Design and Implementation of a Data Mesh Architecture for Domain-Oriented Document Classification

Ahmed Bel Hadj Youssef
5th year student at INSA Lyon

Abstract

Large organizations increasingly rely on automated document classification to organize, secure, and exploit their information assets. However, traditional centralized data architectures such as Data Lakes often fail to scale organizationally, especially when documents belong to heterogeneous business domains with distinct ownership, governance, and lifecycle constraints.

This internship report explores the design and implementation of a Data Mesh architecture tailored for enterprise document classification. We analyze the challenges of building a domain-oriented data platform, review existing architectural approaches from both academia and industry, and compare multiple candidate architectures. Based on these findings, we propose a practical architecture combining GitLab for document versioning, Apache Airflow for orchestration, and Hadoop-based storage for domain-specific data nodes. Our approach emphasizes decentralization, traceability, and scalability while remaining feasible in an industrial context.

Keywords

Data Mesh, Document Classification, Data Architecture, Distributed Systems, Enterprise Data Platforms

1 Introduction

As part of the 5IF PhD Track internship at INSA Lyon, this work focuses on the design of a modern data architecture for large-scale document classification in an enterprise context. Document classification is essential for information retrieval, compliance, security, and knowledge management. Enterprises often rely on centralized repositories or Data Lakes to manage documents, but these solutions struggle to scale with organizational complexity and diverse business domains. Centralized architectures can become bottlenecks, suffer from poor data quality oversight, and fail to provide the flexibility required for domain-specific document processing.

To overcome these limitations, this work explores the application of the *Data Mesh* paradigm. Data Mesh shifts the focus from centralized data ownership toward a decentralized, domain-oriented approach. While originally proposed for analytical and structured data, its principles offer promising strategies for managing less structured, document-centric workloads, which involve high variability in formats, complex pre-processing, and strong requirements for traceability and semantic understanding.

The objective of this work is threefold:

- Identify the challenges of building a Data Mesh for enterprise document classification,
- Analyze existing architectural alternatives from literature and industry,
- Propose and justify a concrete, implementable architecture that balances decentralization, traceability, and operational feasibility.

2 State of the Art

In this section, we discuss how data mesh have been introduced after data lakes architectures (Section 2.1), the founding principles of the data mesh architecture (Section 2.2) and how domain-oriented document classification takes place in data meshes (Section 2.3).

2.1 From Data Lakes to Data Mesh

Initially, Data Lakes have been introduced to ease the joint usage of various datasets being produced without coordination, leading to heterogeneous datasets. In turn, Data Lakes focus on storing raw data in a single platform (often based on HDFS – Hadoop Distributed File System) and process it on-the-fly, i.e., when users interact with the lake. Existing Data Lakes are surveyed in [5, 7]. While scalable from an infrastructure perspective, Data Lakes tend to become *data swamps* due to weak ownership, inconsistent governance, and limited discoverability.

Data Mesh, introduced by Dehghani [3], advocates four core principles: domain-oriented data ownership, data as a product, self-serve data infrastructure, and federated computational governance. Recent studies highlight that Data Mesh improves scalability at the organizational level, but introduces new challenges related to interoperability and tooling [1, 4].

2.2 Data Mesh Principles

Data Mesh is founded on four key principles that guide the design of scalable, domain-oriented data architectures [2, 4]:

(1) **Domain-oriented decentralized data ownership and architecture:** Data ownership is shifted to the teams closest to the business domain, making them responsible for data quality, availability, and lifecycle management. This reduces bottlenecks associated with centralized data teams and improves alignment with business needs.
(2) **Data as a product:** Each dataset is treated as a product, with clear interfaces, discoverability, trustworthiness, and documentation. Service-level objectives (SLOs) ensure that data consumers can reliably access and use the data, just as they would interact with any other product.
(3) **Self-serve data infrastructure as a platform:** The infrastructure abstracts away complexity, enabling domain teams to build, deploy, and operate their own data products independently. This principle empowers teams while maintaining consistency across the organization.

(4) **Federated computational governance:** Global policies and standards are applied automatically through the platform to ensure interoperability, security, and regulatory compliance. Governance is enforced computationally rather than manually, allowing decentralization without sacrificing control.

These principles collectively enable organizations to scale their data architectures not just technically, but also organizationally, by promoting autonomy, traceability, and quality while maintaining global standards.

2.3 Domain Partitioning in Data Mesh

While the foundational principles of Data Mesh were primarily developed and validated using highly structured data, such as CSV datasets or relational tables, their application to unstructured textual data remains an open challenge. In structured environments, data can often be mapped to domains based on explicit schema metadata. However, long-form documents lack this inherent structure, requiring a deep "domain-specific semantic interpretation" to determine their appropriate placement within the mesh.

Existing Data Mesh implementations often rely on the manual classification of input data into specific data mesh nodes [3]. This manual approach creates a significant operational bottleneck as the volume of documents grows. To improve scalability, researchers have proposed the automatic and semantic classification of documents into domains. Yet, as noted in recent literature [6], this task is inherently difficult. The complexity arises because document classification for Data Meshes must account for subtle semantic nuances, domain-specific vocabularies, and the overlapping boundaries of business contexts. Bridging the gap between raw text and decentralized domain ownership is the primary motivation for the automated classification approach explored in this internship.

3 Problem Formulation and Approach

We now expose our problem statement (Section 3.1) and the proposed approach (Section 3.2).

3.1 Problem Statement

The core problem addressed within this internship is the following: *how to design a Data Mesh architecture that enables scalable, domain-oriented document classification while preserving traceability, governance, and operational feasibility?*

At first glance, one might think that document classification can be done by directly assigning documents to organizational domains. However, this is more complex in practice. Organizational domains often do not match the way documents are technically stored, as files are spread across different systems that were built over time and not designed around domain responsibilities. In addition, centralized processing pipelines reduce the autonomy of domains: when all transformations are handled in one place, teams depend on a central entity to make changes, which slows down development and limits flexibility. Another difficulty lies in the limited reproducibility of document transformations. Document processing steps such as cleaning, enrichment, or classification are often updated and tested frequently, but these changes are not always well tracked, making it hard to reproduce results or understand how a document was processed. Finally, there is a constant trade-off between experimentation and governance. Teams need freedom to experiment with new models and approaches, while the organization must still ensure rules related to security, compliance, and data quality, which can be difficult to balance.

In this work, we focus mainly on the misalignment between organizational domains and technical storage, and on the problems caused by centralized pipelines.

Data Lake Data Mesh	Hadoop Hadoop	Hadoop PostgreSQL	GitLab Hadoop
Scalability	high	moderate	moderate
Data Volume	high	moderate	high
Versioning	low	low	high
Ease of setup	low	moderate	high
Use-case	large	moderate	moderate

Table 1: Comparison of 3 candidate architectures for our framework.

3.2 Methodology

Our approach consists of:

(1) Proposing a **novel data mesh architecture for automatic domain-oriented document classification** based on the review of multiple architectural candidates compared across technical and organizational criteria, as detailed in Section 4,
(2) Creating the associated **complete software stack** to develop our proposed data mesh architecture, as detailed in Section 5,
(3) Designing a concrete **implementation** for scalabale and flexible document classification pipelines (as explained in Section 6).
(4) **Evaluating our approach** both in terms of architecture and efficiency, as examplified in Section 7.

Finally, we conclude with the scientific contributions of this internship in Section 8 and discuss future directions in terms of research and technical implementation.

4 Global Architecture Design

In order to propose an efficient and domain-oriented document classification data mesh, we have defined and reviewed **3 candidate architectures** based on **5 criteria** (whose values range from small/low to large/high – the higher, the better). In turn, we examined the following architectures: an Hadoop Data Mesh relying on a Hadoop Data Lake (Section 4.1), a PostgreSQL Data Mesh relying on an Hadoop Data Lake (Section 4.2), and a Hadoop Data Mesh on top of a GitLab Data Lake (Section 4.3). Table 1 summarizes our architectures review. Finally, in Section 4.4, we explain our design choices in terms of global architecture.

4.1 Hadoop Data Lake + Hadoop Data Mesh

- **Scalability**: Fully distributed architecture supporting horizontal scaling for both storage and computing. Each domain can be scaled independently using dedicated Hadoop resources.
- **Data Volume**: Designed for very large datasets (GB to TB+), with HDFS providing efficient storage and processing across domains.
- **Versioning**: Relies on HDFS snapshots and lineage tracking, which require explicit configuration and operational overhead.
- **Ease of Setup**: High operational complexity due to multiple Hadoop components, security configuration, and cluster management.
- **Use Case Suitability**: Best suited for industrial-scale big data platforms where performance and scalability outweigh setup and maintenance costs.

4.2 Hadoop Data Lake + PostgreSQL Data Mesh

- **Scalability**: Scales efficiently at the Data Lake level using Hadoop, while domain-level scalability is constrained by PostgreSQL's vertical scaling model.
- **Data Volume**: Supports large volumes in the Data Lake, with domain datasets expected to remain moderate in size for optimal database performance.
- **Versioning**: Partially supported through ETL reproducibility and optional database-level versioning strategies.
- **Ease of Setup**: Moderate complexity, combining a single Hadoop cluster with a widely adopted relational database.
- **Use Case Suitability**: Well adapted to academic projects and small-to-medium systems where domain ownership must be demonstrated with limited infrastructure overhead.

4.3 GitLab Data Lake + Hadoop Data Mesh

- **Scalability**: Domain-specific Hadoop or object storage nodes can be scaled independently, while overall scalability is bounded by GitLab storage limits.
- **Data Volume**: Raw document storage is constrained by GitLab and Git LFS quotas, whereas domain-level datasets stored in Hadoop scale efficiently.
- **Versioning**: Native Git-based versioning ensures fine-grained traceability, reproducibility, and auditability of documents and pipelines.
- **Ease of Setup**: Simple to bootstrap using GitLab's managed tooling, with additional complexity limited to domain-specific Hadoop nodes.
- **Use Case Suitability**: Particularly suitable for document-centric pipelines, i.e., data pipelines where the primary artifacts are documents (CSV files in our case) rather than large tabular datasets or event streams. The pipeline focuses on ingesting, versioning, transforming, and validating documents, with strong traceability and reproducibility, proof-of-concept Data Mesh implementations, and environments requiring strong governance.

4.4 Selected architecture

Finally, we chose to combine a GitLab Data Lake with a Hadoop Data Mesh. For our project, the volume of data in the Data Lake is limited, and old documents can be deleted when no longer needed, so strict persistence and organization are not critical. What matters most is consistency, scalability, and large-volume support at the Data Mesh level, which Hadoop guarantees. In addition, GitLab provides native versioning and traceability, making it easy to manage documents and pipelines, while setup and maintenance remain simple due to GitLab's managed tools. This combination ensures a practical, scalable, and maintainable architecture suited to our document-centric workflows.

5 Complete Software Stack

Based on the general design proposed in Section 4, we now detail the complete software stack we developed for our scalable and flexible data mesh allowing for automatic domain-oriented document classification:

- **GitLab** as the general document Data Lake, ensuring native versioning, immutability, and fine-grained access control,
- **Apache Airflow** for orchestrating ingestion, classification, enrichment, and domain materialization pipelines,
- **Hadoop HDFS / MinIO** as domain-specific Data Mesh nodes, where each node serves as an isolated Data Lake for a single domain, providing:
  - **Hardware-level separation**: each domain node runs on dedicated computing and storage resources, reducing the risk of resource contention,
  - **Security isolation**: ACLs – standing for Access Control List – and network segmentation prevent unauthorized cross-domain access,
  - **Scalability**: nodes can be independently scaled depending on the volume and frequency of document updates.
- **PostgreSQL** as a metadata store for managing domain categories, user roles, and eligibility rules. Each document's classification, domain assignment, and team member permissions are recorded in dedicated schemas to ensure auditable governance.
- **FastAPI Web Framework** used to provide a real-time dashboard for team members, enabling: viewing domain-specific documents, exploring category assignments, filtering datasets by user eligibility, and managing interactive queries on metadata. Authentication is implemented via `Python-JOSE`, supporting JWT-based secure access and role-specific permissions. The platform also allows team members to run Python category classification scripts and handle LLM API calls as services.

This design combines the strengths of a Git-based Data Lake, containerized Hadoop Data Mesh nodes, and relational metadata storage to achieve a flexible, secure, and traceable architecture.

6 Implementation

Based on our general architecture (Section 4) and complete software stack (Section 5), we now detail the practical implementation of our system on the following components: the orchestrator (Section 6.1),

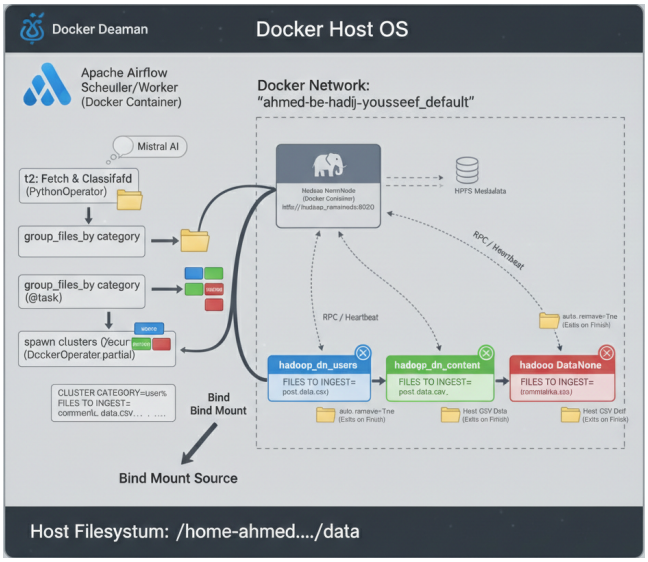


Figure 1: Orchestration via Airflow and Domain Isolation using Dockerized Hadoop Nodes (generated with Gemini).

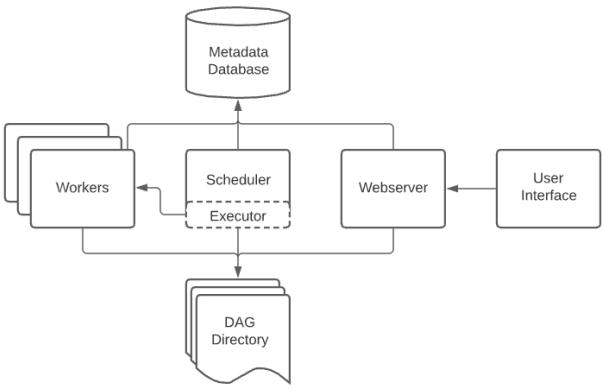


Figure 2: Apache Airflow architecture and interaction between core components

the node creation (Section 6.2), and the automatic domain-oriented document classification (Section 6.3).

Figure 1 illustrates the system architecture, which relies on a Docker bridge network named `ahmed-bel-hadj-youssef_default` to enable communication between a central Hadoop NameNode and dynamically spawn DataNode containers. The orchestration is handled by Apache Airflow, which triggers the host's Docker daemon to deploy DataNode containers on demand. Once active, the DataNodes register with the NameNode using RPC and heartbeat mechanisms over the virtual network, while accessing raw CSV data through direct bind mounts from the host's physical filesystem.

6.1 Apache Airflow Architecture and Orchestration Role

Figure 2 illustrates the core components of the Apache Airflow architecture. Airflow follows a modular and distributed design composed of a *Webserver*, a *Scheduler*, one or more *Workers*, and a centralized *Metadata Database*. Workflows in Airflow, referred to as Directed Acyclic Graphs (DAGs), define sequences of tasks to be executed, including data ingestion, transformation, validation, and loading operations. These workflows are stored in a shared DAG directory and executed according to their scheduling and dependency rules.

The **Scheduler** is responsible for parsing DAG definitions and determining task execution order based on dependencies and time constraints. Once tasks are ready to run, they are dispatched to available **Workers** through the **Executor**. Task execution state, logs, and metadata are persisted in the **Metadata Database**, ensuring fault tolerance and reproducibility. The **Webserver** provides a user interface for monitoring pipeline execution, inspecting task states, and managing operational workflows.

Apache Airflow plays a central role in our architecture by orchestrating interactions between heterogeneous software components, including GitLab, Hadoop-based Data Mesh nodes, PostgreSQL metadata services, and containerized domain environments. Given the multi-step and interdependent nature of document ingestion, classification, enrichment, and distribution across domain-specific nodes, a robust workflow orchestration layer is required to ensure consistency, traceability, and operational reliability.

In our implementation, Airflow DAGs are defined using Python and leverage multiple operators adapted to the needs of the project:

- **PythonOperator** is used to execute custom Python scripts responsible for document classification, metadata extraction, and enrichment logic.
- **SQLExecuteQueryOperator** is employed to initialize and update PostgreSQL metadata tables, including category definitions, user-role mappings, and eligibility rules.
- **DockerOperator** is used to dynamically instantiate isolated containers representing domain-specific Data Mesh nodes. Each container corresponds to a category and enables secure, hardware-level isolation between domains while supporting horizontal scalability.

This orchestration strategy enables a clear separation of concerns between data ingestion, metadata management, processing, and storage. Furthermore, by dynamically provisioning containers per category, the system aligns with Data Mesh principles by treating each Hadoop-based domain node as an autonomous data product, benefiting from independent scaling, enhanced security isolation, and reduced resource contention.

Overall, Apache Airflow provides the backbone for coordinating distributed components in a reproducible and observable manner, making it a key enabler of the proposed hybrid Data Lake and Data Mesh architecture.

6.2 Dynamic vs. Static Data Node Creation

There are two main approaches to creation nodes in the Data Mesh: either statically or dynamically. We compare these two approaches in Table 2. Based on this comparison, we ultimately chose dynamic

Feature	Static	Dynamic
Provisioning	Pre-created nodes	On-demand creation
Scalability	Fixed, limited	Flexible, elastic
Resource Use	Can be idle	Efficient, allocates as needed
Complexity	Simple setup	Requires orchestration
Cost	Potentially higher	Potentially lower
Latency	Always available	Slight delay at creation

Table 2: Comparison of static vs. dynamic Data Node creation.

Data Node creation based on document categories. Since the set of categories encountered in input documents is not known in advance, static provisioning would lead to unnecessary resource allocation. Although dynamic node creation is more complex to configure and orchestrate, it provides major cost benefits and allows the system to adapt automatically to newly discovered categories by allocating compute and storage resources only when required.

6.3 Automatic domain-oriented document classification

6.3.1 Keyword-Based Classification. The first method we implemented for automatic domain-oriented document classification is keyword-based classification. In this approach, we prepare a predefined dataset where each key corresponds to a document category and the associated values represent relevant column names for that category. To classify a new document, we use a brute-force algorithm that searches for exact matches between the document's columns and the keywords in each category. Each category is then assigned a score based on the number and relevance of matching keywords, and the document is ultimately assigned to the category with the highest score. This method is simple, transparent, and efficient for datasets with well-defined columns and known categories. However, its reliance on exact keyword matches limits its ability to generalize, making it less effective for documents with unstructured content or for categories that were not predefined.

6.3.2 Semantic Similarity-Based Classification. A second approach improves upon the keyword-based method by incorporating semantic similarity using the *Word2Vec* library. Instead of relying on hard keyword matching, this method computes similarity scores between document content and category keyword vectors. While slightly more computationally intensive, this approach remains cost-efficient and achieves higher classification accuracy by capturing semantic relationships between words.

Although these methods are effective for classification into **predefined categories**, they are inherently limited in dynamic environments. Since categories must be known in advance, these approaches are not suitable for scenarios where new categories — and consequently new Data Nodes — must be created dynamically.

6.3.3 LLM-Based Dynamic Category Detection. To support dynamic category creation, the use of a **Large Language Model (LLM)** is required. LLMs are capable of identifying previously unseen categories by reasoning over document content, making them well-suited for dynamic Data Node provisioning.

For reasons of **European digital sovereignty**, the Mistral model was selected. Minor differences in accuracy compared to alternative LLMs were not considered critical for this use case.

6.3.4 Mistral-Based Dynamic Category Detection. In our implementation, the Mistral model is used to infer the most relevant category for each CSV document based on its column headers. For each file, we read the column names and construct a prompt instructing Mistral to select a single category describing the dataset. The prompt also includes the list of previously used categories, with the instruction to reuse an existing category if the new document is similar, or create a new one only if necessary. By setting a low temperature (0.2) in the API call, we ensure deterministic and consistent classification. This approach allows the system to dynamically expand its set of categories while maintaining coherence with already existing Data Nodes, enabling efficient orchestration of dynamic Data Node creation in the Data Mesh.

6.3.5 Metadata-Guided Classification Strategy. To optimize both cost and performance, identified categories are stored in a **PostgreSQL metadata database**. At each pipeline execution, the list of existing categories is injected into the LLM prompt. The model is then instructed to either:

- assign the document to one of the existing categories, or
- propose a new category when the document content clearly does not fit any known category.

This hybrid strategy reduces unnecessary category proliferation, improves consistency across pipeline runs, and limits LLM usage costs while preserving the ability to dynamically discover new domains.

7 Evaluation of the proposed approach

The proposed architecture (recall Sections 5 and 6) is available on GitLab at https://gitlab.insa-lyon.fr/5if/ahmed-bel-hadj-youssef. It was evaluated based on its ability to: (*i*) implement a decentralized Data Mesh using containerized infrastructure, as assessed in Section 7.1, and (*ii*) automatically classify documents based on their organizational domain, as examplified in Section 7.2.

7.1 Architecture Evaluation

The architecture was assessed along the following dimensions:

- **Pipeline Reproducibility**: Airflow orchestration and Python-based operators ensure every data lifecycle stage—from fetching to classification—is logged and reproducible using historical document versions.
- **Domain Autonomy**: By deploying independent Hadoop DataNode containers (e.g., `hadoop_dn_users`), each domain functions as a self-contained lake. This enables isolated scaling and clear ownership of data assets.
- **Data Persistence**: The use of Docker Bind Mounts ensures high-performance I/O and data durability, mapping the host filesystem directly to the domain containers to prevent data loss during container lifecycles.
- **Scalability and Security**: The dedicated Docker network provides logical segmentation. New domains can be added horizontally as isolated nodes, while JWT-based authentication restricts access to specific containerized environments.

Overall, combining a Git-based Data Lake with Hadoop-driven domain isolation provides traceability and operational efficiency. However, challenges remain in managing GitLab storage quotas and minimizing synchronization latency between the distributed DataNodes and the central metadata store during high-concurrency operations.

7.2 Domain-oriented classification

As an example of domain-oriented classification in our Mistral-based document classification *Datamesh* project, we considered the ingestion of a CSV file (`chatgpt-reddit-comments.csv`) containing the following columns: `comment_id`, `comment_parent_id`, `comment_body`, and `subreddit`.

After processing the document, the classification model assigns it to the `"community"` category. The system then checks whether a data node corresponding to the `"community"` category already exists in the Datamesh. If such a node exists, the document is attached to it; otherwise, a new data node named `"community"` is automatically created, and the document is stored within it.

8 Conclusion

We now conclude this report with a focus on the scientific contributions (Section 8.1), a discussion on future works (Section 8.2), as well as the lessons learned during this internship (Section 8.3).

8.1 Scientific Contributions

This work demonstrates that Data Mesh principles can be effectively applied to document-centric workloads, bridging the gap between theoretical concepts and industrial practice. We proposed a hybrid architecture combining GitLab-based document versioning, Apache Airflow orchestration, and containerized Hadoop Data Mesh nodes, which enables scalable, traceable, and secure management of domain-specific documents. A key innovation is the dynamic creation of Data Nodes based on document categories, allowing the system to adapt to previously unseen categories while minimizing resource costs. Furthermore, by integrating metadata-guided LLM-based classification, the architecture supports hybrid automated and rule-based categorization, improving both flexibility and classification accuracy. The proposed approach also emphasizes reproducibility and operational transparency, providing a solid foundation for benchmarking and further optimization in enterprise environments.

8.2 Future work

As a future improvement, we plan to build a small labeled dataset composed of approximately **50 representative documents**, manually annotated with their correct categories. This dataset will serve as a reference benchmark for evaluating classification performance.

A second improvement would be to propose an evaluation function which automatically computes accuracy metrics by comparing predicted categories against the known ground truth. This will allow objective comparison between different classification approaches and parameter configurations. Such a metric-driven evaluation framework will be particularly valuable when fine-tuning the LLM (supported by Mistral) or iterating on prompt engineering strategies. Having a stable evaluation baseline ensures that any model improvement or architectural change can be quantitatively assessed rather than relying solely on qualitative judgment.

In addition, we plan to evaluate the **trade-offs between local LLM deployment and API-based inference** using Mistral. Running the LLM locally provides full control over data and inference and may become cost-effective at scale with a high number of users or classification requests. However, it requires significant infrastructure, maintenance, and operational overhead. In contrast, using the Mistral API offers a lightweight, low-maintenance solution suitable for prototyping and small to medium workloads. As the number of users or classification requests grows, a careful cost-benefit analysis will be needed to determine the threshold at which transitioning from API-based inference to a local deployment becomes advantageous.

8.3 Internship Insights and Lessons Learned

From an internship perspective, this project offered valuable exposure to the practical challenges of designing and deploying a distributed, domain-oriented data platform. It highlighted the trade-offs between flexibility and operational complexity, particularly when dynamically provisioning Data Nodes and integrating LLM-based classification. The work also underscored the importance of cost efficiency, scalability, and security in containerized infrastructures, as well as the critical role of metadata and versioning for governance and traceability. Beyond technical insights, the project reinforced the need to balance theoretical paradigms, such as Data Mesh, with real-world constraints in enterprise environments. Overall, the internship provided a comprehensive understanding of how emerging data architectures can be operationalized for large-scale, document-centric workflows, offering lessons that are relevant for both future research and professional practice.

Acknowledgments. Finally, I would like to thank Nelly Barret, assistant professor at INSA Lyon and the LIRIS lab, for this opportunity. In particular, I appreciated her availability and advices on research and implementation questions.

For this report, I used Gemini to generate Figure 1 and to help rewrite certain paragraphs with improved grammar and fluency. Additionally, I used AI as an initial step to verify some information during the writing process.

References

[1] Ivo Blohm, Felix Wortmann, Christine Legner, and Felix Köbler. Data products, data mesh, and data fabric: New paradigm (s) for data and analytics? *Business & Information Systems Engineering*, 66(5):643–652, 2024.

[2] Guillaume Bodet. *The Practical Guide to Data Mesh.* Zeenea, 2024.

[3] Zhamak Dehghani. *Data Mesh.* Marcombo, 2022.

[4] Abel Goedegebuure, Indika Kumara, Stefan Driessen, Willem-Jan Van Den Heuvel, Geert Monsieur, Damian Andrew Tamburri, and Dario Di Nucci. Data mesh: a systematic gray literature review. *ACM Computing Surveys*, 57(1):1–36, 2024.

[5] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12571–12590, 2023.

[6] Michalis Pingos and Andreas S. Andreou. Discovering data domains and products in data meshes using semantic blueprints. *Technologies*, 12(7), 2024.

[7] Franck Ravat and Yan Zhao. Data lakes: Trends and perspectives. In *International Conference on Database and Expert Systems Applications*, pages 304–313. Springer, 2019.