Toward Generalizable and Extensible Workflow Automation for Multi-Source Data Processing

Toward Generalizable and Extensible Workflow Automation for Multi-Source Data Processing

1st Danial Soltanali Khalili
Politecnico di Torino
Torino, Italy
danial.khalili@polito.it

2nd Alessio Viticchié
AlphaWaves S.r.l.
Torino, Italy
a.viticchie@awaves.it

3th Felice Cetrone
AlphaWaves S.r.l.
Torino, Italy
f.cetrone@awaves.it

4th Edoardo Patti
Politecnico di Torino
Torino, Italy
edoardo.patti@polito.it

5th Alessandro Aliberti
Politecnico di Torino
Torino, Italy
alessandro.aliberti@polito.it

***Abstract*—Modern data-driven applications increasingly demand flexible, scalable, and generalizable solutions to manage complex, heterogeneous workflows. Traditional Workflow Management Systems (WMS) often fall short in dynamic, high-throughput contexts due to their reliance on static configurations and limited adaptability. This paper introduces a modular and extensible framework for building dynamic data processing pipelines capable of integrating multiple data sources and evolving analytical tasks. The proposed system employs a three-layer architecture to facilitate pipeline construction, validation, and deployment through reusable components and standardized metadata. The framework simplifies the creation and connection of processing steps by using a standardized, modular approach that ensures data compatibility and logical consistency, making it accessible and useful for both expert and non-expert users. A real-world use case involving IoT data from the GAIA Meta-platform validates the system's effectiveness, showcasing features such as runtime block isolation, robust input validation, and dynamic workflow execution. This framework aims to significantly reduce integration overhead while enhancing reusability and adaptability across diverse domains.**

***Index Terms*—Dynamic Data Pipelines, Modular Workflow Automation, Heterogeneous Data Integration, Semantic Validation**

I. INTRODUCTION

Data is ubiquitous in our modern world, spanning various domains and impacting numerous aspects of daily life. Non-expert users often face significant challenges when attempting to harness this vast amount of information to extract valuable insights and make informed decisions [1], [2]. In today's fast-paced industries, rapid access to data-derived insights is essential. Advances in cloud computing and related technologies have narrowed the gap between data creation and its exploitation, yet bottlenecks in data processing remain prevalent [3].

Current Workflow Management Systems (WMS) are widely adopted for their utility in constructing and managing analytical processes across various disciplines. However, these systems exhibit significant limitations in highly dynamic and high-throughput environments. One primary shortcoming is their reliance on static configurations, which makes it challenging to adapt workflows to evolving data sources or integrate new Application Programming Interfaces (APIs) [4]–[6]. Therefore, there is a clear demand for supplementary methodologies that facilitate API-driven, dynamic pipeline creation and support flexible multi-source data integration.

In response to the increasing demand for adaptable and interoperable data processing solutions, this paper introduces a general-purpose framework for building dynamic pipelines. The architecture operates effectively in diverse data-intensive contexts, enabling users to assemble, validate, and execute complex processing sequences using modular, reusable components. For example, in predictive maintenance, users can build pipelines that ingest sensor data from industrial machines to forecast failures; in environmental monitoring, pipelines can dynamically combine weather station data and satellite feeds for early warning systems; for smart grid optimization, it can process time series data from smart meters to balance energy loads.

By structuring workflows around clearly defined metadata and standardized interfaces, the framework ensures strong guarantees of compatibility and correctness, while supporting customization and extension. Its architecture promotes maintainability and ease of integration, enabling the progressive construction of workflows that remain robust in the face of changing data schemas or evolving analytical requirements.

The practical relevance of the framework is demonstrated through its initial application within the context of the GAIA Meta-platform [7], a domain-agnostic infrastructure designed to enable semantic harmonization and analytical operations over heterogeneous IoT data sources.

The GAIA Meta-platform was selected as the initial testing environment due to its representative complexity and its capacity to provide access to real-world, heterogeneous IoT data sources. In this context, GAIA served as a data provider, supplying diverse inputs such as smart meter teleme-

This publication is part of the project NODES which has received funding from the MUR-M4C2 1.5 of PNRR founded by the European Union - NextGenerationEU (grant agreement no. ECS00000036).

try and environmental sensor readings. These data streams enabled the instantiation of a real-world pipeline focused on energy consumption analysis, which in turn demonstrated the framework's ability to interoperate with existing platforms, dynamically construct workflows, and enforce structural and semantic validation. This use case illustrates the system's adaptability, extensibility, and practical relevance across data-intensive application scenarios.

The rest of the paper is organized as follows. Section II reviews existing solutions and related works. Section III summarizes the key requirements the system must fulfill. Section IV details the proposed methodology for creating, describing, and executing dynamic data processing pipelines. Section V presents a practical use case demonstrating the system's utility. Finally, Section VI provides a unified discussion and conclusion, summarizing the findings and outlining future directions.

II. Related Works

The integration of complex, large-scale, and heterogeneous data has long driven the evolution of workflow systems and data integration methodologies. Traditional Extract, Transform, Load (ETL) processes are foundational in business intelligence systems, facilitating the collection and normalization of data from disparate sources for centralized analysis [8]–[10]. Recently, on-demand ETL approaches have emerged to support more dynamic integration scenarios and improve reproducibility in scientific research [11], [12].

Workflow Management Systems (WFMS) extend these data integration strategies by enabling the design, execution, and monitoring of task sequences aimed at achieving specific goals [13], [14]. Scientific workflow systems further adapt this paradigm for data-intensive research, emphasizing scalability, collaboration, and automation [15], [16]. Tools such as Pegasus and Apache Airflow have become prominent for their ability to manage distributed workflows using Directed Acyclic Graphs (DAGs), which represent workflows as nodes and directed edges where each edge denotes a dependency between tasks [17], [18].

Despite the advances brought by tools like KNIME [19], [20], Nintex [21], Pegasus [22], and Airflow [23], significant challenges remain. Many existing systems are constrained by static configurations, requiring manual modification when adapting to new requirements or evolving data schemas [24], [25]. Their reliance on predefined workflows often impedes dynamic task composition and limits adaptability in fast-changing environments [26]. Furthermore, data flow and control logic are frequently hardcoded, reducing reusability and generalizability.

Addressing these limitations, recent approaches emphasize the use of flexible and extensible workflow orchestration frameworks. For example, enhanced DAG-based systems support dynamic pipeline construction and resource-aware task scheduling [27], [28]. Logging, monitoring, and modular task definition improve debugging and performance optimization [29], [30]. These techniques provide a partial solution but often stop short of enabling semantically-rich pipeline generation across heterogeneous sources.

In response to current limitations, we propose a system that integrates metadata-based workflow specification, modular construction, and flexible execution into a unified, extensible framework. The novelty of this approach lies in its ability to abstract and modularize pipeline components while maintaining adaptability across execution engines, i.e., software platforms responsible for scheduling and running workflows. This enables domain experts, data engineers, and non-specialist users to collaboratively build, reuse, and evolve data workflows without deep system-level knowledge. The framework supports high-level workflow synthesis, effective integration of heterogeneous data sources, and flexible pipeline adaptation to evolving contexts. These capabilities assist both expert and non-expert users by reducing integration complexity, fostering reusability, and enabling more intelligent and responsive data infrastructures.

III. Requirements Analysis

From a requirements perspective, the proposed dynamic data processing pipeline system must adhere to several critical principles to ensure functionality, scalability, and adaptability. A foundational requirement underpinning the development of this framework is its dual usability by both expert and non-expert users. The system has been explicitly conceived to lower the technical entry barrier for non-specialist users, enabling them to construct, configure, and execute complex data workflows with minimal prior knowledge of programming or data engineering. Simultaneously, it provides expert users with the flexibility and extensibility required to define custom processing blocks and enforce domain-specific logic. This inclusive approach ensures that the framework can serve a broad spectrum of users, from domain specialists and analysts to developers and system architects, across diverse application domains such as energy management, smart city analytics, industrial monitoring, healthcare data integration, and financial data processing. These requirements are as follows:

- *Enable the Dynamic Creation of Pipelines:* The system must allow users to create and modify data processing pipelines dynamically. This flexibility is essential to adapt workflows to changing data sources, business needs, and analytical goals without requiring downtime or major reconfiguration.
- *Facilitate Block Generation:* The system must empower expert users to easily create new processing blocks. A processing block is a modular, reusable unit that encapsulates a data transformation or analysis function within the pipeline. Easy creation is essential to quickly adapt to new data sources, analytical methods, or domain requirements, enabling fast integration of new functionality without major architectural changes.
- *Support Multiple Data Formats:* The system must handle a wide range of data formats, including time-series and non-temporal data. Supporting multiple formats is critical to integrate heterogeneous data sources across different

domains and use cases—for example, processing sensor telemetry, electronic health records, transaction logs, or weather forecasts. The system must provide flexible, accurate data transformation mechanisms that maintain data integrity across formats.

- *Ensure Compatibility Between Blocks:* The system must guarantee compatibility between processing blocks through compile-time and runtime type checking. This is necessary to prevent data mismatches, ensure correct data flow, and maintain pipeline integrity as blocks are added or updated. For instance, when integrating a statistical analysis block after a data cleaning block in an industrial monitoring pipeline, the system must validate that the output structure is compatible. The system must provide robust validation mechanisms to enforce type and interface compatibility between blocks.
- *Allow Customizable Type Checking for External Data:* The system must support customizable type validation for external data sources. This feature enables users to define domain-specific validation rules that align external data with internal processing requirements, ensuring data integrity and consistency. For example, a user may need to enforce a schema check for financial transaction records or validate sensor metadata in an IoT deployment. The system must allow users to extend or replace default type checks with custom validation logic as needed.
- *Manage Dependency Contradictions Between Blocks:* The system must be capable of automatically managing the dependencies between processing blocks. This includes resolving any conflicts that may arise due to contradictory dependencies and ensuring that blocks are executed in the correct sequence. The system should provide a conflict resolution mechanism that ensures that data flows through the pipeline efficiently, without delays or errors due to dependency mismatches.
- *Provide Comprehensive Logging and Monitoring:* The system must include a comprehensive logging and monitoring framework that enables users to track and troubleshoot the operation of the pipeline. This feature will support both research and production needs, providing real-time visibility into the system's performance and facilitating debugging, performance optimization, and overall system maintenance. The logging mechanism should capture detailed information on data flows, block execution, and system errors, offering a robust foundation for monitoring and troubleshooting.

IV. Methodology

The proposed system enables the creation, description, and execution of dynamic, modular data processing pipelines through a structured, three-layer architecture (Figure 1). The figure depicts the three fundamental layers of the proposed architecture: the Data Representation Layer, which formalizes processing block and pipeline metadata; the Creation Layer, which enables the dynamic generation, validation, and configuration of modular workflows; the Execution Layer, where a dedicated Translation Service converts abstract pipeline models into concrete execution instructions tailored to the specific underlying engine (e.g., Apache Airflow). This design ensures flexibility, extensibility, and maintainability across a variety of application domains.

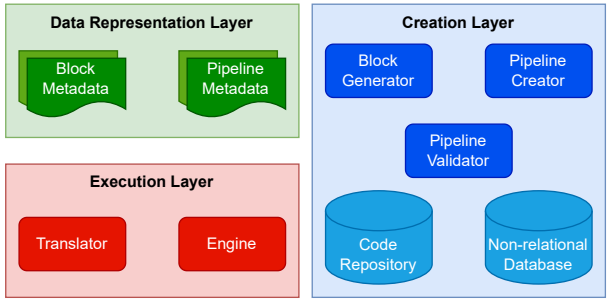

Fig. 1. Architecture of the dynamic data processing framework.

A. Data Representation Layer

The *Data Representation Layer*, as illustrated in Figure 1, defines the metadata and structural descriptions required to model both individual processing blocks and complete pipelines (i.e., *Block Metadata* and *Pipeline Metadata*). This metadata serves as the foundation for ensuring compatibility between components, enabling automated validation of data flows and promoting the reuse of processing logic across multiple pipelines.

A processing block in this system is a modular and reusable unit that encapsulates a specific data transformation, analysis function, or operation. Each block is described by attributes such as its name, version, description, input and output specifications, and required dependencies. The implementation of each block is provided in Python, chosen as the primary programming language for its widespread adoption in data science, machine learning, and scientific computing. Python's rich ecosystem of libraries (e.g., pandas, scikit-learn, NumPy) facilitates rapid development of data processing functions while ensuring compatibility with industry-standard tools. Its balance between simplicity and expressiveness makes it accessible for both expert developers and domain specialists, enabling faster onboarding and collaboration across multidisciplinary teams.

Each block's metadata also includes explicit input and output definitions using standard data types such as lists, dictionaries, or dataframes. The system records dependency requirements to ensure necessary libraries are installed and managed during execution. Additionally, optional semantic properties can enrich input and output specifications, enabling validation beyond basic type checking.

A pipeline represents an orchestrated sequence of processing blocks connected together to form a complete data workflow. It defines how data flows from one block to another, specifying the order of operations, data transformations, and dependencies required to achieve a specific processing goal. The layer captures Pipeline Metadata describing how blocks

are connected, including the sequence and mapping of data flows, execution priorities to preserve dependencies, and error-handling strategies to define recovery behavior. By formalizing this information, the system creates explicit, verifiable representations that support automated compatibility checks and clear data lineage.

B. Creation Layer

The *Creation Layer*, as shown in Figure 1, provides the functionality required to compose, validate, and manage both processing blocks and pipelines dynamically. It contains all the operational services and tools that enable users to actively create, configure, and validate these entities. It acts as the bridge between abstract block definitions and executable workflows, enabling both expert and non-expert users to assemble and configure pipelines tailored to specific tasks, while also supporting the development and registration of new blocks.

Within this layer, expert users can generate new processing blocks using the *Block Generator* service, by specifying their metadata, implementation logic in Python, and dependencies. The system packages these blocks into self-contained modules saved in a *Code Repository* (i.e. PyPI) and stores their definitions in a *Non-Relational Database* (i.e. MongoDB), and maintains version control to support traceability and reuse. This process allows the system to evolve and expand functionality without requiring modifications to its core architecture.

Once the blocks have been defined, users can construct analytical pipelines through the *Pipeline Generator*, by selecting and interconnecting the appropriate blocks in accordance with their specific analytical objectives. The system accommodates adapters to ingest data from external sources, for example MySQL or MongoDB, and exporters to store results in various destinations, including InfluxDB or PostgreSQL, offering flexibility in integrating diverse data environments. Pipelines can be reused across different contexts by substituting adapters and exporters as needed, without altering the core processing logic. For example, the same pipeline designed for performing anomaly detection or data normalization can be reused across different datasets or domains by simply switching the data source adapter and output exporter.

During the pipeline creation, the system leverages the *Pipeline Validator* to execute a validation process aimed at verifying the coherence of input and output specifications across all interconnected processing blocks. This procedure guarantees alignment between consecutive blocks, thereby preventing runtime errors and safeguarding data integrity. Compatibility checks are performed both on data types and, where applicable, on semantic properties defined by the user. These semantic validations may invoke external HTTP services to perform context-specific compatibility assessments. If incompatibilities are detected, users are notified and provided with information to resolve the issues before deployment.

C. Execution Layer

The *Execution Layer*, as outlined in Figure 1, is responsible for translating validated pipeline definitions into executable workflows and managing their execution within production environments.

One of the foundational components of this layer is the *Execution Engine*, which orchestrates the actual execution of processing blocks in accordance with the predefined pipeline structure. This module ensures that processing tasks are executed sequentially and correctly, in accordance with the defined data and control dependencies. The validation of incoming data from the adapters is carried out once, at the beginning of the pipeline execution. This is achieved by injecting dedicated system blocks into the pipeline, whose sole function is to validate that input data conforms to predefined formats and schemas. These validation blocks act as gatekeepers, ensuring that only compliant data is propagated to downstream processing stages, thereby safeguarding data integrity and supporting robust error handling.

Another essential component of this layer is the *Translator* service, which acts as an interpreter between abstract pipeline models and their concrete implementations. More precisely, it transforms the metadata associated with pipelines and processing blocks into a set of executable instructions, tailored to the specifications of the target execution engine. This design fosters adaptability by decoupling the Execution Layer from upstream components, thereby enabling support for heterogeneous execution backends. The abstraction introduced by the Translator service further ensures engine-agnostic execution, such that extending the framework to new platforms requires solely the implementation of a translation module capable of mapping its formalism to the target engine's execution semantics.

Currently, the framework integrates Apache Airflow as its primary execution engine. Airflow's Directed Acyclic Graph (DAG)-based model is leveraged to represent each pipeline as a collection of interdependent tasks. Within this model, each processing block corresponds to an individual task, and the inter-block data flows are represented as directed edges, thereby ensuring correct execution order and dependency resolution. To address software dependency isolation, each task is executed within a dedicated Python virtual environment. This strategy prevents versioning conflicts and promotes maintainability across diverse processing components.

The execution environment is augmented by a comprehensive logging infrastructure, which captures detailed information regarding data flow trajectories, task execution states, and system-level errors. These logs are instrumental for auditing, debugging, and performance tuning. Furthermore, the system incorporates rollback mechanisms that allow restoration to a prior consistent state in the event of execution failures, thus enhancing operational resilience.

V. Use Case: Building a Pipeline for Energy Analysis

To validate the functionality and applicability of the dynamic pipeline system, we present a real-world use case that leverages the GAIA Meta-platform [7] as a data provider. GAIA serves as a domain-agnostic infrastructure designed

to support semantic harmonization and analytical operations across heterogeneous IoT data sources. Within this context, the platform enables the integration and processing of heterogeneous data, offering a representative and operationally complex testbed.

This use case does not aim to achieve accurate forecasting outcomes, but to experimentally validate the framework's ability to describe, coordinate, and deploy complex workflows.

It serves as a demonstrator for assessing the expressiveness of the underlying formalism used to describe processing blocks and pipelines, as well as the platform's interoperability with existing data systems in a realistic operational context.

To evaluate the framework in a realistic scenario, the use case focuses on residential energy forecasting and optimization, supporting both households and Renewable Energy Communities (RECs) in managing energy demand.

The pipeline automates the transformation of daily energy logs into an hourly dataset, forecasts next-day hourly consumption using a machine learning model, extracts hourly historical baselines (mean and standard deviation), and generates smart usage recommendations based on forecasted demand deviations.

The workflow, depicted in Figure 2, adopts a non-linear Directed Acyclic Graph (DAG) structure, allowing tasks to consume outputs from multiple predecessors.

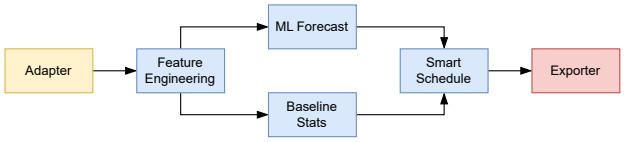

Fig. 2. Non-linear workflow structure for the energy analysis pipeline. The adapter feeds the feature engineering block, which branches into forecasting and statistical analysis paths. Both are merged in the final smart scheduling stage.

Each task in the pipeline is implemented as an independent, reusable block, described through a JSON schema. For example, the `ml_forecast` block specifies its dependencies using the `requirements` field:

```
"requirements": ["pandas", "numpy==2.2.2",
                 "scikit-learn", "joblib"]
```

This demonstrates how the platform dynamically constructs pipelines from versioned, modular components. The `requirements` field also instructs the system to build an isolated virtual environment, preventing dependency conflicts between blocks—a key requirement for maintainability and reuse.

Inputs to each block are typed using a standard schema. For instance, the `features_df` input is declared with the expected internal structure:

```
"features_df": {
  "type": "pandasDataFrame",
  "items": {
    "object": {
      "hour": {"type": "int"},
      "consumption": {"type": "float"}
    }
  }
}
```

This ensures that all data transformations across the pipeline adhere to expected formats, enabling both static and runtime type safety. Validation is reinforced using the `check_paths` directive:

```
"check_paths": {
  "features_df.consumption":
  "feature_engineering.features_df.consumption"
}
```

This guarantees that the structure of the data output from `feature_engineering` matches the expectations of downstream blocks, such as `ml_forecast`. The system performs these checks during pipeline compilation to create the pipeline blueprint.

Additionally, the `paths` field supports fine-grained mapping of individual columns between blocks, allowing flexibility in data routing and transformation. This mechanism underlies the ability to construct non-linear DAGs—such as the one shown in Figure 2—where outputs from a single task (or a portion of an output) are consumed concurrently by multiple successors.

The block at the start of the pipeline (i.e. Adapter) is responsible for connecting the pipeline to an external data source from which input data is retrieved. The JSON snippet provided below exemplifies the definition:

```
"adapter_description": {
  "connection_name": "pyGAIA",
  "database_type": "gaia",
  "query": "SELECT *
              FROM energy_consumption;",
  "variables_list":
  ["date", "consumption24", "metadata"]
}
```

This configuration defines the adapter module, specifying the connection to the GAIA infrastructure. It identifies the external database type (in this case, GAIA), the query for data ingestion, and the list of expected variables. The `variables_list` field allows the system to anticipate the structure of the incoming dataset, enabling automatic validation and mapping of the retrieved data to the corresponding pipeline inputs.

Finally, the same blocks—such as `ml_forecast` and `smart_schedule`—can be reused in other domains like traffic or water consumption by reusing the metadata interface and modifying only the input adapter. This demonstrates the pipeline system's extensibility and long-term adaptability.

The proposed energy analysis pipeline offers a concrete demonstration of the system's architecture. It validates core capabilities including modular block execution, DAG-oriented orchestration, runtime isolation, type compatibility, and seamless external data integration. These characteristics make the platform suitable for a broad range of real-world, data-intensive applications in both technical and non-technical environments.

VI. Discussion and Future Works

This work presents a modular and extensible framework for dynamic data processing pipelines, structured around a layered architecture that separates concerns across Data Representation, Creation, and Execution Layers. This design enhances flexibility, maintainability, and scalability, supporting diverse data-driven applications.

A key innovation of the system stems from the capacity of its formal, machine-readable descriptions to support automated validation at two distinct levels: data type consistency and semantic correctness via user-defined parameters. These robust validation mechanisms ensure data integrity and enable the system to accommodate both generic and domain-specific workflows, enhancing adaptability across contexts. This capability also promotes human interpretability. The Creation Layer operationalizes these representations into modular, reusable components, while the Execution Layer manages their orchestration through Apache Airflow, and remains extensible to alternative execution backends. The support for Python ensures broad accessibility and integration with the data science ecosystem. Through the use of adapters and exporters, the system supports heterogeneous data environments, allowing pipelines to be reused across varying contexts.

Future directions include extending support to additional orchestration platforms (e.g., Apache Spark for distributed data processing), and exploring AI-assisted tools for automated workflow generation. Further work will also address optimization of resource allocation and enhancement of security for sensitive data processing.

In conclusion, the proposed framework offers a generalizable and robust solution to the challenges of building and managing dynamic data workflows in evolving, heterogeneous environments.

References

[1] R. Espinosa, L. Garriga, J. Zubcoff, and J. Mazón, "Knowledge spring process - towards discovering and reusing knowledge within linked open data foundations," 2014.

[2] T. Pengo, S. Holden, and S. Manley, "Palmsiever: a tool to turn raw data into results for single-molecule localization microscopy," *Bioinformatics*, vol. 31, pp. 797–798, 2014.

[3] D. Stodder, "Faster insights from faster data," *TDWI Best Practice Report*, vol. Q1, 2020.

[4] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. d. Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[5] E. Deelman, "Challenges of running scientific workflows in cloud environments," *Proceedings of the 6th Workshop on Scientific Cloud Computing*, 2015.

[6] C. Shan, C. Wu, Y. Xia, Z. Guo, D. Liu, and J. Zhang, "Adaptive resource allocation for workflow containerization on kubernetes," *Journal of Systems Engineering and Electronics*, vol. 34, pp. 723–743, 2023.

[7] A. Viticchié, F. Cetrone, C. Camarda, V. Vassallo, L. Napoli, E. Patti, and A. Aliberti, "Gaia meta-platform: enabling multi-energy vectors data analysis via iot federation," in *2024 IEEE International Conference on Environment and Electrical Engineering and 2024 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*. IEEE, 2024, pp. 1–6.

[8] G. Yu, J. Liu, J. Du, M. Hou, and V. Sugumaran, "An integrated approach for massive sequential data processing in civil infrastructure operation and maintenance," *Ieee Access*, vol. 6, pp. 19 739–19 751, 2018.

[9] J. D. Shi, M. Li, Q. H. Zhang, and X. Wei, "Etl application in telecom industry based on teradata," in *Advances in Computational Modeling and Simulation*, ser. Applied Mechanics and Materials, vol. 444. Trans Tech Publications Ltd, 1 2014, pp. 1702–1707.

[10] E. Nakucçi, V. Theodorou, P. Jovanovic, and A. Abelló, "Bijoux: Data generator for evaluating etl process quality," in *Proceedings of the 17th International Workshop on Data Warehousing and OLAP*, 2014, pp. 23–32.

[11] P. Kathiravelu, "On-demand big data integration: a hybrid etl approach for reproducible scientific research," 2018.

[12] S. Krishnan, D. J. Haas, M. J. Franklin, and E. Wu, "Towards reliable interactive data cleaning," *Proceedings of the Workshop on Human-in-the-Loop Data Analytics*, 2016.

[13] S. Lu and J. Zhang, "Collaborative scientific workflows supporting collaborative science," *International Journal of Business Process Integration and Management*, vol. 5, p. 185, 2011.

[14] W. M. P. v. d. Aalst, "Business process management demystified: a tutorial on models, systems and standards for workflow management," *Lecture Notes in Computer Science*, pp. 1–65, 2004.

[15] J. Liu, L. Villaseñor-Pineda, E. Pacitti, A. Costan, P. Valduriez, G. Antoniu, and M. Mattoso, "Efficient scheduling of scientific workflows using hot metadata in a multisite cloud," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, pp. 1940–1953, 2019.

[16] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, pp. 457–493, 2015.

[17] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Workflow management in griphyn," in *Grid Resource Management*, J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2004, pp. 99–116.

[18] M. Nara, A. Shaikh, and R. Pradhan, "Managing data pipeline with apache airflow," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 244–250, 2023.

[19] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel, "Knime-the konstanz information miner: version 2.0 and beyond," *AcM SIGKDD explorations Newsletter*, vol. 11, no. 1, pp. 26–31, 2009.

[20] A. Fillbrunn, C. Dietz, J. Pfeuffer, R. Rahn, G. A. Landrum, and M. R. Berthold, "Knime for reproducible cross-domain analysis of life science data," *Journal of Biotechnology*, vol. 261, pp. 149–156, 2017.

[21] C. Richardson and J. R. Rymer, "The forrester wave™: Low-code development platforms, q2 2016," Forrester Research, Inc., Tech. Rep., April 2016.

[22] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda, "Griphyn and ligo, building a virtual data grid for gravitational wave scientists," in *11th IEEE International Symposium on High Performance Distributed Computing, HPDC 02*, 2002.

[23] X. Li and B. Zou, "An automated data engineering pipeline for anomaly detection of iot sensor data," 2021.

[24] G. M. Yenni, E. M. Christensen, E. K. Bledsoe, S. R. Supp, R. M. Diaz, E. White, and S. K. M. Ernest, "Developing a modern data workflow for regularly updated data," *PLOS Biology*, vol. 17, 2019.

[25] A. M. Behdani, J. Lai, C. Kim, L. Basalelah, T. Halsey, K. L. Donohoe, and D. Wijesinghe, "Optimizing pharmacogenomic decision-making by data science," *PLOS Digital Health*, vol. 3, 2024.

[26] M. Syafrudin, G. Alfian, N. L. Fitriyani, and J. Rhee, "Performance analysis of iot-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing," *Sensors*, vol. 18, p. 2946, 2018.

[27] A. K. Jana, "The mlops approach to model deployment: a road map to seamless scalability," *Journal of Artificial Intelligence &Amp; Cloud Computing*, pp. 1–4, 2022.

[28] E. E. Wahyudi, M. Auzan, A. Dharmawan, D. E. Nuryanto, N. Susyanto, G. Samodra, and D. S. Hadmoko, "Akuisisi data prediksi curah hujan secara periodik menggunakan apache airflow," *Journal of Informatics, Information System, Software Engineering and Applications (INISTA)*, vol. 4, pp. 1–12, 2022.

[29] M. Kotliar, A. V. Kartashov, and A. Barski, "Cwl-airflow: a lightweight pipeline manager supporting common workflow language," *GigaScience*, vol. 8, 2019.

[30] A. Mastoras and T. Groß, "Chunking for dynamic linear pipelines," *ACM Transactions on Architecture and Code Optimization*, vol. 16, pp. 1–25, 2019.