Big Data Research 43 (2026) 100574

Contents lists available at ScienceDirect

Big Data Research

journal homepage: www.elsevier.com/locate/bdr

MIND: A metadata-driven INgestion design pattern for efficient data ingestion

Chiara Rucco *, Antonella Longo , Motaz Saad

Department of Engineering Innovation, University of Salento, Lecce, 73100, Italy

ARTICLE INFO

Keywords:
Design patterns
Ingestion
Cloud computing

ABSTRACT

Data ingestion plays a crucial role in enterprise data management, particularly when dealing with large-scale datasets. This paper introduces the Metadata-driven INgestion Design pattern (MIND), a flexible, metadata-driven design pattern for cloud-based big data management. MIND enhances adaptability by enabling dynamic adjustments to ingestion types, schema updates, table additions, and the integration of new data sources.

Validated on the Azure cloud platform, MIND demonstrates scalability, feasibility, and efficiency in reducing ingestion time and operational complexity. By relying on metadata for pipeline orchestration, MIND offers a robust solution to the challenges of high-volume data processing, providing a more agile and maintainable approach to data workflows. This work contributes to the evolution of metadata-driven architectures and offers a foundation for future advancements in data management technologies.

1. Introduction

Data engineering involves designing and building pipelines to collect, store, and analyze large-scale data, ensuring accessibility, reliability, and readiness for analysis [1]. Design patterns are crucial in developing scalable and resilient data architectures that can handle the large data volumes generated by modern organizations.

Data Engineering Patterns (DEPs) are standardized methods in data engineering that include ETL processes, data pipelines, and data stream management [2]. Data Engineering Design Patterns (DEDP) are best-practice solutions to common challenges using optimized and tested approaches [2].

Several scholars and data engineers (practitioners) have suggested methods for efficiently ingesting large volumes of data [3–5]; however, a gap persists for certain ingestion scenarios that involve multiple data sources with varying ingestion needs (full or incremental ingestion). Current design patterns struggle to manage the complexities of high-volume data ingestion. They lack the flexibility to accommodate diverse data sources, frequent schema changes, and varying ingestion requirements. This forces data engineers to implement multiple pipelines, some for periodic full data loads and others for incremental or real-time updates, leading to inefficiencies and added complexity. In this work, we consider ingestion scenarios involving datasets that scale from 10^6 one million records to over 10^9 one billion records, often accompanied by frequent schema modifications and diverse ingestion requirements (full, incremental, or hybrid). These conditions exceed the capabilities of conventional ETL or pipeline designs and motivate the need for a metadata-driven ingestion pattern such as MIND.

In this paper, we propose a new design pattern for big data ingestion in cloud platforms. Our proposed solution is a metadata-driven, cloud-neutral design pattern that streamlines ingestion by combining full and incremental processes into a single, cohesive pipeline that adapts seamlessly to varying needs.

We introduce the **MIND** (Metadata-driven INgestion Design pattern) as a solution. The MIND design pattern proposes the use of a unified metadata table where data engineers store the ingestion configuration, along with a procedure that reads the configuration from the metadata table, validates it, and then ingests the data according to the defined ingestion needs specified in the metadata table for each data source. This approach enhances the feasibility of data ingestion processes, allowing seamless changes to ingestion types, schema modifications, table additions, and the integration of new sources with minimal effort. The proposed MIND ensures dynamic changes without extensive manual intervention, reducing data ingestion time.

The heart of the MIND design pattern is a Unified Metadata Table (UMT), storing configurations for each data source, including credentials, ingestion type, change detection methods. The design is Cloud-Agnostic, tested on Azure, and integrates seamlessly with orchestrators like Azure Data Factory and Google Cloud Composer, ensuring portability and flexibility. The MIND pattern is cloud-neutral ingestion framework. Our primary experimental validation is carried out on Microsoft Azure. Additionally, lightweight tests on Google Cloud was conducted in

* Corresponding author.
E-mail addresses: chiara.rucco@unisalento.it (C. Rucco), antonella.longo@unisalento.it (A. Longo), motazk.saad@unisalento.it (M. Saad).

https://doi.org/10.1016/j.bdr.2025.100574
Received 22 March 2025; Received in revised form 25 August 2025; Accepted 23 November 2025
Available online 27 November 2025

Table 1
Comparison of data architecture approaches.

Architecture	Primary Focus	Metadata-Driven	Use Cases
MIND (our work)	Efficient Data Ingestion	Yes	General Data Ingestion
Lambda	Batch & Real-Time Processing	No	Requiring Both Batch & Real-Time Views
Kappa	Real-Time Processing	No	Real-Time Analytics, IoT
Data Mesh	Decentralized Data Ownership	Metadata is Important for Data Products	Organizations with Diverse Data Needs
Data Vault	Historical Data Management	Metadata is Key for Auditability	Data Warehousing, Auditing

another work [6], which demonstrated that the metadata-driven orchestration principles underlying MIND can be effectively replicated outside Azure. These results indicate that MIND is not tied to a specific provider but represents a generalizable ingestion approach adaptable to multiple cloud platforms.

Key advantages of this approach include adaptability, optimization, and ease of use. Adaptability automatically adjusts ingestion techniques based on the characteristics of the data source. Optimization focuses resources on processing only essential data. Ease of use simplifies pipeline management with centralized configurations.

The remainder of this paper is organized as follows. Section 2 reviews related work on data engineering design patterns in cloud environments. Section 3 introduces the MIND design pattern and details the Unified Metadata Table. Section 4 describes the validation methodology, datasets, and benchmarking experiments. Section 5 presents the results and their interpretation, while Section 6 outlines future work and potential extensions.

2. Related works

This section reviews related works on data engineering design patterns, cloud-based data engineering design patterns, and data ingestion in cloud architectures.

Research has highlighted the importance of well-structured data ingestion pipelines to manage the complexities of handling massive data volumes from various sources. Key basic patterns in this domain include ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform). More advanced patterns include Lambda Architecture, Kappa Architecture, Data Mesh, and Data Vault [3,7–9]. In domains such as the Internet of Things, significant contributions have addressed the challenge of interoperability among heterogeneous platforms, devices, and protocols. For example, Badii et al. [10] propose taxonomies and highlight open challenges regarding interoperability in IoT, focusing on enabling communication and collaboration across diverse systems. However, these studies primarily target communication standards and semantic integration, and do not directly address the operational or architectural complexities associated with dynamic, large-scale data ingestion pipelines in cloud environments.

Turning to existing Data Engineering design patterns in the literature, we see that our Design Pattern focuses on efficient data ingestion, while Lambda [7] emphasizes batch and real-time processing, Kappa [8] targets real-time analytics, Data Mesh [11] supports decentralized ownership, and Data Vault specializes in historical data management. We prioritize a metadata-driven approach, unlike Lambda and Kappa. Metadata plays a role in Data Mesh for product management and is key to auditability in Data Vault. Our solution supports general data ingestion, making it versatile, while others cater to specific needs like analytics, IoT, or compliance. Table 1 summarizes the comparison between our work and different DEDPs in the literature. Although various Data Engineering Patterns (DEPs) and Data Engineering Design Patterns (DEDs) have been proposed, including concepts like data mesh and lambda architecture, significant gaps still exist in addressing the challenge of high-volume data ingestion from diverse sources that require different processing techniques. The rapidly evolving field of cloud-based data engineering has seen substantial contributions, yet the complexities of scaling data pipelines across varied environments and integrating heterogeneous data sources remain underexplored.

While Lambda and Kappa architectures focus on integrating batch and real-time processing, and Data Mesh emphasizes decentralized ownership of data products, MIND specifically targets the ingestion bottleneck by centralizing configuration in metadata. Unlike Lambda, which requires maintaining parallel batch and streaming layers, MIND unifies ingestion methods (full, incremental, hybrid) in a single pipeline that dynamically adapts at runtime. Similarly, while Data Mesh relies on organizational and governance principles for scalability, it does not prescribe ingestion mechanisms; MIND complements such paradigms by offering a concrete, metadata-driven ingestion layer that can operate within or across mesh-based environments. In this sense, MIND advances beyond existing design patterns by simplifying ingestion complexity while remaining compatible with higher-level architectures.

Major cloud providers such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP) have developed comprehensive data engineering design patterns tailored to their platforms. These patterns enable businesses to build scalable, secure, and efficient data pipelines [12–14]. Our approach differs from major cloud providers like Azure, AWS, and Google Cloud, which focus on cloud-specific patterns, data-driven architectures, and industry-specific analytics solutions, respectively. Key advantages of the MIND Design Pattern include its hybrid ingestion model, cloud-agnostic nature, and comprehensive metadata usage, setting it apart from the cloud providers' solutions. Table 2 summarizes the comparison between our work and our work with to DEDPs offered by Azure, AWS, GCP.

On the other hand, several researchers used the concept of meta-data to address streamline data ingestion processes Vattumilli [15], to maintaining consistency across data pipelines [16], and to build flexiable and scalable data pipelines on azure platforms [17].

A significant contribution from recent studies, such as the one by Vattumilli [15], highlights the promise of metadata-driven approaches to streamline data ingestion processes. By abstracting much of the complexity in data processing, metadata-driven pipelines enable faster integration and facilitate easier modifications, providing flexibility to adapt to changing business requirements. Despite offering a high-level theoretical framework for metadata management, Vattumilli's work falls short in providing practical examples and implementation strategies, leaving key gaps in addressing real-world data engineering challenges.

In contrast, a recent Databricks guide [16] provides more practical insights into implementing metadata-driven ingestion by emphasizing the use of control tables. These tables are pivotal in managing metadata, maintaining consistency across data pipelines, tracking state changes, and supporting schema evolution. While Databricks' approach is hands-on, it has limitations. The use of control tables is specific to the Databricks platform, which restricts its applicability to organizations using non-Databricks environments. Furthermore, the guide does not comprehensively address the complexities involved in scaling metadata-driven pipelines across multiple cloud environments or integrating them with diverse, heterogeneous data sources.

Similarly, Rahul Gosavi's blog [17] on building a metadata-driven framework in Azure Data Factory further contributes to the development of flexible and scalable data pipelines. Gosavi emphasizes how Azure Data Factory's metadata-driven approach can decouple pipeline logic from data sources and destinations, enabling

Table 2
Comparing our work with to DEDPs offered by Azure, AWS, GCP.

Feature	Primary Focus	Ingestion Methods	Cloud-Agnostic	Metadata-Driven
MIND (our work)	Metadata driven, cloud-agnostic data ingestion	Hybrid (full and incremental)	Yes	Yes
Azure	Cloud-specific data management patterns	Varies based on pattern (e.g., Event Sourcing, Materialized View)	No (specific to Azure)	Varies based on pattern
AWS	Data-driven architectural patterns	Varies based on pattern (e.g., Serverless Data Lake, Kappa Architecture)	No (specific to AWS)	Varies based on pattern
GCP	Industry-specific data analytics solutions	Varies based on solution (e.g., Dataflow-based ETL, BigQuery Lambda Architecture)	No (specific to GCP)	Varies based on solution

dynamic pipeline execution by parameterizing linked services, datasets, and pipelines. While this work offers a practical approach to flexible data integration in Azure, it does not fully address the challenge of scaling such frameworks across multiple cloud environments or handling the complexities of integrating a broad range of heterogeneous data sources. Additionally, the focus on Azure-specific tools may limit the applicability of this framework for organizations using different or hybrid cloud providers.

Furthermore, metadata-driven ingestion is particularly relevant for healthcare applications, where data must be continuously ingested from heterogeneous sources such as IoT devices, patient monitoring systems, and electronic health records. For instance, the work of Dutta et al. [18] highlights the challenges of integrating high-frequency, multi-source data in real time. Our proposal directly addresses these challenges by centralizing ingestion logic in metadata, ensuring adaptability to schema evolution, and reducing manual interventions, which are critical in healthcare environments that demand reliability and compliance.

Beyond traditional data engineering domains, metadata-driven design patterns are also highly relevant in sustainability-focused applications. For example, Data-Driven Sustainability: Leveraging Big Data and Machine Learning to Build a Greener Future [19] demonstrates how scalable data and machine learning architectures can drive efficiency and environmental benefits. However, such approaches often assume that ingestion bottlenecks are already solved, overlooking the challenges of integrating heterogeneous, high-volume datasets from multiple domains (e.g., energy, climate, and transportation). Despite these advances, there remains a clear gap: existing metadata-driven frameworks are often tied to specific cloud providers and lack a unified mechanism to combine incremental, full, and hybrid ingestion methods within a single pipeline. MIND contributes to filling this gap by proposing a cloud-agnostic, metadata-driven ingestion pattern that enables scalability, adaptability, and portability across heterogeneous environments. The next section presents the MIND architecture and explains how it dynamically switches between ingestion methods according to metadata configurations.

3. MIND: Metadata-driven INgestion design pattern

In this section, we introduce MIND (Metadata-driven INgestion Design pattern), a cloud-agnostic design pattern for a robust and scalable ingestion. MIND addresses the challenges of efficiently ingesting and integrating data from multiple sources, such as Oracle databases, Excel files, SQL databases, APIs, and other data models. The MIND leverages metadata as the core mechanism for defining and orchestrating data ingestion processes, ensuring scalability, flexibility, and ease of maintenance.

The main concept of MIND is to use a metadata table, which is a central repository that stores configuration details about the ingestion method for each data source. These configurations include:

Table 3
Unified Metadata Table (UMT) (Mapping Table).

Data Source	Table	Credentials	Primary Key	Watermark	Ingestion Type
Src1	Table1	****	PK1	WM1	Full
Src2	Table1	****	PK2	WM2	Incremental
Src2	Table2	****	PK3		Incremental

- Data source details (e.g. name, credentials)
- Ingestion type (full or incremental)
- Schema updates

Also, our solution includes an implementation of a procedure (implemented in python) that reads these configurations from the meta-data table and run the ingestion pipeline accordingly [20].

In this proposed design pattern, a metadata-driven approach serves as the foundation for managing and orchestrating the data ingestion process. Using a mapping table stored in an accessible source for the specific use case, such as a SQL database, we can define not only the basic parameters for each table to be ingested but also additional attributes to further optimize the process.

3.1. A basic example of metadata table

An example of basic metadata table can be found in Table 3. Here we include some parameters that will help the orchestrator chosen in the specific use case to connect to data sources through the authentication method provided, read the tables specified, and initiate the ingestion process based on the metadata defined (ingestion type, watermark, primary key).

This table can be extended with additional parameters for granular control over the ingestion workflow. The metadata table can include:

- Schema Definition: Specifies table schemas (e.g., columns, data types), enabling validation of incoming data and capturing schema changes (e.g., new columns) to trigger schema evolution in the pipeline.
- Refresh Schedule: Defines ingestion frequency (daily, weekly, on-demand), optimizing resource utilization by scheduling runs only when needed.
- Data Partitioning: Includes criteria (e.g., date, region) to enable ingestion of large datasets in smaller chunks, improving performance and scalability.
- Source Type: Classifies sources (e.g., transactional, log, batch) to apply specific processing rules, enhancing speed and accuracy.
- Data Transformation Rules: Lists transformations (e.g., cleansing, enrichment) applied during ingestion, avoiding hardcoding in pipeline logic.
- Ingestion Status and Error Logging: Tracks ingestion progress (e.g., last successful run, records ingested) and logs errors for troubleshooting.

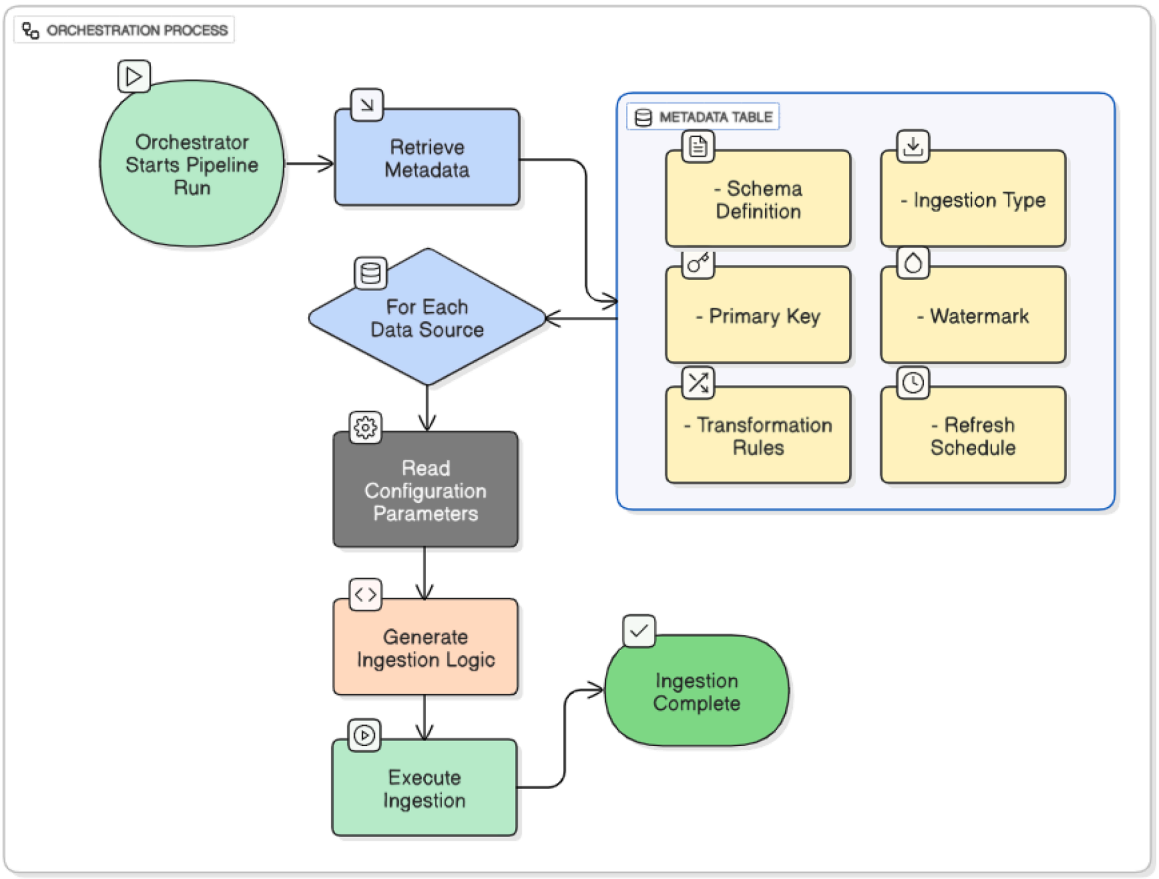

Fig. 1. Unified Metadata Table (UMT) and orchestration flowchart.

- Data Source Versioning: Tracks source version history to maintain compatibility and manage schema or data changes effectively.

To better illustrate the functionality of the Unified Metadata Table, Fig. 1 shows its structure and its interaction with the orchestration process. Each row of the metadata table corresponds to a single data source and contains configuration parameters such as ingestion type, primary key, watermark, schema definition, transformation rules, and refresh schedule. The orchestrator retrieves this metadata at runtime, dynamically generating the ingestion logic without requiring changes to pipeline code. This separation of configuration from execution allows MIND to adapt quickly to new sources, schema changes, or ingestion requirements.

Centralizing parameters in the metadata table makes pipelines configurable and adaptable to diverse use cases. The orchestrator retrieves parameters from metadata, enabling a single, reusable pipeline for multiple sources and ingestion types. For example, specifying "incremental ingestion" activates only relevant logic, while custom requirements (e.g., unique schema or transformations) are handled without modifying pipeline code.

This metadata-driven approach optimizes pipelines by dynamically adjusting based on parameters like ingestion methods, refresh schedules, and transformation rules. The pipeline operates efficiently, processes data accurately, and adapts to changes seamlessly, minimizing manual effort.

MIND leverages metadata to simplify ingestion, improving flexibility and scalability. Centralized configuration in metadata reduces manual intervention, allowing engineers to focus on optimizing data processing and analysis instead of managing complex pipelines.

3.2. Additional examples of metadata tables

Here, we propose additional examples of metadata tables, along with potential use cases and scenarios where they can enhance the efficiency and flexibility of data pipelines. These examples demonstrate how metadata-driven designs can address common challenges in data engineering workflows.

3.2.1. Schema definition, versioning, and partitioning

To handle schema changes and optimize ingestion for large datasets, the metadata table tracks both schema versions and partitioning strategies. The schema information ensures that the pipeline adapts to structural changes, such as new columns or modified data types, while partitioning details enable efficient data handling based on partition columns and frequencies Table 4.

Example:

Use Case: When a new column is added or partitioning is applied, such as partitioning by DateColumn for monthly data ingestion, the pipeline will automatically update its logic based on the schema version and partitioning details.

3.2.2. Transformation rules and error logging

The metadata table can define transformation rules (e.g., cleaning, filtering) and track the status and errors of the ingestion. Transformation rules, such as applying a data cleansing script, are automatically referenced during ingestion. In addition, error logs and ingestion statuses provide visibility into the health of the pipeline and allow quick resolution of issues Table 5.

Table 4
Schema and partitioning metadata.

Data Source	Table	Schema Definition	Schema Version	Partition Column
Src1	Table1	Column1(int), Column2(varchar)	v1	DateColumn
Src2	Table1	Column1(int), Column2(varchar), Column3(date)	v2	Region

Table 5
Transformation and error logging metadata.

Data Source	Table	Transformation Rule	Last Ingested	Error Log
Src1	Table1	CleanDataScript.sql	2025-01-18	No Errors
Src2	Table2	FilterOutInactive.sql	2025-01-17	Timeout Error

Example:
Use Case: For example, Source1's Table1 uses a data cleansing script, and Source2's Table2 logs an error if ingestion fails due to a timeout. The pipeline automatically references the transformation scripts and updates the error log for troubleshooting.

By consolidating schema versioning, partitioning, transformation rules, and error handling into the metadata table, the pipeline becomes adaptable and efficient, capable of scaling with minimal manual intervention.

Table 6 summarizes meta-data table examples and their use cases.

4. Validation, testing, and benchmarking meta-data tables

Metadata-driven ingestion pipelines have emerged as a paradigm shift in addressing state-of-the-art challenges by decoupling pipeline logic from implementation details. Using metadata as a dynamic and centralized configuration layer, these pipelines enable scalable, adaptable, and reusable workflows that align with the principles of modern data architectures. This section describes how we validate, test, and benchmark the proposed Design Pattern Fig. 2.

Our validation follows a systematic approach structured in three phases:

- Metadata Table Validation
- Pipeline Behavior Validation
- Performance Benchmarking

4.1. Metadata table validation

As discussed in the previous section, metadata serves as the foundation for driving the entire ingestion process. However, its accuracy and consistency are critical, as errors in the metadata can propagate through the pipeline, leading to ingestion failures, incorrect transformations, or inconsistencies in the final output. Rigorous validation processes are therefore essential to ensure that metadata remain a reliable and accurate guide for pipeline behavior. With validated metadata, pipelines can dynamically adapt to various scenarios, handle schema changes seamlessly, and execute transformations efficiently while preserving the reliability of the entire system. Although the validation testbed was implemented on Azure, the MIND pattern does not rely on Azure-specific components. The Unified Metadata Table and orchestration logic can be ported to equivalent services on other providers (e.g., AWS Glue, GCP Dataflow, Databricks on multi-cloud).

Table 6
Metadata table examples.

Metadata Table Example	Use Cases
Mapping Table	Stores basic parameters for each table to be ingested, including data source, table name, credentials, primary key, watermark, and ingestion type. Enables the orchestrator to connect to data sources, read specified tables, and initiate the ingestion process based on the defined metadata.
Schema Definition, Versioning, and Partitioning	Handles schema changes and optimizes ingestion for large datasets. Tracks schema versions and partitioning strategies, allowing the pipeline to adapt to structural changes (e.g., new columns) and efficiently handle data based on partitioning criteria.
Transformation Rules and Error Logging	Defines transformation rules (e.g., cleaning, filtering) and tracks ingestion status and errors. Transformation rules are automatically applied during ingestion, and error logs provide visibility into pipeline health for troubleshooting.

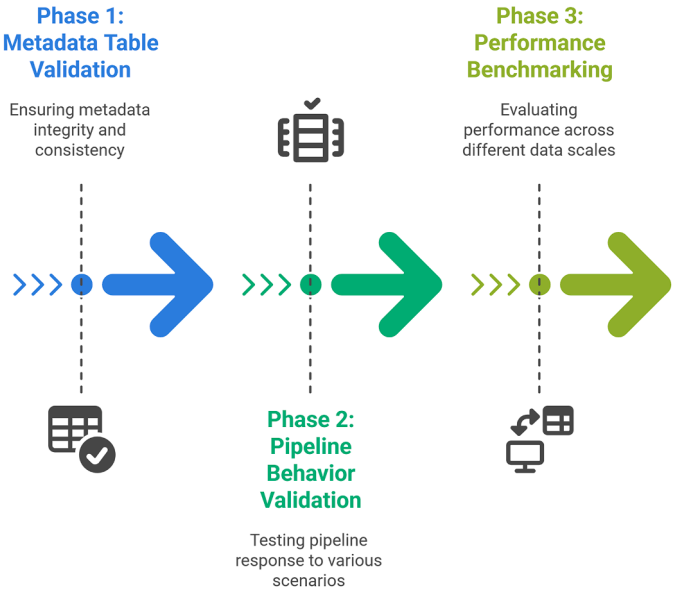

Fig. 2. Validation steps for MIND design pattern.

To ensure the metadata tables meet the necessary standards, we constructed a comprehensive testbed containing all the essential parameters needed to guarantee the functionality and adaptability of the pipelines. The metadata table includes the following items:

- **Table Name**: The name of the table being ingested or processed.
- **Data Source Name**: The name of the source system or file from which data is being ingested (e.g., MySQL, S3, API).
- **Data Source Credentials**: Credentials required to access the data source, such as usernames, passwords, tokens, or keys.
- **Primary Key**: The unique identifier for records in the dataset (e.g., ID column).
- **Watermark**: A column used for incremental ingestion to track the last processed record (usually a timestamp).
- **Refresh Schedule**: Defines how frequently the data is ingested (e.g., hourly, daily, weekly).
- **Partitioning**: Specifies how the data is partitioned (e.g., by date, region, or other categories).
- **Schema**: The table schema, including column names and data types.
- **Data Transformation Script**: A reference to a script that applies transformations to the raw data (e.g., transformations.py).
- **Ingestion Type**: Specifies the type of ingestion (full, incremental, delta).
- **Last Ingestion Date**: The timestamp of the last successful data ingestion.

- **Error Log**: A field to capture errors encountered during ingestion or transformation.
- **Incremental Ingestion Type**: Specifies the method used for incremental ingestion (date-based, hash-based).
- **Data Quality Rules**: Rules for data validation and cleaning, such as removing duplicates or handling missing values.
- **Transformation Rules**: Specific transformation logic or formulas used during data processing (e.g., currency conversion or unit conversions).
- **Batch Size**: Defines the size of the data chunks to be processed in each pipeline run.
- **Concurrency Level**: The number of parallel processes to use during ingestion, which can affect performance.
- **Retry Policy**: Defines the rules for retrying failed ingestion attempts, such as the number of retries and backoff time.
- **Last Error Timestamp**: The timestamp of the last failure event in the ingestion process.
- **Status**: Indicates the current status of the pipeline (e.g., active, paused, failed).
- **Data Validation Script**: A reference to a script for validating data quality (e.g., validate_data.py).
- **Data Lineage**: A reference to previous transformations or processes that led to the current data (used to track the data pipeline flow).
- **Source File Format**: The format of the incoming data (e.g., CSV, JSON, Parquet).
- **Compression Type**: Specifies if the data is compressed (e.g., gzip, snappy).
- **Column-Level Encryption**: Specifies if any columns are encrypted and the encryption method used.
- **Schema Evolution**: Specifies how changes in the source schema (such as column additions) should be handled.
- **Historical Data Retention Policy**: Defines how long historical data should be retained in the pipeline (e.g., 30 days, 6 months).
- **Source System Version**: The version of the data source schema or system.
- **Data Validation Rules**: Validation logic to ensure the data meets expected standards, such as range checks or uniqueness checks.
- **Audit Logs**: Logs tracking the changes and activities within the pipeline.
- **Notification Rules**: Specifies how notifications (e.g., Email or Slack) should be sent in case of errors or successful ingestion.
- **Source Record Count**: The number of records in the source dataset, which helps track data volume for monitoring.
- **Target System**: The target system or database where the data is being ingested (e.g., Data Warehouse, NoSQL Database).
- **Data Sync Type**: Defines whether the data sync is real-time, batch, or near real-time.
- **Data Validation Frequency**: Defines how often the data validation process is triggered (e.g., on ingestion, daily).
- **Historical Data Handling**: Defines how old data should be handled, such as whether to overwrite or merge data during ingestion.

In order to fully validate the consistency and completeness of our metadata table, we developed a python script for validation with some basic steps described in the following subsections. The python script is executed on Databricks engine.

4.1.1. Mandatory field checks

The validation process starts by ensuring all required fields, such as `table_name`, `data_source_name`, and `primary_key`, are present in the metadata table. Missing fields trigger an alert. The `check_mandatory_fields` function scans for these missing fields.

4.1.2. Data type consistency

The `check_data_types` function verifies that each field conforms to the expected data type (e.g., integers for `batch_size`, strings for `status`). Any mismatches are flagged, along with the expected and actual types.

4.1.3. Standardization and correction of values

The `standardize_values` function ensures fields like `ingestion_type` and `refresh_schedule` have valid standardized values. Invalid or misspelled values are corrected using fuzzy matching techniques.

4.1.4. Consistency checks

Consistency checks are performed to ensure related fields align. For example, if `ingestion_type` is "incremental", the `incremental_ingestion_type` must not be empty, and the `watermark` field must be set for date-based ingestion. Functions like `check_ingestion_type_consistency` and `check_watermark_consistency` handle these checks.

4.1.5. Filling optional fields

Optional fields missing values are automatically populated with predefined defaults. The `fill_optional_fields_with_defaults` function ensures reasonable defaults like setting `last_ingestion_date` to a specific timestamp.

4.1.6. Final validation and reporting

Finally, the `validate_consistency` function consolidates all checks and generates a summary of alerts, ensuring the metadata is ready for ingestion once all validations are complete.

The logic described here is implemented in a script available in our GitHub [20] repository. The script contains self-explanatory comments that explain the content and purpose of each function, making it easy for others to understand and adapt the validation process. The GitHub repository also include an example for meta-data table, the data used in our experiments and the core transformation script to run the data pipeline using the meta-data ingestion approach.

4.2. Pipeline behavior validation

To ensure the reliability and adaptability of the ingestion pipeline, a series of validation tests were conducted under controlled conditions. The testbed for these validations was designed to simulate real-world scenarios and included a staging environment with a data lake, metadata tables, and a fully functional ingestion pipeline based on MIND design pattern. In this context, the metadata table acted as the central configuration point, storing all parameters necessary for the pipeline's operations, including ingestion types, schema definitions, transformation rules, and partitioning strategies. The MIND approach ensures that all configurations and ingestion logic are dynamically retrieved from the metadata table, making the pipeline adaptable to various data sources and ingestion requirements without manual reconfiguration.

To ensure robust validation of the MIND design pattern, we designed our experiments using two distinct datasets with varying characteristics, complexity levels, and update patterns:

- Dataset A - Tourism Data by World Region (UNWTO) [21] : This dataset contains tourism statistics with a simplified schema of three columns (region, year, tourist_arrivals), representing approximately 150 records per year across multiple regions. We scaled this dataset synthetically to create test versions ranging from 1K to 1 billion records to evaluate performance across different data volumes. The dataset's simple structure makes it ideal for testing basic ingestion mechanics and scalability patterns.
- Dataset B - Bank Transaction Dataset [20] : A more complex transactional dataset featuring 15+ columns including transaction_id, customer_id, transaction_amount, transaction_date, merchant_category,

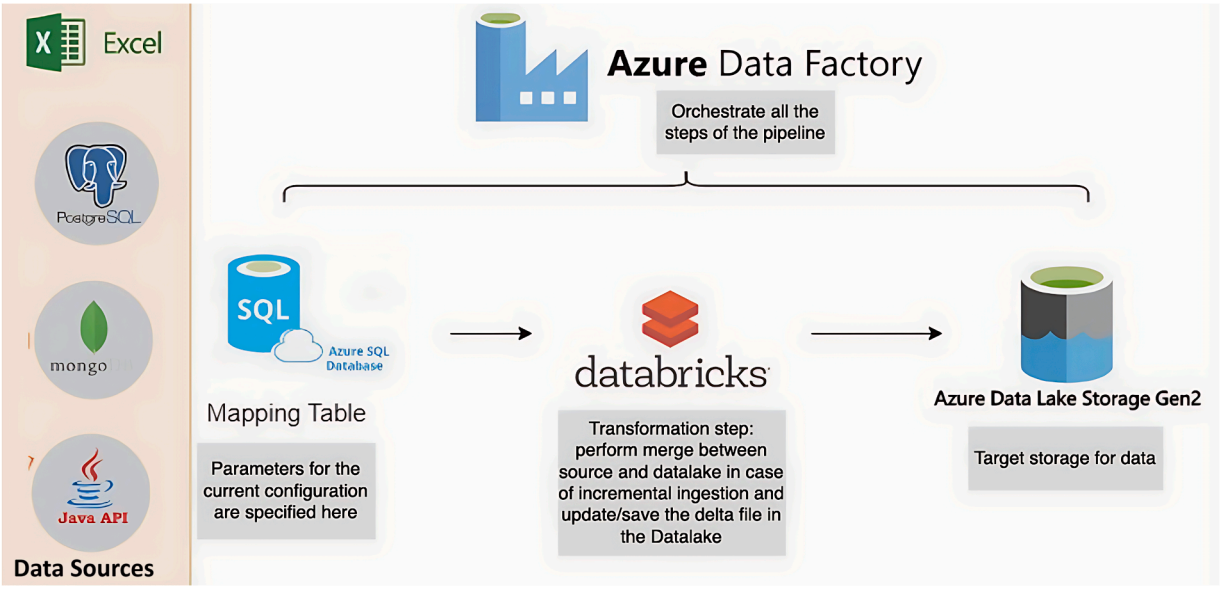

Fig. 3. Azure-based testbed architecture used for validating MIND design pattern.

and geographic information. This dataset simulates real-world financial transaction patterns with frequent updates, complex relationships, and varying data types. We generated realistic transaction patterns with temporal clustering and seasonal variations to test incremental ingestion capabilities.

The data pipeline used in our experiment is a normal data pipeline that consists of ETL operations.

To validate the effectiveness of the proposed MIND design pattern, we utilized an Azure cloud infrastructure as our testbed. This infrastructure, depicted in Fig. 3, includes essential tools such as Azure Data Factory for orchestration, Azure Databricks with Apache Spark for data processing, a SQL database for metadata storage, and Azure Data Lake Gen2 for storing ingested data. This integrated setup allowed us to perform comprehensive tests across different scales of data, comprising datasets with 1K, 100K, and 1M rows respectively. Using Azure Data Factory, we seamlessly integrated the metadata-driven approach, where parameters such as data source, table name, ingestion method (incremental or full refresh) and other configurations were dynamically retrieved from the SQL database. This level of customization ensured that each data ingestion task could adapt to the specific characteristics and update frequencies of the data sources involved.

4.3. Performance benchmarking

In this test, we are testing three ingestion methods on four identical tables of the same size. In the first case, full ingestion, the pipeline was configured to refresh entire tables, based on metadata specifying a complete data overwrite. This test ensured that the pipeline could delete existing records and load the entire dataset without errors or data loss. In the second case, all tables are ingested using incremental methods: the pipeline uses primary keys and hashing to ingest only new or updated records. Test scenarios included varying volumes of incremental updates to validate the pipeline's ability to scale and maintain data consistency. In the third case, two tables are ingested fully while the other two are ingested incrementally. This scenario simulates a data source that does not support incremental ingestion, possibly due to only providing a single snapshot of data. The flexibility to configure the ingestion method for each table in the metadata table allows us to retain the benefits of incremental ingestion for at least two tables, even in such mixed-source environments.

Table 7
Ingestion time comparison (in minutes).

#rows	Dataset	Incremental	Full	Hybrid
1K	A	0.9	0.7	0.6
	B	1.2	1.0	1.0
10K	A	1.1	1.5	1.3
	B	2.4	3.0	2.7
100K	A	2.2	4.5	3.1
	B	3.5	8.0	5.0
1M	A	3.4	9.2	5.2
	B	4.7	14.0	7.0
1B	A	5.1	18.3	7.1
	B	6.0	26.0	9.3

The metadata-driven approach is key in these tests. By specifying each table's ingestion method (full, incremental) within the metadata, the pipeline can dynamically adjust to the data characteristics and business requirements. This flexibility ensures that the pipeline operates efficiently, adapting without the need for manual adjustments or hard-coding.

We calculated the execution times for full ingestion, incremental ingestion, and hybrid ingestion, implemented using hash comparison, assuming that at least 60 % of the table data remained unchanged between pipeline runs.

The results are summarized in Table 7 below, where dataset A represents the tourism dataset [21], while dataset B represents the bank transaction dataset [20]:

The incremental ingestion method demonstrates its clear advantages as dataset sizes increase, reducing unnecessary processing, optimizing resource utilization, and minimizing execution times. As shown in the table above, the incremental ingestion method consistently outperforms full ingestion, especially for large datasets, making it the preferred choice for scalable data workflows. Additionally, hybrid ingestion, a combination of full and incremental ingestion methods, provides flexibility by ensuring optimal resource use and balancing execution speed with data accuracy. Hybrid ingestion is particularly beneficial when both up-to-date data and resource optimization are essential. It can be concluded from the results that the design pattern saves time and efforts for efficient data ingestion.

Table 8
Schema changes detected and adapted by the pipeline.

Change Type	Detected Change	Action Taken by Pipeline
New Column Added	schema change	Automatically added to target schema without manual intervention
Column Data Type Change	transaction_date from STRING to DATE	Target schema updated to reflect new data type
Removed Column	card_number column	Column removed from target schema to reflect the latest version

In addition to optimizing ingestion methods, MIND also supports Dynamic Schema Updates, an essential feature for handling schema changes in evolving datasets. In a typical data pipeline, schema changes such as adding new columns, changing data types, or adjusting key fields may require manual intervention and hard-coded logic. However, in MIND design pattern, metadata versioning captures schema changes, ensuring the pipeline adapts to these updates automatically without disrupting data processing. For instance, in a recent test, we simulated schema changes in the Bank Trasaction Dataset [20] by introducing a new column. This schema change was automatically detected by the pipeline using metadata versioning, and the target schema was updated accordingly. The pipeline adjusted to the new column seamlessly, without any need for hard-coding, demonstrating the power of metadata-driven schema management.

To illustrate how the pipeline handles dynamic schema changes, the following table shows a simulation of schema modifications, including the detection of changes and the automatic update of the target schema Table 8.

Through this metadata-driven approach, the pipeline is able to scale efficiently, ensuring that data ingestion workflows are flexible, optimized, and easily maintainable. MIND provides the foundation for these adaptive configurations, enabling organizations to manage and streamline their data ingestion processes in a scalable and efficient manner.

5. Conclusions

The implementation of a metadata-driven ingestion pipeline significantly reduces complexity and enhances operational efficiency. By centralizing configurations within a metadata table, a single pipeline can handle multiple data sources, eliminating the need for separate pipelines for each source. This results in faster development times, with onboarding new data sources requiring just a few hours to update the metadata table compared to several days for traditional approaches. Maintenance overhead is also minimized, as schema changes and ingestion logic updates are dynamically handled through metadata, reducing manual interventions and error rates.

Moreover, this approach introduces a high degree of customization and flexibility. Parameters such as ingestion type, transformation logic, or refresh schedules can be modified by simply updating a single cell in the metadata table, without requiring changes to the underlying pipeline code. This simplifies the process of adapting to new business requirements, data sources, or operational constraints, further emphasizing the value of a metadata-driven approach, as exemplified by MIND (Metadata-Driven Ingestion Network and Design).

Beyond quantitative improvements, the reduction in ingestion time and operational complexity translates into tangible benefits for organizations. Faster ingestion cycles enable near real-time analytics and more timely decision-making. Reduced manual configuration lowers the risk of human error and decreases both the time and effort required to onboard new data sources, an especially critical factor for industries managing dynamic and sensitive data, such as finance and healthcare. Furthermore, minimizing long execution times not only accelerates delivery but also reduces operational costs, since prolonged processes typically consume more resources. Finally, the ability to unify ingestion strategies within a single metadata-driven framework simplifies governance and ensures consistent application of data quality and compliance rules.As demonstrated by our experiments, the benefits of this paradigm are measurable and substantial. Metrics such as time-to-production, scalability, and error reduction underscore its effectiveness: organizations can achieve up to 50 % faster pipeline development, reduce maintenance efforts by over 80 %, and lower operational costs by 30 %. Additionally, the reusability of pipeline components enables a standardized and consistent approach to data ingestion across diverse scenarios, while automated validation processes ensure data quality and reliability.

The combination of these features allows data engineers to focus on higher-value tasks, such as designing advanced data models or optimizing analytics frameworks, rather than spending time on repetitive pipeline development and maintenance. By enabling a scalable, cost-efficient, and highly adaptable solution to modern data engineering challenges, metadata-driven pipelines, as facilitated by MIND, set a new benchmark for operational excellence in data architectures.

The applicability of MIND is not limited to controlled experiments. Its metadata-driven ingestion model directly supports real-world domains such as finance (fraud detection pipelines requiring rapid incremental ingestion), healthcare (continuous monitoring data with schema evolution), and tourism (integration of heterogeneous regional datasets). By reducing time-to-ingestion and simplifying maintenance, MIND offers a practical framework that organizations can adopt to accelerate data-driven decision making across industries.

While MIND demonstrates clear advantages in scalability, adaptability, and efficiency, several limitations remain. Our test scenarios were based on structured tabular datasets; future work should explore semi-structured and unstructured sources such as JSON logs or sensor data streams. Second, while the Unified Metadata Table simplifies pipeline orchestration, it introduces a dependency on accurate metadata management, meaning that errors in metadata definition could propagate into ingestion failures. In addition, the metadata table itself can become a single point of failure, and if not implemented following best security practices (e.g., integration with key vaults or secure credential management), it may expose organizations to potential security risks. Addressing these limitations will help refine and extend the applicability of MIND.

6. Future works

Building on the foundation of metadata-driven pipelines, the next step is to explore the integration of AI to optimize metadata configurations dynamically, enhancing pipeline performance. This involves deploying an AI model to analyze ingestion logs and recommend metadata optimizations, such as optimal partitioning strategies or scheduling adjustments. The generated recommendations will then be applied to the metadata table, followed by rerunning the ingestion pipeline to evaluate the impact. Performance metrics before and after these optimizations will be compared to validate the improvements and assess the accuracy of the AI-driven suggestions. This innovative approach aims to achieve significant performance gains with minimal manual intervention, demonstrating the potential of AI to proactively tune metadata and elevate the overall efficiency and scalability of modern data pipelines.

Another important direction is the integration of domain-specific compliance and governance rules into the metadata table, particularly for regulated sectors such as finance and healthcare. In addition, expanding MIND's application to edge and IoT scenarios, where ingestion occurs in resource-constrained environments, would broaden its practical impact. Finally, benchmarking against a wider range of design patterns, including serverless ingestion architectures, will provide a more comprehensive evaluation of MIND's performance.

CRediT authorship contribution statement

Chiara Rucco: Writing – original draft, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Antonella Longo:** Supervision, Conceptualization; **Motaz Saad:** Writing – review & editing, Writing – original draft, Validation, Methodology.

Data availability

Link to the data is shared in the document

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Chiara Rucco reports financial support and administrative support were provided by Unisalento. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research was partially supported by grant from Italian Research Center on High Performance Computing, Big Data and Quantum Computing (ICSC) funded by EUNextGenerationEU (PNRR-HPC, CUP:C83C22000560007).

References

[1] A. Jain, S. Sumit, C. Monga, S. Mittal, Data engineering: an overview from a future perspective, in: 2023 6th International Conference on Contemporary Computing and Informatics (IC3I), 6, 2023, pp. 653–659. https://doi.org/10.1109/IC3I59117.2023.10398128

[2] S. Spati, Data engineering design patterns (DEDP), 2024, Online book https://dedp.online/.

[3] I.A. Machado, C. Costa, M.Y. Santos, Data mesh: concepts and principles of a paradigm shift in data architectures, Procedia Comput. Sci., 196 (2022) 263–271.

[4] A.A. Munshi, Y.A.R.I. Mohamed, Data lake lambda architecture for smart grids big data analytics, IEEE Access, 6 (2018) 40463–40471.

[5] P. Sangat, M. Indrawan-Santiago, Sensor data management in the cloud: data storage, data ingestion, and data retrieval, Concurrency Comput. Pract. Exper. 30 (1) (2018) e4354.

[6] C. Rucco, A. Longo, M. Saad, Enhancing data ingestion efficiency in cloud-based systems: a design pattern approach, Data Sci. Eng. (2025). https://doi.org/10.1007/s41019-025-00300-2

[7] M. Kiran, P. Murphy, I. Monga, J. Dugan, S.S. Baveja, Lambda architecture for cost-effective batch and speed big data processing, in: 2015 IEEE International Conference on Big Data (Big Data), IEEE, 2015, pp. 2785–2792.

[8] J.B.N. Penka, S. Mahmoudi, O. Debauche, An optimized kappa architecture for IoT data management in smart farming, Int. J. Ubiquitous Syst. Pervasive Netw. 17 (2) (2022). https://doi.org/10.5383/JUSPN.17.02.002

[9] D. Linstedt, M. Olschimke, Building a Scalable Data Warehouse with Data Vault 2.0, Morgan Kaufmann, 2015.

[10] C. Badii, P. Bellini, D. Cenni, P. Nesi, Interoperability in internet of things: taxonomies and open challenges, Mob. Netw. Appl. 23 (3) (2018) 490–527.

[11] E. Hechler, M. Weihrauch, Y. Wu, Data fabric and data mesh research areas, in: Data Fabric and Data Mesh Approaches with AI: A Guide to AI-Based Data Cataloging, Governance, Integration, Orchestration, and Consumption, Springer, 2023, pp. 375–392.

[12] Microsoft, Cloud design patterns, 2023, Available online: https://learn.microsoft.com/en-us/azure/architecture/patterns/.

[13] AWS, Architectural patterns to build end-to-end data driven applications on AWS, 2023, Available online: https://docs.aws.amazon.com/whitepapers/latest/build-e2e-data-driven-applications/data-driven-architectural-patterns.html.

[14] GCP, Data analytics design patterns - Google cloud, 2023, Available online: https://cloud.google.com/architecture/reference-patterns/overview.

[15] P.K. Vattumilli, Metadata-driven ETL pipelines: a framework for scalable data integration architecture, Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol. 9 (6) (2024) 1224–1229. https://doi.org/10.32628/cseit241061224.

[16] Databricks Community, Metadata-driven ETL framework in databricks (Part-1), 2024, https://community.databricks.com/t5/technical-blog/metadata-driven-etl-framework-in-databricks-part-1/ba-p/92666.

[17] R. Gosavi, Building a metadata-driven framework in azure data factory for flexible pipeline execution, 2024, Accessed: 2024-01-27, https://www.linkedin.com/pulse/building-metadata-driven-framework-azure-data-factory-gosavi/.

[18] P. Dutta, B. Singh, S. Lal, M. Arora, A. Raghav, G. Benneh Mensah, H. Alkattan, Encoding IoT for patient monitoring and smart healthcare: connected healthcare system fostering health 6.0, Babylonian J. Internet Things 2023 (2024) 48–58. https://doi.org/10.58496/BJIoT/2023/007

[19] M. Mohammed, M. Ahmed, V. Abdullayev, Data-driven sustainability: leveraging big data and machine learning to build a greener future, Babylonian J. Artif. Intell. 2023 (2023) 17–23. https://doi.org/10.58496/BJAI/2023/005

[20] Chiara Rucco, Data engineering design pattern github repo, 2025, https://github.com/ChiaraRucco/DataEngDesignPattern_paper.

[21] United Nations World Tourism Organization (UNWTO), Tourism data by world region - UNWTO (2019), 2019, https://github.com/owid/owid-datasets/blob/master/datasets/Tourism%20data%20by%20World%20Region%20-%20UNWTO%20(2019)/Tourism%20data%20by%20World%20Region%20-%20UNWTO%20(2019).csv.