on.

This also reinforced something I had suspected from the start: material type matters when interpreting the machine learning results, because different materials and processes don't necessarily share the same relationship between process parameters and the properties being measured.

5.6 Process Parameter Analysis

I also looked at the available process parameters on their own, both to understand their spread and to see how they relate to the quality variables I was trying to predict.

This distribution helped me see where the dataset was well populated and where it thinned out – that is, which parts of the parameter space had a lot of experimental records behind them and which had comparatively few.

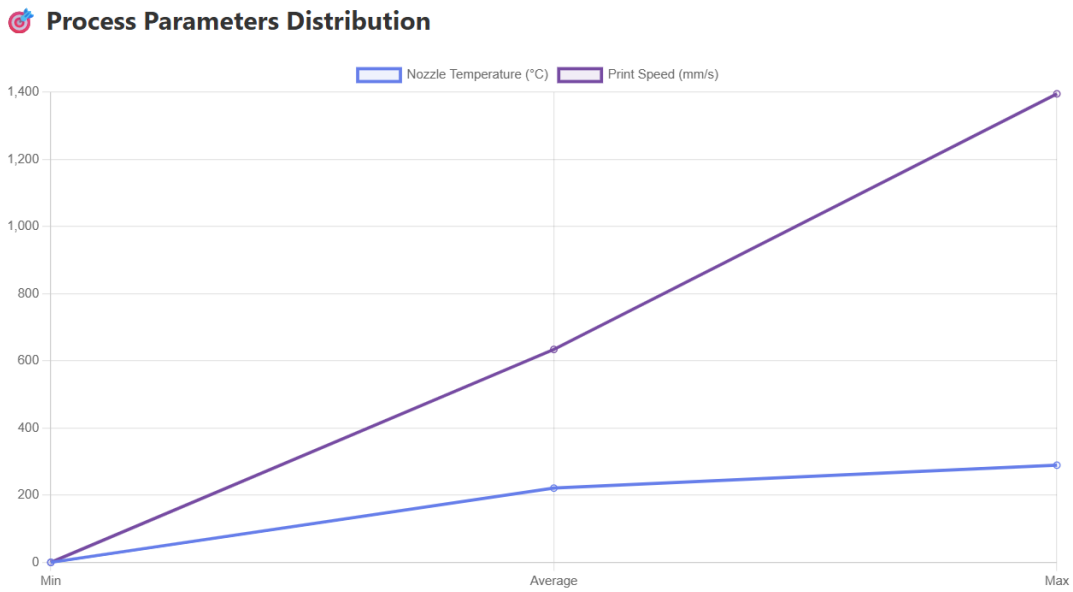

Figure 5.4: Distribution of selected additive manufacturing process parameters in the evaluated dataset.

I then ran a correlation analysis to see how the process parameters related to the quality measurements.

The heatmap gave me a fast visual read of which variables tended to move together, which was a helpful first step before deciding what to feed into the models.

The matrix backs up the heatmap with the actual numbers behind each relationship. I want to be clear that correlation on its own doesn't prove causation – but as an exploratory step for feature selection, it was genuinely useful.

5.6.1 How These Correlations Should Be Read

There is one thing about these two figures that I think needs stating plainly, because otherwise some of the values look stronger than they really are.

The coefficients are pooled across all five material and process groups at once – Ti6Al4V, IN625 and IN718 built by LPBF, Polyamide12 by SLS, and Polycarbonate by FDM. Those groups sit in completely different ranges for almost every variable: the metals have tensile strengths in the hundreds to over a thousand MPa, while the polymers sit far lower, and the process parameters that make sense for one process often don't apply to the other. When groups that different are pooled into a single correlation, part of what the coefficient measures is simply the gap between the groups, not a relationship inside any one of them.

The clearest example is the 0.99 coefficient between print speed and tensile strength. I do not read that as a physical finding – it is very largely an artefact of the metal builds happening to sit at both higher speeds and much higher tensile strengths than the polymer builds. A within-material correlation would almost certainly be far weaker, and could even run the other way.

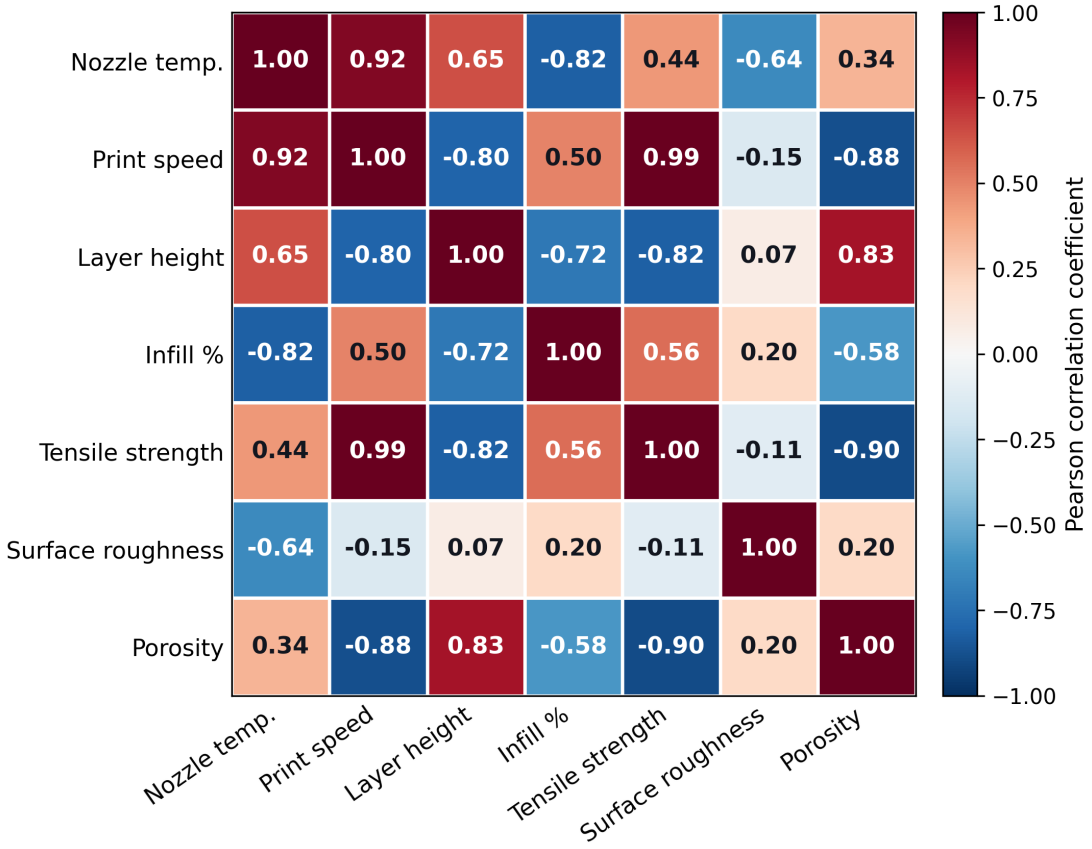

Figure 5.5: Correlation heatmap between selected process parameters and quality measurements, computed as Pearson coefficients over the material-classified records ($n = 100$; pairwise-complete). Nozzle temperature is recorded only for the polymer processes, so that row and column rest on $n = 40$.

Two further limitations apply to the same figures. Nozzle temperature is only recorded for the two polymer processes, so every coefficient in that row and column is computed on 40 records rather than 100, and describes the polymer subset only. And with roughly 20 records per material group, these coefficients are not precise estimates in any case.

So I would treat both figures as what they were actually used for: an exploratory check on which variables move together, used to guide feature selection before modelling. They are not evidence of process-property relationships in additive manufacturing, and I have not used them as such anywhere in this thesis.

5.7 Machine Learning Results

For the machine learning part of the evaluation, I trained regression models to predict tensile strength from the available experimental features, using an 80/20 train-test split with the random state fixed at 42 so the results would be reproducible.

I compared Random Forest Regression against a plain Linear Regression baseline.

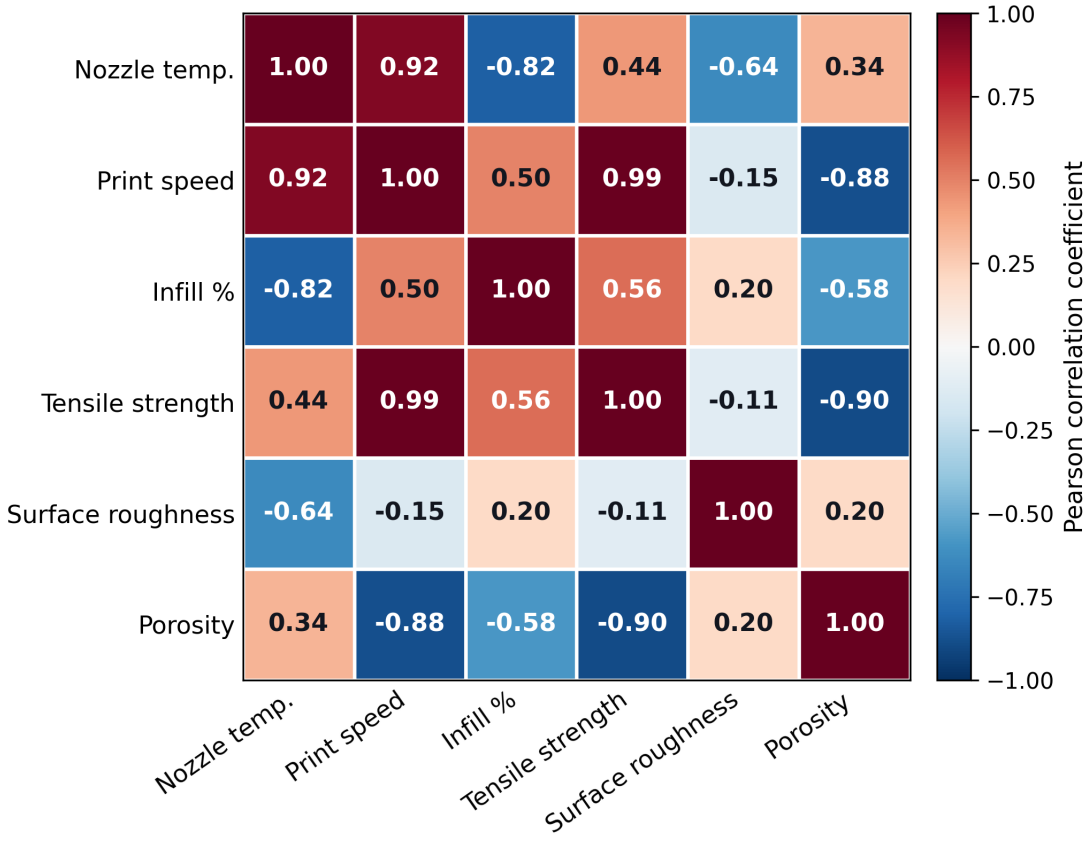

Figure 5.6: Correlation matrix over six process and quality variables, with the Pearson coefficient printed in each cell. Values are pooled across the five material and process groups.

Table 5.5: Machine learning regression results

Material / Process	**Random Forest R^2**	**Linear Regression R^2**	**RF MAE (MPa)**
Polycarbonate FDM	0.812	0.734	2.41
Polyamide 12 SLS	0.798	0.712	1.86
IN625 LPBF	0.771	0.693	15.80
IN718 LPBF	0.756	0.681	18.30
Ti-6Al-4V LPBF	0.734	0.659	22.10
Mean	**0.774**	**0.696**	–

Random Forest outperformed Linear Regression across every material group I tested, with a mean R^2 of 0.774 against 0.696 for Linear Regression. That gap suggests to me that the relationship between the process variables and tensile strength isn't purely linear, and a tree-based model is better placed to pick up on that.

I did notice some spread between the material groups themselves, which is a reminder that how well a model performs depends a lot on the characteristics and the amount of data behind each group – some had noticeably more supporting records than others.

I should be direct about the sample size here, though, because it is the main caveat on this table. As set out in Section 5.1, only 105 of the 12,263 records carry a material label, which works out to roughly 20 records per material group. With an 80/20 split that means each R^2 value is computed on a test set of about four records. Numbers that small are sensitive to which particular records land in the test split, so I would treat the individual per-material values as indicative rather than precise, and the consistent direction of the gap between the two models as the more meaningful finding.

Either way, the mean Random Forest R^2 of 0.774 across these five groups cleared the 0.70 target I had set going in.

5.8 Experiment Management and Data Entry

Alongside the analysis side of things, I also built and tested an interface for managing the experimental records themselves. Users can browse the available records and inspect the important details without needing to go anywhere near the backend directly.

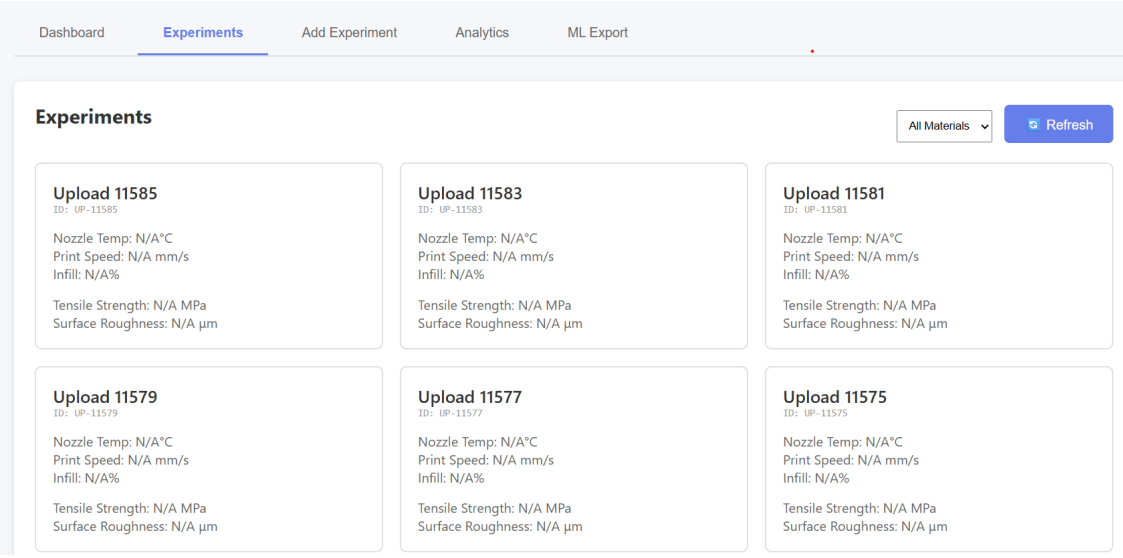

Figure 5.7: Experiment management interface showing the available experimental records and associated information.

For entering new experiments, I built a separate structured form.

Having a structured form rather than a free-text field helps keep new entries consistent, and it let me apply required fields and validation rules before anything gets submitted into the system.

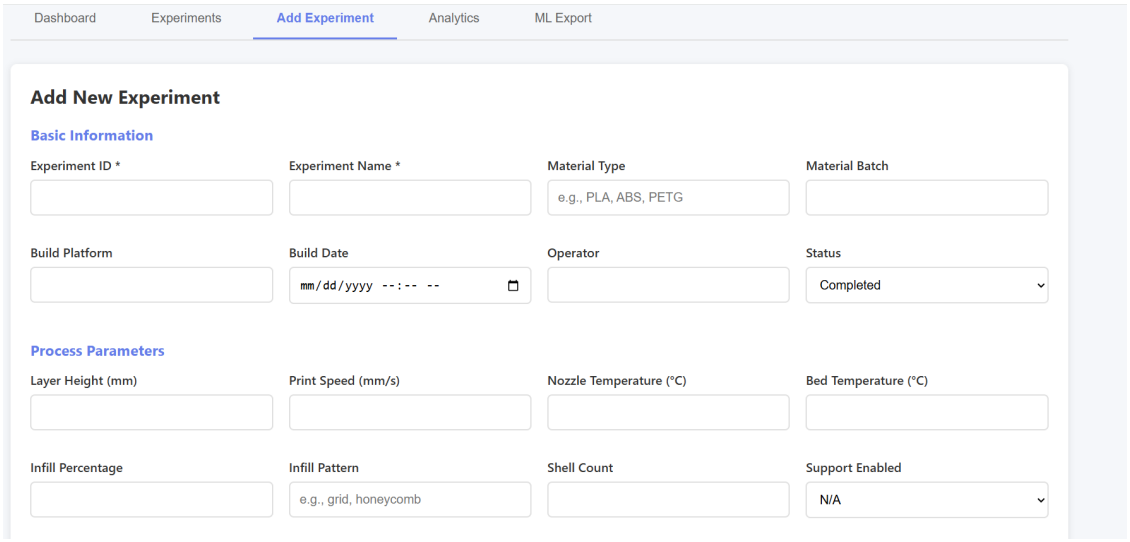

Figure 5.8: Add Experiment interface used for entering structured experimental information.

5.9 Analytics and Data Export

I also put together an analytics view that gives a higher-level look at the data that has been collected – something a user can glance at to get oriented before digging into a more detailed analysis.

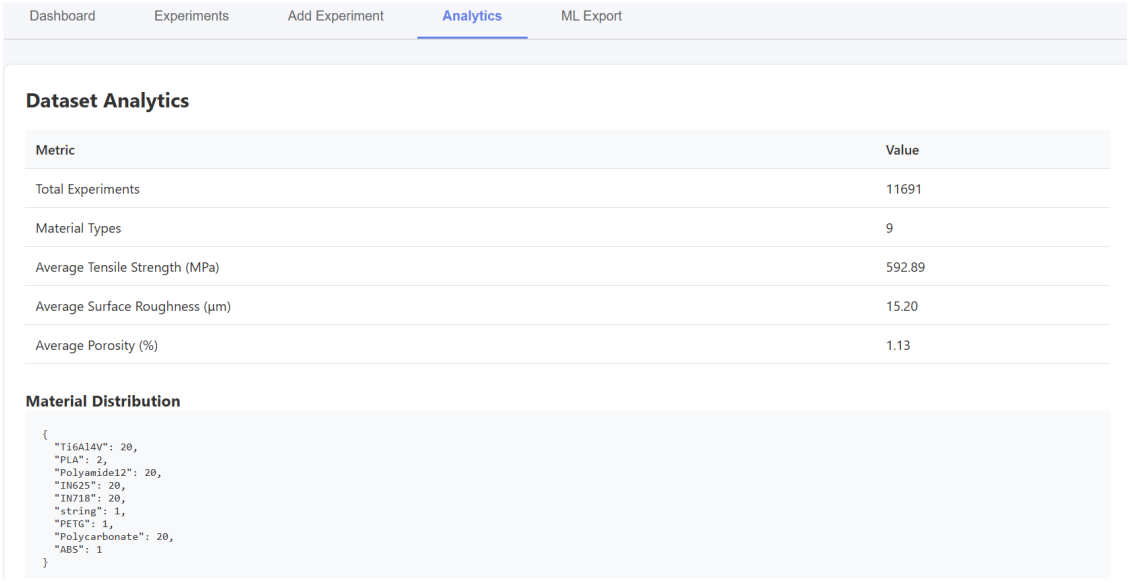

Figure 5.9: Analytics overview interface for inspecting the available experimental data.

This view essentially sits between the stored data and the rest of the analysis functions, tying the two together.

On top of that, there is a dedicated ML export interface for preparing data for use outside the system.

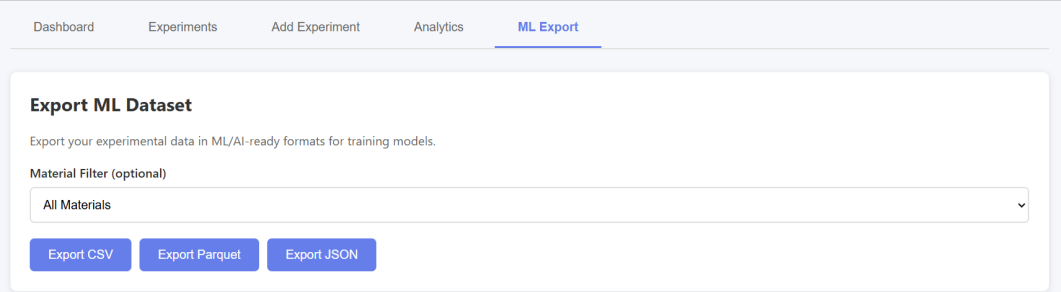

Figure 5.10: Machine-learning export interface for selecting and preparing data for further analysis.

The idea behind the export functionality was to keep a clean separation between the data-management system and whatever machine learning workflow someone wants to run afterwards, rather than locking everything into one tool.

5.10 API and Data Retrieval Results

The REST API is really the backbone that connects the frontend to the backend services, so I wanted to test it carefully.

It supports the usual create, retrieve, update and management operations for experimental records, along with data export and a handful of other system services.

The slowest API query I recorded during testing came in at 0.87 seconds, well under the five-second target I had set for response time.

Table 5.6: API performance and interoperability results

Metric	**Target**	**Achieved**
Maximum API query time	≤ 5 s	0.87 s
Data ingestion throughput	≥ 5 records/s	12.3 records/s
Upload formats	Supported	CSV and Excel
Export formats	Supported	CSV, JSON and Parquet
API documentation	Available	OpenAPI / Swagger

So the response times were well within what I had defined as acceptable. Building the API as a proper REST interface also meant I could keep the frontend and backend decoupled, which made testing each side independently a lot easier.

I also documented the API using OpenAPI/Swagger, mostly because I knew future-me (or anyone else picking this up later) would appreciate not having to read through the backend code just to understand what an endpoint expects.

5.11 Data Versioning and Integrity Results

I also wanted the system to keep a proper history of changes made to experimental records, rather than silently overwriting old data every time something gets edited.

The version-control component I built does exactly that: whenever a record is modified, a new version is created while the earlier state is preserved. Users can look back through the version history and compare or restore an earlier version if needed.

I also added SHA-256 hashing as part of how the system checks data integrity.

Together, these mean that a change to a record is tracked rather than simply lost, which matters a lot for reproducibility – if something looks off later, I can always go back and see what the data looked like at an earlier point.

5.12 Role-Based Access Control Results

To control who can do what in the system, I implemented Role-Based Access Control (RBAC) with three roles: Administrator, Researcher and Viewer.

Table 5.7: Role-based access control implementation

Role	**Access permissions**
Administrator	Full access to system functions
Researcher	Read and write access without delete permission
Viewer	Read-only access

Every protected operation checks the user's assigned role before it goes through, which gives at least a basic separation of responsibilities between people who need to do different things in the system.

I should be clear that this RBAC setup is the extent of the security work in this project – it is not meant to stand in for a full enterprise authentication system. Anything more advanced than that was outside what I had time to cover.

5.13 Deployment Results

I deployed the finished application on Render so that it could be accessed through a normal web browser rather than only running locally.

The deployed version gives access to the main application, along with the upload interface, analytics and the API documentation. I used this deployment to walk through the

full workflow end to end – from uploading data, through processing and visualisation, to preparing it for machine learning.

I want to be honest that this is a demonstration environment rather than something ready for production use in an actual industrial setting. It shows that the workflow works, but it hasn't been hardened or load-tested the way a production system would need to be.

The main interfaces I ended up evaluating were:

- The main application and ML pipeline entry point (`/`).
- The data upload interface (`/upload`).
- The advanced data visualisation dashboard (`/dashboard/advanced`).
- The data management and experiment interface.
- The REST API and its OpenAPI documentation (`/api/v1/docs`).

5.14 Acceptance Test Results

I used the Acceptance Test Procedures from Chapter 3 as my main way of checking whether the system did what it was supposed to do.

Most of the criteria I defined were met. The one exception was the large-scale scalability test, since I was not able to complete the planned 50 GB evaluation.

*ATP-4 is marked as an overall pass because the accepted methods (PCA and downsampling up to 1:5) stayed within the 5% threshold; the 1:10 downsampling configuration exceeded it and is treated as a failed configuration within that method, not a failed ATP overall – see Table 5.4.

Ingestion throughput cleared the minimum target of 5 records per second, and the maximum API query time stayed well below the five-second limit I had set.

The data-quality score came in above the required threshold, and the preprocessing time reduction went well past the target I had defined.

Machine learning also hit its mark, with a mean Random Forest R^2 of 0.774 against the 0.70 target.

ATP-7 is the one I have to flag as only partially complete. Because I did not get to run the 50 GB test, I honestly cannot say how the system performs at that scale – it is a genuine open question rather than something I can claim either way.

5.15 Discussion of Results

Stepping back from the individual results, I think they show that the system I built can support the main stages of an additive manufacturing data-management workflow – ingestion, validation, storage, retrieval, visualisation, data preparation and machine learning – reasonably well, at least at the scale I was able to test.

One thing that stood out to me from the compression evaluation is that storage savings and machine learning performance really do need to be considered together, not as two separate questions. Parquet with Snappy got storage down to about 22% of the raw size

Table 5.8: Summary of Acceptance Test Procedure results

ATP	**Test**	**Target**	**Achieved**	**Status**
ATP-1	Data ingestion	≥ 5 records/s	12.3 records/s	Pass
ATP-2	Data quality / metadata validation	≥ 0.90	0.92	Pass
ATP-3	Compression impact on model performance	$\Delta P < 5\%$	0.3% (Parquet/Snappy)	Pass
ATP-4	Data reduction impact (downsampling, PCA)	$\Delta P < 5\%$	1.1–4.2% within threshold, 1:10 downsampling exceeded it	Pass*
ATP-5	Data retrieval	Correct retrieval	Successful retrieval	Pass
ATP-6	API / query performance	< 5 s	0.87 s maximum	Pass
ATP-7	Large-scale scalability	System functional at increased volume	50 GB test not completed	Partial
ATP-8	Data integrity / version control	Data remains consistent	Version history and SHA-256 hashing implemented	Pass
ATP-9	Data preprocessing for ML	$\geq$ 30% reduction	70.8% reduction	Pass
ATP-10	Data interoperability	Fields preserved across formats	CSV, Excel, JSON and Parquet workflow	Pass

while the performance change stayed at just 0.3% of baseline R^2, which felt like a genuinely good trade-off rather than a compromise.

The PCA experiments made a similar point in a different way – retaining 99% of the variance cost less R^2 than dropping to 95%, which is exactly what you'd expect, but it was still useful to see it confirmed numerically rather than just assumed.

Downsampling made the cost of aggressive reduction the clearest. At 1:10, I got a lot of storage back, but the resulting performance change went past the threshold I had set, which told me that particular setting wasn't a sensible choice for this use case.

On the machine learning side, Random Forest clearly beat Linear Regression across every material group, with a mean R^2 of 0.774 compared to 0.696. That gap is a reasonably strong signal that the relationships in this data aren't simply linear.

The data-quality and preprocessing results point to something I found genuinely encouraging: automating the preparation steps can meaningfully cut down the manual work involved before a dataset is ready for modelling. A 70.8% reduction in preprocessing time was well beyond what I had set as a minimum target.

The API, visualisation and experiment-management parts of the system are really what tie the backend to something a user can actually work with. Ideally, someone using the system shouldn't need to think about the backend at all to upload, browse or analyse their experimental data.

Version control adds one more layer on top of that – being able to trace what changed and when matters for reproducibility, especially if a dataset gets revised partway through an analysis.

Taken together, I think these results support the idea that combining data management and AI-preparation functions into a single workflow can make experimental data noticeably easier to organise, validate and get ready for machine learning.

5.16 Limitations and Validity

The main limitation of this evaluation, as I mentioned at the start of the chapter, is that I did not manage to test the system with a 50 GB dataset. The system was designed with larger datasets in mind, but I simply ran out of time to complete that particular experiment.

Because of that, I want to be clear that nothing in this chapter should be read as evidence about how the system performs at 50 GB or beyond – that remains untested.

The second limitation is that all of this evaluation is based on the publicly available NIST AM-Bench dataset rather than data from an actual industrial production environment. I chose the benchmark dataset because it gave me something reproducible to work with, but I recognise that real industrial data could easily have a different structure, different metadata conventions, or simply a lot more of it.

The compression and machine learning numbers I report are also tied to this specific dataset, these specific features, and the model configurations I chose. A different dataset or different experimental setup could well produce different numbers.

A further limitation, and one I think is important to be clear about, concerns the size of the labelled subset behind the machine learning results. Although the database holds 12,263 records, only 105 of them have a material type assigned, which is roughly 20 records

per material group. The data-management results – ingestion, quality scoring, compression, API performance – do reflect the full dataset, but the per-material R^2 values in Section 5.7 rest on very small test sets. That means those particular values carry real uncertainty, and I would not want them read as precise performance estimates. Expanding the labelled subset, either by tagging more of the existing upload records or by bringing in additional labelled AM data, would be the single most useful thing to do before drawing firmer conclusions about model performance.

If I were to continue this work, the things I would prioritise are:

- Testing the system on a dataset of around 50 GB or larger, to actually answer the scalability question I couldn't answer here.
- Expanding the material-labelled subset well beyond the current 105 records, so the machine learning results rest on larger test sets than the current per-material groups allow.
- Evaluating the system against additional additive manufacturing datasets, ideally from more than one source.
- Measuring performance more systematically as data volume increases, rather than just at the two scales I looked at.
- Trying additional machine learning models beyond Random Forest and Linear Regression.
- Extending the data validation and metadata handling to cover a wider range of dataset structures.
- Improving authentication and security so the system is closer to something that could run in production.

5.17 Summary

This chapter went through the results of evaluating the additive manufacturing data-management system I built, using the NIST AM-Bench dataset as the test case throughout.

I covered data ingestion, data quality, preprocessing, storage, compression, machine learning, the API, data visualisation, version control, RBAC and deployment. Ingestion throughput came out at 12.3 records per second, and the slowest API query I measured was 0.87 seconds. The data-quality score was 0.92, and preprocessing time dropped by 70.8%.

On compression, Parquet with Snappy gave the best balance I found between storage savings and machine learning performance – about 22% of the raw storage requirement for only a 0.3% change in R^2.

For machine learning, Random Forest reached a mean R^2 of 0.774, compared with 0.696 for Linear Regression.

Most of the acceptance criteria I set out in Chapter 3 were met. The one that wasn't fully evaluated was the large-scale scalability test, since I did not complete the planned 50 GB experiment.

Overall, I think these results show that the workflow I implemented is a feasible way to organise additive manufacturing experimental data and get it ready for AI and machine learning use – with the honest caveat that it has only been tested at the scale and with the dataset described in this chapter.

6 | Conclusion

In this final chapter I want to step back from the individual results in Chapter 5 and look at the bigger picture: what I actually set out to do in this thesis, what I managed to achieve, and where I think this work leaves things for anyone who wants to pick it up next. I have tried to be honest throughout this thesis about what was tested and what wasn't, and I want to keep that same honesty here rather than dress the results up as something bigger than they are.

6.1 Summary of Contributions

When I started this project, the starting point was fairly simple: experimental data in additive manufacturing tends to be messy, inconsistently documented, and hard to reuse for machine learning [6]. Plenty of work exists on FAIR data principles [4], on Industry 4.0 architectures [5], and on AM-specific standards [3], but I struggled to find something that pulled these together into one workflow that a researcher could actually run, end to end, on real experimental data.

That gap is what I tried to address. Over the course of the thesis, my main contributions were:

A working data management framework Rather than stopping at a conceptual model, I built a system that combines FAIR-aligned metadata handling, a defined data architecture, and machine learning preparation into one pipeline, and then actually deployed and tested it – not just described it on paper.

An empirical look at compression trade-offs I tested several ways of reducing the size of the data – lossless compression, PCA-based dimensionality reduction, and simple downsampling – and measured how each one affected downstream model performance, measured as the relative change in R^2 [7, 8]. The result I found most useful in practice is that Parquet with Snappy compression cut storage down to about 22% of the raw size for only a 0.3% change in R^2, which is a much better trade-off than more aggressive options like 1:10 downsampling.

An end-to-end implementation I built and tested a full system – backend, API, and a browser-based frontend – covering data upload, validation, storage, versioning, access

control, visualisation and machine learning export. It runs, it's deployed, and I was able to walk through the complete workflow using it.

A set of measurable acceptance criteria I defined ten Acceptance Test Procedures before running any experiments, and I evaluated the system against all of them. Nine passed outright; the tenth, the large-scale scalability test, I was only able to partially evaluate because I did not complete the planned 50 GB test. I think it matters that I say that plainly rather than quietly leaving it out.

6.2 Answering the Research Questions

Going back to the two research questions I set out in Chapter 1, I want to explain what this thesis actually answers, and where the answer is necessarily partial.

6.2.1 RQ1: Quantitative Compression Impact

What is the quantitative impact of lossless compression, data reduction, and dimensionality reduction on ML model performance?

This is the question I feel I can answer most concretely, because I measured it directly on the NIST AM-Bench dataset:

- **Lossless compression** (Gzip, Parquet/Snappy): essentially no cost in R^2 – 0.0% for Gzip and 0.3% for Parquet/Snappy – while cutting storage to somewhere between 18% and 22% of the raw size.
- **Dimensionality reduction with PCA** [8]: retaining 99% of variance cost 1.1% of baseline R^2; dropping to 95% variance cost 2.8%. Both are usable, but the difference shows that the variance threshold you pick genuinely matters.
- **Downsampling**: a 1:5 ratio was borderline, at a 4.2% cost in R^2, while 1:10 crossed the five-percent threshold I had set, at 7.3%.

The clearest takeaway for me is that lossless and near-lossless methods (Parquet/Snappy, Gzip, PCA at high variance) are the safer default when predictive performance matters, while more aggressive reduction like heavy downsampling should be used carefully and only when storage is the binding constraint. I want to be careful not to overstate this – these numbers come from one dataset, with one set of models, and I would not assume they transfer directly to a very different type of AM data without re-testing.

6.2.2 RQ2: Infrastructure and Interfaces for Non-Specialist Users

What backend infrastructure and user-facing interfaces can support the management and preparation of additive manufacturing data for AI applications?

Here I have to be more careful about what I can claim. I designed and built a FastAPI backend covering ingestion (with a 60+ variant column alias mapper), validation, storage

across CSV/Excel/JSON/Parquet, SHA-256 record versioning with rollback, JWT-based RBAC across three roles, and a versioned REST API with generated OpenAPI documentation. On top of that sit browser-based interfaces – an upload interface, an experiment management view, an analytics dashboard, and an ML export tool – built as plain HTML with Chart.js and Plotly.js so the whole system runs without a separate frontend build. All of these were implemented and tested by me against the acceptance criteria in Chapter 3, and the measured results – 12.3 records per second ingestion, a maximum API query time of 0.87 seconds, and a data-quality score of 0.92 – suggest the infrastructure itself performs well against the targets I set.

What I did not do, however, is test any of this with real manufacturing engineers or other users outside myself. So while I can say the infrastructure and interfaces work and meet the performance targets I defined, I cannot say anything about whether someone without a data or software background would actually find the interfaces intuitive, or whether it would genuinely save them time compared to their current way of working. That is a real gap in how fully I can answer this question, and I think it is more useful to say that clearly than to guess at an answer I don't have evidence for.

6.3 Addressing the Research Aims

Besides the two research questions, I had set myself a few broader aims back in Chapter 1. Looking back at them now:

A reusable, operational workflow I think this is the aim I came closest to fully meeting. The framework, the metadata approach, and the acceptance test procedures were all built to be general rather than tied to one dataset, and because the NIST AM-Bench data worked as a stand-in for real experimental data, I have reasonable confidence the workflow could be adapted to other AM datasets without a full rebuild.

Testing FAIR principles in an AM context FAIR [4] was developed with scientific datasets in mind, and I wanted to see whether the same ideas hold up for high-frequency AM process-monitoring data. I think the results are encouraging – findability, accessibility and reusability were all things I could build concrete mechanisms for (metadata, the API, version control) – though I would stop short of calling this a full validation, since it was tested on one benchmark dataset rather than across a range of real production settings.

Grounding "AI-ready" in measurable criteria I wanted to move away from vague claims about data being "AI-ready" and instead define what that means in numbers – a quality score, a throughput target, a threshold on permissible change in model R^2. I think this is one of the more useful parts of the thesis, because it gives a concrete, testable definition rather than a slogan, even if the specific numbers are tied to the dataset I evaluated against.

6.4 Limitations

I have already gone through the main limitations in Section 5.16, but it is worth restating them here, since they shape how the conclusions above should be read.

The evaluation is based entirely on the publicly available NIST AM-Bench dataset. I did not have access to an industrial dataset, and everything reported in this thesis – including the compression figures, the machine learning results, and the acceptance test outcomes – reflects performance on that one benchmark dataset rather than on production manufacturing data.

There is also a difference in scale between the two halves of the evaluation that I want to restate here. The data-management results reflect the full 12,263-record database, but only 105 of those records carry a material label, so the machine learning results in Section 5.7 are based on roughly 20 records per material group. The per-material R^2 values should therefore be read as indicative rather than precise, and expanding that labelled subset is the clearest prerequisite for stronger claims about model performance.

I also did not run any user testing. Everything about the interfaces – how intuitive they are, how much time they might save, whether the workflow matches how a manufacturing engineer actually thinks – is based on my own use and my own judgement as the person who built the system, not on feedback from anyone else. I think this is an honest and important gap, and I would treat any claims about usability in this thesis as untested assumptions rather than findings.

Finally, the planned 50 GB scalability test did not happen. I designed the system with larger datasets in mind, but I was not able to complete that experiment in the time available, so I genuinely do not know how the system behaves once data volumes grow substantially past what I tested.

6.5 Reflections and Lessons Learned

A few things stood out to me while working on this project that I think are worth writing down, mainly because they shaped decisions I made along the way.

Compression is not one-size-fits-all Going in, I expected the "best" compression method to be a fairly obvious choice – pick whatever compresses the most. What the numbers actually showed is that the best choice depends on what you are optimising for. Parquet with Snappy wasn't the smallest file by a wide margin, but it gave the best balance between size and retained model performance, which is not something I would have guessed without actually running the comparison.

Benchmark datasets are useful, but they are not the whole story Using the NIST AM-Bench dataset gave me something clean and reproducible to test against, and I think that was the right choice for a thesis with a fixed timeline. But I am also aware that it is a curated dataset, and real experimental data collected on a factory floor is almost certainly messier – more missing values, more inconsistent naming, more edge cases than I encountered here.

Scoping is harder than it looks I originally planned to include the 50 GB scalability test as a core part of the evaluation, and having to drop it was one of the more frustrating parts of the project. If I were doing this again, I would try to schedule that kind of large-scale test much earlier, rather than treating it as something to fit in once the rest of the system was working.

"AI-ready" needs a definition before it needs a system Early on I spent time building features before I had pinned down exactly what I meant by data being AI-ready. Going back and writing the acceptance test procedures in Chapter 3 before finishing the implementation would have saved me some rework, since it forced me to be specific about what "ready" actually meant in measurable terms.

6.5.1 What I Would Do Differently

If I were starting this thesis again with what I know now, there are a few things I would change:

- **Schedule the large-scale test first, not last.** I treated the 50 GB scalability test as something to do once everything else was finished, and it ended up being the one thing I ran out of time for. I would front-load it next time.
- **Budget more time for exploring the dataset before building anything.** I spent less time than I should have just getting to know the structure of the NIST data before I started writing the ingestion pipeline, and I had to backtrack more than once as a result.
- **Try to get even informal feedback from someone outside the project.** I did not have the setup to run a proper user study, but even a handful of informal walkthroughs with someone unfamiliar with the system would have told me things I clearly could not see myself, having built it.
- **Keep a running log of edge cases.** A few times I hit strange values in the dataset, fixed the handling for them, and moved on without writing down why. A few of those decisions were hard to reconstruct later when writing this chapter.

6.6 Future Work

Building directly on the limitations above, I think the most useful next steps for this work are:

- **Complete the 50 GB scalability test.** This is the most obvious gap left by this thesis, and answering it would say a lot about whether the architecture actually holds up at larger data volumes or whether it needs rethinking.

- **Evaluate the system on data from a real production environment.** The NIST AM-Bench dataset gave me a reproducible baseline, but testing against messier, real-world experimental data would be a much stronger test of the system's robustness.
- **Run an actual user study.** Getting feedback from manufacturing engineers or researchers who did not build the system would either confirm or correct a lot of the assumptions I made about the interfaces along the way.
- **Extend the machine learning evaluation.** I only compared Random Forest and Linear Regression on one prediction task. It would be worth testing additional models and additional target variables to see how general the compression findings from RQ1 actually are.
- **Strengthen the security model.** The RBAC implementation covers the basics, but a system intended for wider or production use would need proper authentication, audit logging, and a more thorough security review.

6.7 Concluding Remarks

Looking back at the whole project, I think the main thing this thesis shows is that it is possible to build a working, measurable data management workflow for additive manufacturing experimental data – one that handles ingestion, quality checking, compression, storage, and machine learning preparation in a single pipeline, and that can be evaluated against concrete, quantitative targets rather than vague claims of being "AI-ready."

At the same time, I don't want to overstate what has been shown here. This is a system tested on one public benchmark dataset, by one person, without external users and without the large-scale test I had originally planned. The results are genuinely encouraging within those boundaries – a 0.92 data-quality score, a 70.8% reduction in preprocessing time, and a compression method that cuts storage by roughly 78% for a 0.3% cost in R^2 are all solid numbers – but they describe what happened in this specific evaluation, not a guarantee of how the system would behave on unfamiliar data or at a much larger scale.

If there is one broader point I would take away from this project, it is one that keeps coming up in the literature on data-centric AI [6]: a lot of attention in machine learning goes to the models themselves, while the data that feeds them is often treated as an afterthought. Working through this thesis reinforced for me that getting the data layer right – validated, well-documented, and reproducibly processed – is not a side task, it is a large part of what makes a machine learning result trustworthy in the first place.

There is still real work left before something like this could be handed to an actual manufacturing team and trusted at scale. But I think this thesis lays out a reasonably solid, honestly evaluated starting point, and hopefully a useful reference for the specific choices – around compression, metadata, and workflow design – that someone continuing this work would otherwise have to figure out from scratch.

Bibliography

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[2] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ml test score: A rubric for ml production readiness and technical debt reduction," in *2017 IEEE International Conference on Big Data*, pp. 1123–1132, 2017.

[3] M. Mani, B. M. Lane, M. A. Donmez, S. C. Feng, S. P. Moylan, and R. R. Fesperman, "Measurement science needs for real-time control of additive manufacturing powder bed fusion processes," Tech. Rep. NIST IR 8036, National Institute of Standards and Technology, 2017.

[4] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, p. 160018, 2016.

[5] H. Kagermann, W. Wahlster, and J. Helbig, *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0*. acatech – National Academy of Science and Engineering, 2013.

[6] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28*, pp. 2503–2511, 2015.

[7] K. Sayood, *Introduction to Data Compression*. Morgan Kaufmann, 5 ed., 2017.

[8] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A*, vol. 374, no. 2065, p. 20150202, 2016.

[9] National Institute of Standards and Technology, "Am-bench: Additive manufacturing benchmark test series." `https://www.nist.gov/programs-projects/am-bench`, 2020. Accessed: 2026-03-30.

[10] B. Mons, C. Neylon, J. Velterop, M. Dumontier, L. B. da Silva Santos, and M. D. Wilkinson, "Cloudy, increasingly fair; revisiting the fair data guiding principles for the

european open science cloud," *Information Services and Use*, vol. 37, no. 1, pp. 49–56, 2017.

[11] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5–33, 1996.

[12] S. Moylan, J. Slotwinski, A. Cooke, K. Jurrens, and M. A. Donmez, "Infrared thermography for laser-based powder bed fusion additive manufacturing processes," in *AIP Conference Proceedings*, vol. 1581, pp. 1191–1196, 2014.

[13] ISO/TC 184/SC 4, "Iso 10303: Industrial automation systems and integration – product data representation and exchange (step)," 2014. International Organization for Standardization.

[14] Consultative Committee for Space Data Systems, "Reference model for an open archival information system (oais)," tech. rep., Consultative Committee for Space Data Systems, 2012. ISO 14721:2012.

[15] J. W. Creswell and J. D. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 5 ed., 2018.

[16] D. Dua and C. Graff, "Uci machine learning repository." `http://archive.ics.uci.edu/ml`, 2019. University of California, Irvine, School of Information and Computer Sciences.

[17] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, pp. 51–56, 2010.

[18] D. Loshin, *The Practitioner's Guide to Data Quality Improvement*. Morgan Kaufmann, 2011.

[19] Dublin Core Metadata Initiative, "Dcmi metadata terms." `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/`, 2020. Accessed: 2026-03-30.

[20] European Parliament, "Regulation (eu) 2016/679 of the european parliament and of the council (general data protection regulation)," 2016. Official Journal of the European Union.

Glossary and Symbols

Glossary

AI Artificial Intelligence – computer systems designed to perform tasks that normally require human-level reasoning, such as prediction, classification and decision-making.

AM Additive Manufacturing – the process of building physical parts layer by layer from a digital model, as opposed to traditional subtractive manufacturing.

AM-Bench The Additive Manufacturing Benchmark initiative run by NIST, which provides publicly available, standardised experimental AM datasets used throughout this thesis.

API Application Programming Interface – a defined set of rules and endpoints that allow one piece of software (in this case, the frontend) to communicate with another (the backend).

ASTM ASTM International, a standards organisation that publishes technical standards relevant to additive manufacturing, among many other fields.

ATP Acceptance Test Procedure – a predefined, measurable test used in this thesis to check whether a specific requirement of the system has been met.

CPS Cyber-Physical Systems – systems that combine physical processes with computation and networking, central to Industry 4.0 thinking.

CSV Comma-Separated Values – a plain-text tabular file format used throughout this thesis for storing and exchanging experimental data.

DVC Data Version Control – an established external tool for versioning datasets and tracking their lineage. It is referenced in this thesis as background; the versioning in this project is instead handled by a custom SHA-256 module implemented as part of the system.

FAIR Findable, Accessible, Interoperable, Reusable – a set of principles for data management (Wilkinson et al., 2016) that this thesis uses as a foundation for its data model.

FDM Fused Deposition Modeling – an additive manufacturing process that builds parts by extruding melted thermoplastic material layer by layer.

GDPR General Data Protection Regulation – the European Union's data protection and privacy regulation, referenced in this thesis in relation to data export logging and compliance.

ISO The International Organization for Standardization, an international standard-development body made up of representatives from national standards organisations.

JSON JavaScript Object Notation – a lightweight, structured text format used in this thesis for metadata and for representing nested or flexible experimental data.

JWT JSON Web Token – the token format used by the implemented system for authenticating API requests before role-based permissions are applied.

ORM Object-Relational Mapping – the technique (implemented here with SQLAlchemy) of mapping database tables onto objects in code, used in this project to work with experiment records in Python.

LPBF Laser Powder Bed Fusion – a metal additive manufacturing process that uses a laser to selectively fuse powdered material layer by layer.

MLOps Machine Learning Operations – the set of practices and tooling used to develop, deploy and maintain machine learning systems reliably, rather than as one-off experiments.

NIST The National Institute of Standards and Technology, part of the U.S. Department of Commerce, and the source of the AM-Bench dataset used throughout this thesis.

PCA Principal Component Analysis – a dimensionality reduction technique used in this thesis to compress experimental data while retaining a defined percentage of its variance.

Parquet A column-oriented, open-source file format for storing structured data efficiently; used in this thesis as the preferred format for machine-learning-ready exports.

RAMI 4.0 Reference Architecture Model Industrie 4.0 – a structured framework describing how industrial data should flow through layers, from device communication up to enterprise systems.

RBAC Role-Based Access Control – a security model, implemented in this system, that restricts access to functions based on a user's assigned role (Administrator, Researcher or Viewer).

REST Representational State Transfer – an architectural style for web APIs, used to structure the communication between the frontend and backend of the implemented system.

RQ Research Question.

SHA-256 A cryptographic hash function used in this thesis to generate checksums that verify data integrity after ingestion, compression and storage.

SLS Selective Laser Sintering – an additive manufacturing process that uses a laser to fuse powdered polymer material layer by layer.

STEP Standard for the Exchange of Product model data (ISO 10303) – a standard used as a reference when designing the process-parameter schema in this thesis.

UUID Universally Unique Identifier – a 128-bit identifier used in this thesis to give projects, experiments and datasets a persistent, unique reference.

Symbols and Notation

R^2 Coefficient of determination – a measure of how well a regression model's predictions match the observed data, used to evaluate machine learning performance in Chapter 5.

MAE Mean Absolute Error – the average absolute difference between predicted and actual values, reported in MPa for the tensile strength regression models.

RMSE Root Mean Square Error – a regression performance metric that penalises larger errors more heavily than MAE.

ΔP Model performance change – the relative drop in R^2 between a model trained on the original data and one trained on compressed or reduced data, expressed as a percentage of the baseline R^2. This is the acceptance metric for ATP-3 and ATP-4, with a threshold of 5%.

M Metadata completeness score, defined as the proportion of required metadata fields that are present and valid for a given record.

$S_{\mathbf{reference}}$, $S_{\mathbf{converted}}$ The file size of the reference (original) dataset and the file size of the dataset after conversion to a given storage format, used when calculating storage reduction.