SLEGO: A Collaborative Data Analytics Architecture with LLM Recommender for Diverse Users

Siu Lung Ng

A thesis in fulfilment of the requirements for the degree of

Doctor of Philosophy

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

July 2025

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Ng**

First name: **Siu Lung** Other name/s: **Alan**

Abbreviation for degree as given in the University calendar: **PhD**

School: **School of Computer Science and Engineering** Faculty: **Faculty of Engineering**

Title: SLEGO: A Collaborative Data Analytics Architecture with LLM Recommender for Diverse Users

Abstract

Abstract

Declaration relating to disposition of project thesis/dissertation

I hereby grant the University of New South Wales or its agents a non-exclusive licence to archive and to make available (including to members of the public) my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known. I acknowledge that I retain all intellectual property rights which subsist in my thesis or dissertation, such as copyright and patent rights, subject to applicable law. I also retain the right to use all or part of my thesis or dissertation in future works (such as articles or books).

For any substantial portions of copyright material used in this thesis, written permission for use has been obtained, or the copyright material is removed from the final public version of the thesis.

Signature **Siu Lung Ng** Witness Date **25 July, 2025**

FOR OFFICE USE ONLY Date of completion of requirements for Award

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Siu Lung Ng
25 July, 2025

Copyright Statement

I hereby grant the University of New South Wales or its agents a non-exclusive licence to archive and to make available (including to members of the public) my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known. I acknowledge that I retain all intellectual property rights which subsist in my thesis or dissertation, such as copyright and patent rights, subject to applicable law. I also retain the right to use all or part of my thesis or dissertation in future works (such as articles or books).

For any substantial portions of copyright material used in this thesis, written permission for use has been obtained, or the copyright material is removed from the final public version of the thesis.

Siu Lung Ng
25 July, 2025

Authenticity Statement

I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis.

Siu Lung Ng
25 July, 2025

Abstract

The field of data analytics is hindered by a persistent gap between technical experts and domain specialists, forcing a trade-off between powerful, complex coding environments and inflexible low-code tools. This fragmentation creates inefficient workflows and impedes knowledge reuse. This thesis introduces SLEGO (Software-Lego), a novel collaborative data analytics architecture designed to solve this problem. The core solution is a unified platform that facilitates collaboration between three key roles: it empowers Technical Experts to create modular, reusable analytical components (microservices); enables Domain Experts to enrich these components with contextual knowledge; and allows End Users to discover and assemble them into complex pipelines using an intuitive graphical user interface (GUI), thereby making sophisticated analytics broadly accessible.

The SLEGO design is guided by three core objectives: enabling End-to-End Collaboration, providing a Modular & Extensible Architecture, and facilitating AI-Guided Knowledge Discovery. To achieve these goals, the platform architecture is composed of five principal components: a Low-Code User Interface, a Modular Microservices Architecture, Unified Cloud Storage, a shared Knowledge Base (KB), and an LLM-Based Recommender. This design operationalises the role-based workflow where the microservice architecture and KB provide modularity, while the LLM recommender is a key element that provides context-aware, verifiable pipeline suggestions by grounding its responses in the human-vetted content of the Knowledge Base, ensuring trustworthy AI-driven guidance.

The architecture's effectiveness is validated through three experiments simulating the lifecycle of different analytics tasks. Experiment 1 demonstrates the foundational collaborative workflow by successfully orchestrating the creation of a new pipeline from scratch through the distinct contributions of all three user roles. Experiment 2 validates the platform's modularity and extensibility by reusing the initial pipeline for a new task and extending it with technologically diverse microservices. Finally, Experiment 3 assesses cross-domain versatility and the AI-recommender's performance. Collectively, these experiments validate SLEGO as an architectural blueprint for a more inclusive, efficient, and intelligent analytics ecosystem.

Acknowledgements

I am sincerely grateful to my principal supervisor, Prof. Fethi Rabhi, whose steady guidance, insightful feedback, and unwavering support shaped every stage of this research.

I also thank the entire team at the Fintech AI Innovation Consortium (FAIC) for their invaluable advice, collaboration, and technical assistance. This project was jointly supported by the Fintech AI Innovation Consortium, its industry partners Ignition Advice and Brew AI, and the UNSW School of Computer Science and Engineering.

My appreciation extends to the wider UNSW research community—fellow PhD candidates, postdoctoral scholars, visiting researchers, and faculty—whose passion for software engineering and computer science continually broadened my perspective and helped shape my research interests and future directions. I am especially grateful to several UNSW PhD candidates and close friends—Jing Wen, Xing Yue, Ruo Yu, Hirad, Ming, and Ty—whose friendship, collaboration, and shared passion for research enriched this journey. The time we spent exchanging ideas and supporting one another made this experience both intellectually fulfilling and personally meaningful.

Finally, I owe my deepest gratitude to my family for their enduring encouragement, patience, and understanding. Their belief in me sustained this journey of personal growth and intellectual exploration.

List of Publications

1. Ng, S.L., Rabhi, F. (2023). A Data Analytics Architecture for the Exploratory Analysis of High-Frequency Market Data. In: van Hillegersberg, J., Osterrieder, J., Rabhi, F., Abhishta, A., Marisetty, V., Huang, X. (eds) *Enterprise Applications, Markets and Services in the Finance Industry. FinanceCom 2022. Lecture Notes in Business Information Processing*, vol 467. Springer, Cham. `https://doi.org/10.1007/978-3-031-31671-5_1`

2. Ng, S.L., Oza, B., Rabhi, F. (2025). Enhancing Collaborative Analytics with an Ontology: A Case Study with High Frequency Data Analysis. In: Hedman, J., Gleasure, R., Bandara, M. (eds) *Enterprise Applications, Markets and Services in the Finance Industry. FinanceCom 2024. Lecture Notes in Business Information Processing*, vol 541. Springer, Cham. `https://doi.org/10.1007/978-3-031-89933-1_3`

3. Rezaei, H.B., Ng, S.L., Rabhi, F. (2025). Enhancing Analytics Pipelines with Advanced Multi-agent Feedback Integration. In: Hedman, J., Gleasure, R., Bandara, M. (eds) *Enterprise Applications, Markets and Services in the Finance Industry. FinanceCom 2024. Lecture Notes in Business Information Processing*, vol 541. Springer, Cham. `https://doi.org/10.1007/978-3-031-89933-1_4`

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AutoML	Automated Machine Learning
AWS	Amazon Web Services
BI	Business Intelligence
CBSE	Component-Based Software Engineering
CDA	Collaborative Data Analytics
CPU	Central Processing Unit
CS	Cloud Storage
CSE	Computer Science and Engineering
CUDA	Compute Unified Device Architecture
DAG	Directed Acyclic Graph
DEG	Domain Expert Group
DMU	Data Management UI
DRAM	Dynamic Random-Access Memory
DSL	Domain-Specific Language
DSRM	Design Science Research Methodology
EDA	Event-Driven Architecture
ETL	Extract, Transform, Load
EUG	End User Group
FAIR	Findable, Accessible, Interoperable, Reusable

GCP	Google Cloud Platform
GUI	Graphical User Interface
HCI	Human-Computer Interaction
I/O	Input/Output
IT	Information Technology
KB	Knowledge Base
KMU	Knowledge Management UI
KRU	Knowledge Retrieval UI
LC/NC	Low-Code/No-Code
LLM	Large Language Model
MCRMSE	Mean Columnwise Root Mean Squared Error
ML	Machine Learning
MMU	Microservice Management UI
MRR	Mean Reciprocal Rank
MSA	Microservice Architectures
NLI	Natural Language Interface
PAU	Pipeline Assembly UI
RAG	Retrieval-Augmented Generation
RMSLE	Root Mean Squared Log Error
RMSSE	Root Mean Squared Scaled Error
ROC AUC	Receiver Operating Characteristic Area Under the Curve
RQU	Recommendation Query UI
SLEGO	Software-Lego
TEG	Technical Expert Group
UI	User Interface

Chapter 1

Introduction

1.1 Research Topic and Background

The rapid growth of data in volume, variety, and velocity [28, 42] has made analytics central to decision-making, innovation, and operational efficiency across many sectors. Organisations increasingly rely on extracting insights from complex datasets to navigate dynamic environments, while scientific research leverages advanced analytical techniques to push the boundaries of knowledge [23]. However, effectively using these data resources is often hindered by a significant capabilities gap: a divide exists between individuals with deep technical and programming expertise and those who possess rich domain knowledge but have limited analytical programming skills [51].

Traditional methods for bridging this gap, such as domain experts outlining requirements for technical teams to implement, often lead to inefficiencies, misinterpretations, and solutions that are not flexible enough for exploratory or evolving analytical needs [72]. While common tools like spreadsheets have made some data tasks more accessible, they often fall short for the scale and complexity of modern data analytics and can introduce challenges regarding accuracy, reliability, and reproducibility [44, 78]. Conversely, powerful programming environments like Python and Jupyter Notebooks [73], while excellent for

experienced data scientists and promoting reproducible research [96, 88], present a steep learning curve for a broader audience. This often results in a heavy reliance on specialised experts, which can create bottlenecks and limit the wider adoption of data-driven practices [41].

Furthermore, performing data analytics is seldom an individual task; it is increasingly a team effort that requires combining diverse perspectives, skills, and knowledge from multidisciplinary teams [96, 59]. These teams often include data scientists, software engineers, domain experts, and business stakeholders, each contributing unique insights and requirements [58, 72]. However, effective collaboration is often hindered by communication barriers, different goals, disconnected tools, and a lack of shared understanding (or "common ground") about data, methods, and interpretations [58, 70]. Key processes—like capturing analytical knowledge, ensuring that developed tools and findings can be reused, and managing the origin (provenance) of complex analyses—are often poorly supported by existing tools. This can lead to duplicated work and a failure to build on the organisation's collective knowledge [17, 93].

The rise of Artificial Intelligence (AI), especially Large Language Models (LLMs) [97], offers significant potential to reshape the data analytics landscape. LLMs can improve how users interact with data, automate parts of creating analytical workflows, offer intelligent guidance, and make it easier for people with different levels of expertise to communicate [50, 99, 67, 20]. However, effectively integrating these AI capabilities into comprehensive, user-friendly, and collaborative analytics platforms is still an active area of research and development. There is a significant need for new system designs that not only leverage the power of AI but also tackle the core challenges of making data analytics accessible, collaborative, and reusable.

1.2 Research Scope, Objectives, and Contributions

This thesis addresses the challenge of enabling effective collaboration in data analytics among users with diverse expertise. To this end, the research proposes, designs, and evaluates SLEGO (Software-Lego), a novel collaborative data analytics architecture. The primary research focus is on the design of a unified architectural framework that bridges the gap between expert-driven development and end-user application. The architecture is designed to lower the technical threshold for engaging with advanced analytics, facilitate structured collaboration across diverse user roles, and foster the systematic creation and reuse of analytical assets. The principal contributions of this thesis, which will be elaborated upon in subsequent chapters, are:

- **An Integrated Architectural Blueprint:** The design of a novel architecture that synergistically combines a microservice ecosystem, a formal graph-theoretic model, and a central, structured Knowledge Base to ensure modularity and reliability.
- **A Framework for AI-Augmented Collaboration:** The definition and operationalisation of a role-based interaction model that is augmented by a knowledge-base-grounded LLM recommender, making the collaborative process both traceable and accessible to users of all skill levels.
- **Systematic Empirical Validation:** The demonstration and evaluation of the architecture's core principles through a series of structured experiments that validate end-to-end collaboration, extensibility, and the efficacy of AI-driven knowledge discovery.

This research aims to provide a theoretically grounded and practically validated architectural blueprint for building more inclusive, efficient, and intelligent collaborative data analytics ecosystems.

1.3 Thesis Outline

This thesis is structured to systematically present the research conducted as follows.

Chapter 2 (Literature Review): provides a comprehensive review of existing literature pertinent to collaborative data analytics. It examines current data analytics platforms and their limitations, discusses the challenges in enabling effective collaboration across diverse user groups, explores the role of Artificial Intelligence (particularly LLMs) in transforming analytics, and concludes by articulating the research gaps that motivate the need for a novel architecture like SLEGO.

Chapter 3 (Research Methodology): details the Design Science Research Methodology (DSRM) adopted for this study. It outlines the iterative process of problem identification, objective definition, design and development of the SLEGO artefact, demonstration through case studies, and evaluation, culminating in the communication of research findings.

Chapter 4 (Design and Architecture of the SLEGO System): presents the principles and the detailed architecture of the SLEGO system. This includes the graph-theoretic model of microservices and pipelines, design guidelines for microservice development, the three-layer system architecture (Storage, Service, UI), the schema of the integrated Knowledge Base, and the design of the LLM-Based Recommendation System. User interaction workflows for different roles are also described.

Chapter 5 (Prototype Implementation): describes the practical implementation of the SLEGO prototype, demonstrating how the architectural principles from Chapter 4 are built as a software prototype. This chapter covers the system's technology stack overview. It details how theoretical concepts, such as DAG-based pipelines and the G1–G6 microservice design guidelines, are translated into a concrete execution engine and standardised

Python functions. The chapter concludes with an illustrative walk-through of building an analytical pipeline, showcasing the tailored user interfaces for Technical Experts, Domain Experts, and End Users.

Chapter 6 (Experiments and Evaluation of SLEGO): presents a series of three experiments designed to systematically evaluate SLEGO's capabilities against its core design objectives. Experiment 1 focuses on foundational collaboration and low-code pipeline construction, demonstrating the initial creation of an analytical asset from scratch. Experiment 2 examines pipeline reuse, extension, and knowledge evolution, showcasing how the platform adapts and grows with new, technologically diverse components. Finally, Experiment 3 assesses SLEGO's cross-domain versatility and the operational efficiency gained from its AI-driven recommender system, validating its effectiveness across multiple, diverse analytical domains.

Chapter 7 (Conclusion and Future Directions): concludes the findings from the preceding chapters, evaluates the extent to which SLEGO achieves its stated objectives, and discusses the implications of the research. This chapter acknowledges the limitations of the current study and proposes the direction of future investigation and development.

Through this structured approach, the thesis aims to present a compelling case for the SLEGO architecture as a significant step towards more effective, accessible, and collaborative data analytics.

Chapter 2

Literature Review

This chapter presents a comprehensive literature review that forms the foundation for understanding the landscape of data analytics, the challenges inherent in current practices, and the emerging opportunities for innovative solutions. Section 2.1 defines the area of Collaborative Data Analytics (CDA), exploring its principles, its value proposition, identifying key stakeholders and their roles, and examining the evolving technological landscape that underpins its advancement. It lays the groundwork by establishing what CDA entails and why it is important. Section 2.2 explores the shifting paradigm from traditional, monolithic, expert-driven systems to modern, democratised, and modular data analytics environments. This includes a discussion on the rise of low-code/no-code paradigms and the emergence of the citizen data scientist. Section 2.3 delves into the core challenges encountered in enabling effective collaborative data analytics. It emphasises the difficulties in bridging expertise gaps, fostering common ground, ensuring knowledge capture, reusability, interoperability, and supporting the end-to-end collaborative workflow. Following this, Section 2.4 discusses emerging solutions and the transformative role of Artificial Intelligence (AI), particularly Large Language Models (LLMs), in reshaping data analytics. This section highlights modular and service-oriented architectures, enhanced knowledge management and reusability, and the application of AI and LLMs for intelligent assistance and automation. Section 2.5 provides a comparative analysis of existing

collaborative analytics tools, evaluating visual workflow systems, notebook-based environments, AI-driven platforms, and specialised collaborative platforms to highlight SLEGO's unique position in the current landscape. Finally, Section 2.6 synthesises the identified research gaps in collaborative data analytics platforms. It articulates the shortcomings in integrated AI-guided pipeline construction, systematic reusability, semantic knowledge management, seamless end-to-end collaboration for diverse user roles, and barriers to adoption and customisation of advanced analytics platforms. Together, these sections provide a robust contextual background, justifying the need for the novel architectural solution proposed in this thesis.

2.1 Defining Collaborative Data Analytics (CDA): Principles and Importance

The field of data analytics is undergoing a significant transformation, driven by the exponential growth of data and the increasing complexity of deriving meaningful insights. Central to this evolution is the concept of Collaborative Data Analytics (CDA), which moves beyond individual efforts to harness the collective intelligence and diverse expertise of teams. This section defines CDA, explores its value proposition, identifies key stakeholders and their roles, and examines the evolving technological landscape that underpins its advancement.

2.1.1 Defining Collaborative Data Analytics (CDA)

Collaborative Data Analytics (CDA) can be defined as the joint, iterative process by which multiple individuals, often with heterogeneous expertise and roles, work together using shared data, tools, and methodologies to achieve common analytical goals and generate insights that would be difficult or impossible to achieve in isolation [59, 83, 70]. This process encompasses the entire data lifecycle, from problem formulation and data acquisition through to analysis, interpretation, decision-making, and the dissemination of results

[96]. Unlike individualistic approaches, CDA emphasises shared understanding, division of labour based on expertise, and the integration of diverse viewpoints to produce more robust and comprehensive outcomes [70].

Several platforms and frameworks have begun to embody CDA principles. For instance, Texera[87] enables multiple users to concurrently edit data science workflows and monitor their execution, fostering a real-time collaborative environment akin to "Google Docs for data analytics". Similarly, DataHub[11, 12] supports simultaneous analysis, modification, and exchange of datasets among individuals and teams, addressing challenges of ad-hoc data sharing [12]. These systems highlight a move towards interactive and shared analytical workspaces.

The concept of CDA extends beyond mere tool-sharing; it involves intricate social interactions, knowledge externalisation, and collective sense-making [70]. As discussed by Lazarova-Molnar and Mohamed [48], CDA can involve the "joint processing of data from multiple and diverse sources, by multiple and diverse processors (human or hardware), for achieving synergistic effect". This broad definition includes not only human collaboration but also the interaction of various computational components. Effective CDA environments, therefore, must support not only the technical execution of analytical tasks but also the communication, negotiation, and trust-building processes inherent in teamwork [14].

2.1.2 The Value Proposition: Why Collaborative Data Analytics Matters

The primary value of CDA lies in its potential to enhance the quality, relevance, and impact of data analysis by bringing together diverse perspectives, skills, and knowledge. In scientific research, particularly in complex fields like systems biology, collaborative analysis enables scientists from different disciplines to build comprehensive models and interpret multifaceted data, leading to deeper insights [45]. Similarly, in business contexts, CDA can lead to "better accepted software" and products that genuinely meet customer needs

by integrating feedback throughout the development lifecycle [26].

The ability to "minimise uncertainty and to support analysts' attempts to reach consensus, while still considering all relevant viewpoints" is a key benefit highlighted by Brennan et al. [14] in the context of multi-analyst visual analytics. Effective CDA is crucial for producing trustworthy and actionable solutions, where value emerges from the negotiation and resolution of tensions between different expert viewpoints [70]. For open data initiatives, co-creation through collaborative platforms can unlock "public value," enhancing outcomes, trust, effectiveness, and openness [16].

CDA platforms aim to "accelerate discovery" by helping users find relevant data, tools, and collaborators, and by facilitating the reuse of prior analytic work [38]. This reduces redundancy and allows teams to build upon collective knowledge. In emergency response, CDA is vital for processing diverse information and supporting coordinated decision-making across multiple agencies [83]. The democratisation of analytics, making sophisticated tools and methods accessible to a broader range of users, is another significant value, enabling "casual business users to prepare and analyse data...without being reliant on expert support" and helping power users perform complex tasks more efficiently [63]. Ultimately, by fostering a shared understanding and enabling the integration of diverse expertise, CDA can lead to more robust, innovative, and impactful outcomes from data.

2.1.3 Key Stakeholders and Roles in the Collaborative Analytics Ecosystem

Effective Collaborative Data Analytics (CDA) platforms must recognise and cater to a diverse array of stakeholders, each contributing unique skills, perspectives, and requirements to the analytical process. These roles are not always rigidly defined and can often overlap, but understanding their distinct contributions is crucial for designing supportive and inclusive CDA environments [96]. Based on existing literature and the proposed architecture's intent, we can identify several key stakeholder categories, which are mapped to the proposed system's roles in Table 2.1.

Common roles identified in the literature include:

- **Domain Experts:** Individuals with deep knowledge of the data's origin field (e.g., finance, healthcare, biology, engineering) [58]. They are critical for defining problems, interpreting results, and ensuring practical applicability [96].
- **Data Scientists/Analysts:** Technical experts skilled in modelling, machine learning, programming, and visualisation [72]. Responsible for method selection, model implementation, and data processing [96].
- **Data Engineers/IT Professionals:** Focused on data infrastructure: acquisition, storage, management, security, and platform scalability [42].
- **Business Users/Decision-Makers:** Primary consumers of insights—managers, executives, or policymakers who need analytics to make decisions. Their feedback ensures analyses deliver organisational value [44].
- **Citizen Data Scientists:** Users outside formal analytics roles, empowered by low-code/no-code platforms to perform advanced analyses [44].
- **Software Developers/Technical Experts:** Develop, validate, and maintain analytical tools/services that others use [72].
- **Communicators/Visualisation Experts:** Specialise in translating findings into understandable narratives and visuals, especially for non-technical audiences [21, 61].

To streamline the interaction within the collaborative environment, we assume that a solution architecture needs to consolidate these diverse stakeholder categories into three primary system roles: *Technical Experts*, *Domain Experts*, and *End Users*. Table 2.1 illustrates the mapping.

Table 2.1: Common User Roles in Data Analytics and Mapping to Assumed System Roles

Common User Role(s) in Literature	**Assumed System Role(s) in This Thesis**	**Key Responsibilities in System**	**Relevant Literature Examples**
Data Scientist, ML Expert, Data Engineer, Software Developer, IT User, Algorithm Developer	Technical Expert	Develops, validates, and maintains analytical microservices; ensures technical robustness and efficiency; contributes to platform infrastructure.	[96, 72, 42]
Domain Expert, Subject Matter Expert, Business Analyst (analytics-focused)	Domain Expert	Provides domain knowledge, annotates microservices, defines best-practice templates, validates approaches, interprets results.	[58, 96]
Business User, Decision-Maker, Casual User, Novice Analyst, Citizen Data Scientist	End User	Uses low-code GUI and AI recommendations to build/execute pipelines; focuses on business/research goals; interprets results for action.	[44]
Project Manager, Manager/Executive (consuming insights)	End User / Domain Expert	Defines analytical goals, oversees projects, consumes insights, ensures strategic alignment.	[96, 70]
Communicator, Visualisation Expert	Domain Expert / End User	Presents results to stakeholders via dashboards, reports, or narratives.	[21, 61, 96]

This mapping underscores the need for a flexible platform that provides different interfaces, tools, and levels of abstraction to cater to each role, fostering a collaborative and productive analytics ecosystem [96].

2.2 The Shifting Paradigm: From Traditional to Modern Data Analytics Environments

The way organisations and individuals approach data analytics is fundamentally changing. This shift is driven by the increasing volume and complexity of data, the demand for faster and more agile insights, and the availability of new technologies. This section explores two key aspects of this paradigm shift: the move from monolithic, expert-driven systems to democratised, modular platforms, and the rise of low-code/no-code approaches empowering citizen data scientists.

2.2.1 The Evolution from Monolithic, Expert-Driven Systems to Democratised, Modular Platforms

Traditionally, data analytics was largely the domain of specialised professionals—dedicated data science teams, statisticians, or IT departments—who operated complex, often monolithic, software systems like traditional BI suites or statistical packages [44, 41]. Business users or domain experts typically had to submit analysis requests and await results, leading to potential bottlenecks, communication overhead, and a disconnect between those possessing contextual domain knowledge and those with the technical analytical skills [63]. These traditional systems, while powerful for experts, were often characterised by:

- **Limited Flexibility and Extensibility:** Monolithic architectures made it cumbersome and time-consuming to integrate new data sources, incorporate novel algorithms, or customise existing workflows to meet evolving needs [94].
- **Scalability Challenges:** Scaling these systems to handle burgeoning data volumes or increasing user loads often proved complex and costly [84].
- **Lack of Agility:** The centralised model frequently struggled to keep pace with the dynamic and ad-hoc analytical requirements of business users needing rapid insights [63].

The modern paradigm in data analytics is increasingly characterised by democratisation, modularity, and interactivity. This involves decomposing complex analytical processes into smaller, manageable, and often reusable components, and providing tools and platforms that empower a broader spectrum of users to engage directly with data and analytics [84]. Several key technological and methodological trends exemplify this evolution:

- **Component-Based and Service-Oriented Architectures:** The adoption of Component-Based Software Engineering (CBSE) principles and, more recently, Microservice Architectures (MSA), has been pivotal [5]. MSA promotes the development of small, independent, and deployable services, each responsible for a specific analytical function (e.g., data ingestion, a particular transformation, a specific model training algorithm) [2]. These microservices can be composed into complex analytical pipelines, offering enhanced flexibility, scalability, and reusability. This approach allows different teams to develop and maintain services independently, using diverse technologies if needed.

- **Visual Workflow Systems:** Platforms like Taverna[1] [68], KNIME[2] [25], RapidMiner[3] [60], and Alteryx[4] [3] have enabled users to construct sophisticated data analysis pipelines visually by connecting nodes or operators that represent specific analytical tasks. This visual paradigm abstracts much of the underlying code, making advanced analytics accessible to users who are not expert programmers, thereby fostering end-user data analytics [41].

  Systems like Texera[5] further enhance this by supporting real-time collaborative editing and interactive execution of these visual workflows [10, 87, 24].

- **Cloud-Native Platforms:** Cloud computing provides the elastic and scalable infrastructure essential for modern data analytics. Cloud-based platforms offer on-demand

[1] `https://incubator.apache.org/projects/taverna.html/`

[2] `https://www.knime.com/`

[3] `https://rapidminer.com/`

[4] `https://www.alteryx.com/`

[5] `https://texera.io/`

access to storage, processing power, and a rich ecosystem of analytical services (e.g., ML platforms, data warehousing services) [26]. This has significantly lowered the barrier to accessing powerful analytical capabilities and facilitated the development of collaborative, web-accessible analytics environments [88]. Major cloud platforms include AWS[6], Microsoft Azure[7], and Google Cloud Platform (GCP)[8].

- **Open Source Ecosystems:** The widespread adoption of open-source programming languages (e.g., Python, R) and libraries (e.g., Pandas[9], Scikit-learn[10], TensorFlow[11], PyTorch[12]) has substantially democratised access to advanced data analytics capabilities. These ecosystems enable practitioners to build on robust, community-maintained foundations, fostering innovation, reducing development time, and lowering costs [30, 47].

This ongoing shift is fundamentally about making data analytics more agile, accessible, and aligned with the diverse needs of users and the dynamic nature of modern data challenges.

2.2.2 The Rise of Low-Code/No-Code Paradigms and the Citizen Data Scientist

A significant driver and manifestation of the democratisation trend in data analytics is the ascent of **Low-Code/No-Code (LC/NC)** platforms [13] and the concurrent emergence of the citizen data scientist [62]. Citizen data scientists are defined as individuals whose primary role is outside of statistics or advanced analytics but who can generate

[6] `https://aws.amazon.com/`

[7] `https://azure.microsoft.com/`

[8] `https://cloud.google.com/`

[9] `https://pandas.pydata.org/`

[10] `https://scikit-learn.org/`

[11] `https://www.tensorflow.org/`

[12] `https://pytorch.org/`

models using advanced diagnostic analytics or predictive and prescriptive capabilities, often leveraging these user-friendly platforms [62]. LC/NC platforms empower these users by providing intuitive, often visual, interfaces that abstract the complexities of underlying programming languages, algorithms, and data infrastructure [41, 13].

Key characteristics and implications of this trend include:

- **Visual Development and Workflow Construction:** Many LC/NC analytics platforms offer drag-and-drop interfaces where users construct analytical workflows by connecting predefined modules or operators representing specific tasks (e.g., data loading, filtering, model training). This visual paradigm significantly lowers the technical barrier, allowing users to focus on the analytical logic and problem-solving rather than intricate coding details [51, 25, 41].
- **Automation of Data Science Tasks:** These platforms often automate various stages of the traditional data science lifecycle, including aspects of data preparation, feature engineering, model selection (AutoML), and even model deployment. This automation further reduces the need for specialised expertise for common tasks. [31, 54]
- **Natural Language Interfaces (NLIs):** A more recent and transformative development is the integration of Large Language Models (LLMs) to enable interaction with data and analytical tools via natural language. Users can describe their analytical goals or ask questions in plain language, and the system translates these into executable queries or workflows. This makes advanced analytics accessible to an even broader audience, including those with no prior exposure to visual programming or structured query languages [35, 52, 99].
- **Increased Agility and Speed of Prototyping:** By simplifying and accelerating the process of building and testing analytical models, LC/NC platforms allow organisations to be more agile in responding to new opportunities or challenges. Rapid prototyping of analytical solutions becomes feasible even for users who are not professional developers [13].

However, the rise of LC/NC and citizen data science is not without its challenges and considerations:

- **Risk of Misapplication and Misinterpretation:** Users with limited statistical or methodological understanding might misapply analytical techniques or misinterpret results generated by "black box" models, potentially leading to flawed conclusions [51].
- **Need for Governance and Quality Control:** As more users generate analytical artefacts, ensuring data quality, model validity, and adherence to organisational best practices and ethical guidelines becomes critical [63].
- **Balancing Simplicity with Power:** There is an inherent tension in designing platforms that are simple enough for novices yet powerful and flexible enough for experts. Over-simplification can limit the scope of analysis, while too much complexity can deter non-technical users [41].
- **Importance of Foundational Understanding:** Even with LC/NC tools, a basic "programming mindset" or "analytical thinking" and troubleshooting skills are beneficial for users to effectively leverage these platforms and critically evaluate their outputs [98].

Despite these challenges, LC/NC platforms, especially when enhanced by AI and LLMs, are fundamentally reshaping the landscape of data analytics. They are key enablers for democratising access to data-driven insights, fostering a more data-literate workforce, and enabling organisations to more broadly embed analytics into their decision-making processes.

2.3 Core Challenges in Enabling Effective Collaborative Data Analytics

Achieving the full potential of collaborative data analytics is challenging: beyond shared tools, teams must overcome human, organisational, and technical barriers—especially in communication, knowledge management, and process integration

2.3.1 Bridging Expertise Gaps and Fostering Common Ground

A fundamental challenge in CDA is the inherent diversity of expertise among collaborators. Data science projects often involve individuals from varied backgrounds, such as data scientists with deep technical and statistical knowledge, domain experts with rich contextual understanding of the data and problem area, and business stakeholders focused on actionable outcomes [72, 96, 14, 83]. This diversity, while a strength, also creates significant communication barriers. Mao et al. [58] illustrate this "different language" problem between data scientists and bio-medical scientists, where establishing and maintaining "common ground"—a shared understanding of goals, data, methods, and interpretations—is a continuous and arduous process.

This expertise gap manifests in several ways:

- **Misinterpretation of Terminology and Concepts:** Technical jargon used by data scientists may be opaque to domain experts, and conversely, nuanced domain knowledge may be lost in translation when communicated to technical teams [72].
- **Differing Goals and Priorities:** Data scientists might prioritize model accuracy or algorithmic novelty, while domain experts may focus on interpretability, actionability, or the ability to ask new, evolving questions [58].
- **Black Box Phenomenon:** Complex models or analytical processes can appear as "black boxes" to non-technical collaborators, hindering their ability to trust, validate, or contribute meaningfully to the analysis [70, 72].
- **Difficulty in Specifying Requirements:** Translating high-level business or research questions into well-defined, computable analytical tasks is a significant hurdle, particularly for those unfamiliar with the capabilities and limitations of data analytics techniques [44].

Platforms aiming to support CDA must actively facilitate the bridging of these gaps. This involves features that promote clear communication, make complex processes transparent,

and help establish a shared understanding among all participants.

2.3.2 Ensuring Knowledge Capture, Reusability, and Interoperability in Collaborative Settings

Effective collaboration relies on the ability to capture, share, and reuse knowledge and analytical assets. However, this presents several challenges in practice:

- **Knowledge Capture:** A significant amount of knowledge generated during data analysis—such as the rationale behind data cleaning decisions, feature engineering steps, model selection criteria, or the interpretation of intermediate results—often remains tacit or poorly documented. This "invisible work" makes it difficult for others (or even the original analyst at a later time) to understand, validate, or build upon previous analyses [96, 93, 56, 70].
  - *Finding Reusable Assets:* Developers and analysts often struggle to find relevant existing components (code, libraries, models, or even entire workflows) due to inadequate repositories, poor search mechanisms, or lack of awareness. Opportunistic reuse, while common, can lead to the integration of poorly understood or low-quality components [95, 76, 57].
  - *Assessing Reusability:* Determining whether a component is truly suitable for reuse in a new context is challenging. This involves assessing not just its functional fit but also its quality, reliability, adaptability, and understandability. Without systematic evaluation, reusability remains a hit-or-miss affair [1, 69, 100].
  - *Maintaining Reusability:* Software artifacts can suffer "reusability decay" over time due to changes in dependencies, data formats, or the underlying platform, making them difficult to run or adapt. The FAIR principles for data, software, and workflows aim to combat this by promoting practices that enhance long-term reusability [9, 90, 91, 64].
- **Interoperability:** Collaborative projects often involve integrating data from diverse

sources and tools developed with different technologies. Ensuring these components can connect to each other—both semantically and technically—is a major hurdle. Semantic interoperability, in particular, requires shared understanding of data meaning, which can be facilitated by ontologies or well-defined data contracts [4, 7, 85, 4, 40].

Platforms supporting CDA must provide robust mechanisms for knowledge capture (e.g., through structured documentation, provenance tracking), facilitate the discovery and assessment of reusable assets, and promote technical and semantic interoperability between different analytical components and data sources.

2.3.3 Supporting the End-to-End Collaborative Workflow

Data analytics is not a linear process but an iterative cycle often involving multiple stages, from problem definition and data acquisition to modeling, evaluation, deployment, and communication of results [96]. Effective collaboration needs to be supported across this entire workflow.

Key challenges include:

- **Fragmented Tooling:** Teams often use a disparate set of tools for different stages of the workflow (e.g., databases for storage, scripting languages for processing, visualisation tools for exploration, presentation software for reporting). This fragmentation can lead to inefficiencies, data transfer issues, and loss of context [46].
- **Lack of Integrated Collaboration Features:** Many analytics tools, while powerful for individual use, lack built-in features for real-time co-editing, shared execution monitoring, or collaborative annotation and sensemaking [88].
- **Managing Handovers:** As work transitions between different team members or roles across workflow stages (e.g., from a data engineer preparing data to a data scientist building a model), crucial context and rationale can be lost if not properly documented and communicated [93] .

- **Iterative Refinement and Feedback Loops:** The exploratory nature of data analysis means that initial plans often change, requiring iterative refinement of goals, methods, and interpretations [58]. Platforms need to support this dynamic process, allowing collaborators to easily revisit previous steps, explore alternatives, and incorporate feedback. [61, 58]
- **Communication with Diverse Stakeholders:** The end-to-end workflow often culminates in communicating findings to stakeholders who may not be data experts (e.g., managers, clients). Tools must support the creation of clear, understandable, and actionable reports or visualisations [72].

Addressing these challenges requires platforms that offer a more integrated and holistic environment, supporting seamless transitions between workflow stages and providing consistent collaborative functionalities throughout the entire data analytics lifecycle.

2.4 Emerging Solutions and the Role of AI in Data Analytics

To address the multifaceted challenges in collaborative data analytics, various emerging solutions and technological advancements are paving the way for more integrated, intelligent, and user-friendly platforms. A significant driver in this evolution is the increasing integration of Artificial Intelligence (AI), particularly Large Language Models (LLMs).

2.4.1 Modular and Service-Oriented Architectures

The shift towards modular architectures, such as microservices, is a key trend in building flexible and scalable data analytics platforms [5, 84]. This approach involves decomposing complex analytical processes into smaller, independent, and reusable services that can be composed into custom pipelines. Such architectures enhance maintainability and allow for

the independent evolution of different analytical components [2]. The performance characteristics of inter-service communication in these architectures are also an active area of research, with event-driven architectures (EDA) often showing advantages over synchronous API-driven approaches for certain workloads [74]. Frameworks like ADAGE [94] and the DASE framework [6] propose structured, service-oriented approaches for designing data-intensive science and analytics platforms, emphasising the importance of well-defined interfaces and knowledge integration.

2.4.2 Enhanced Knowledge Management and Reusability

There is a growing recognition that effective analytics relies on robust knowledge management. This includes not only managing data but also the analytical artefacts (models, workflows, code) and the contextual knowledge surrounding them.

The FAIR principles (Findable, Accessible, Interoperable, Reusable) [90] offer a foundational guideline for managing research data and have been extended to research software [9, 30] and computational workflows [91]. Adhering to these principles makes analytical assets more discoverable, understandable, and reusable.

Systems like the Ontology Service Center (OSC) [80] and approaches like BIGOWL4DQ [8] demonstrates the use of ontologies and semantic web technologies to formalise domain knowledge, data quality rules, and analytical requirements, thereby improving interoperability and shared understanding. The VCU (Variation, Customisation, Usage) model [43] provides a valuable framework for designing reusable software components by clearly defining their different interaction interfaces.

Furthermore, research into measuring and predicting software reusability using data-driven approaches [69, 100, 1] and mining library co-usage patterns [76] offers methods to quantify and enhance the reusability of analytical components. For knowledge summarised by others, Liu et al. (2021) [53] propose the Strata system, which helps consumers evaluate the context, trustworthiness, and thoroughness of such summarised knowledge before reuse.

2.4.3 AI and LLMs for Intelligent Assistance and Automation

The most transformative recent development is the application of AI, especially LLMs, to various aspects of the data analytics lifecycle.

Automated Workflow Generation and Recommendation LLMs are being explored to translate natural language instructions into visual programming pipelines such as InstructPipe [99] or declarative AI analytics workflows like PALIMPCHAT [52]. This significantly lowers the barrier for non-programmers to construct complex analytical processes. Similarly, recent tutorials on LLM for Data Management [50] outline how LLM agents can perform task decomposition and pipeline orchestration.

Code Generation and Assistance Tools like GitHub Copilot[13] and systems like INCODER [27] leverage LLMs for code synthesis and infilling, assisting developers in creating and modifying analytical scripts or software components [77, 67]. However, responsible reuse of AI-generated code, considering licensing and privacy, remains a challenge [39].

Knowledge Discovery and Augmentation LLMs are used to automate knowledge discovery from scientific literature, Systems such as MRAgent [92] adopt multi-agent workflows with progressive ontology prompting to surface causal relationships and LLM-Duo [34] uses a multi-agent approach with progressive ontology prompting for causal discovery. Retrieval-Augmented Generation (RAG) [49] techniques are crucial for grounding LLM outputs in factual knowledge, enhancing their reliability for knowledge-intensive tasks. The Data-Juicer system [18] demonstrates a comprehensive platform for preparing high-quality data for LLM training, itself an AI-assisted workflow.

Enhanced Interaction and Explanation LLMs can power natural language interfaces for querying data and analytical systems (e.g., JarviX by Liu et al. (2023) [54];

[13] `https://github.com/features/copilot`

DataChat by John et al. (2023) [35]). Moreover, the ability of LLMs to generate explanations for complex models or results is a key area for improving trust and understanding in analytics [70]. Techniques like multi-agent debate are even being explored to improve LLM factuality and reasoning [22].

Optimisation Research is also emerging on optimising LLM query processing itself, especially for relational data analytics workloads, to improve efficiency and reduce costs [55].

These emerging solutions, particularly those leveraging AI, offer powerful mechanisms to address many of the traditional challenges in data analytics. However, their effective integration into comprehensive, user-friendly, and collaborative platforms remains an active area of research and development [89, 92].

2.5 Comparative Analysis of Collaborative Analytics Tools

The landscape of data analytics tools is rich and varied, with platforms often excelling in specific areas like visual workflow construction, programmatic flexibility, or AI-driven automation. This section provides a comparative analysis of existing systems and foundational concepts, structured around the key features required for a modern collaborative analytics platform.

2.5.1 Low-Code/No-Code Interfaces and Visual Workflows

A primary trend in democratising data analytics is the adoption of Low-Code/No-Code (LC/NC) interfaces, which lower the technical barrier for users without deep programming expertise. Visual workflow systems are the most common manifestation of this paradigm. Platforms like **KNIME** [25, 81], **RapidMiner** [60], and **Alteryx** [3] allow users to construct sophisticated data analysis pipelines by visually connecting nodes or operators in a

drag-and-drop environment. This approach abstracts away the underlying code, enabling "citizen data scientists" to focus on analytical logic [62, 13]. **Texera**[56, 87] further enhances this model by supporting real-time collaborative editing of these visual workflows, making the process interactive and shared . Newer platforms like **DataChat** [35] and **JarviX** [54] push this further by enabling pipeline construction through natural language or voice commands, creating a no-code experience.

2.5.2 Modular Design and Extensibility

Modern analytics platforms are increasingly moving away from monolithic designs towards modularity, which is crucial for flexibility and reusability. This is exemplified by the adoption of Microservice Architectures (MSA) [5, 84]. Visual workflow systems like **KNIME** [25] and **RapidMiner** [60] achieve modularity through their "node" or "operator" based designs, where each component performs a specific function. Specialised platforms like **Galaxy** [82] and **BioUML** [45] also rely on a modular "tool wrapping" system to integrate a vast number of bioinformatics tools into shareable workflows. Platforms like **Azure ML Studio**[14] [41] use a component-based system for building machine learning pipelines . This modularity is a foundational principle that allows for the composition of complex processes from smaller, independent, and often reusable parts.

2.5.3 Support for Flexible Programming

While LC/NC interfaces are important for accessibility of end users, supporting programmatic access is vital for extensibility and catering to technical experts. Many leading platforms adopt a hybrid approach. **KNIME** [25] and **RapidMiner** [60], for instance, allow users to integrate custom scripts written in R, Python, or Java directly into their visual workflows, enabling experts to create custom nodes or perform tasks not covered by standard operators. **Texera** [87] supports custom logic through Python User-Defined

[14]https://azure.microsoft.com/en-au/products/machine-learning

Functions (UDFs), and **DataChat** [35] provides a Python API alongside its natural language interface . At the other end of the spectrum, environments like **Jupyter**[15] [73] are primarily programmatic, offering maximum flexibility for expert data scientists [96].

2.5.4 Knowledge Management and Reusability

Effective collaboration and reuse depend on robust knowledge management. Platforms like **DataHub** [11] introduced data-centric versioning to address this. More advanced approaches are emerging. Knowledge-driven design frameworks like **DASE** [6] and **BIGOWL4DQ** [8] demonstrate the power of using ontologies to formalise requirements and domain knowledge. **JarviX** utilises a vectorised knowledge repository to provide context-aware explanations [54]. The FAIR principles provide a foundational guideline for making analytical assets (data, software, workflows) more findable, accessible, interoperable, and reusable, but their implementation is still maturing [90, 9, 91].

2.5.5 AI-Driven Guidance and Automation

A transformative trend is the deep integration of AI, particularly LLMs, to provide intelligent assistance. This moves beyond the simple automation of tasks (like in AutoML) towards proactive guidance in the analytical process. Platforms like **DataChat** [35], **JarviX** [54], and **InstructPipe** [99] leverage LLMs to translate natural language user intent into executable workflows or pipeline structures. Specialised agents like **MRAgent** [92] demonstrate how an LLM can orchestrate a complex, multi-tool scientific workflow from start to finish. This AI-driven guidance significantly lowers the barrier to entry for complex analytics. However, these systems are often tailored to specific interaction modes (e.g., NL-to-recipe) or domains, and do not yet offer guidance from a generalised, extensible ecosystem of analytical components.

[15] https://jupyter.org

2.5.6 Role-Based Collaboration and Specialisation

Effective collaboration requires support for diverse user roles and their interactions. Ethnographic studies of data science teams reveal a complex interplay between technical experts, domain experts, managers, and communicators, each with different goals and information needs [72, 58, 96]. Platforms like **Texera**[87] are pioneering real-time collaborative editing and execution monitoring, akin to "Google Docs for data analytics," which facilitates close collaboration between different roles . **DataChat**[35] also supports live collaborative sessions for co-creating analytical artefacts . However, many platforms lack explicit modelling of user roles and tailored interfaces that support the distinct contributions required at each stage of the collaborative lifecycle, from component creation and curation to pipeline assembly and interpretation.

2.5.7 Genericity vs. Domain Specialisation

Analytics platforms often face a trade-off between being a generic, all-purpose tool and a highly effective domain-specific solution. Platforms like **KNIME** [25], **RapidMiner** [60], and **Jupyter** [73] are designed to be generic, applicable across many fields. In contrast, platforms like **Galaxy** [82] and **BioUML** [45] have achieved great success by focusing on the specific needs, tools, and data types of the bioinformatics community. Similarly, agents like **MRAgent** [92] are powerful but highly specialised for Mendelian Randomisation analysis. The challenge for a generic platform is to provide the flexibility to handle diverse domains without sacrificing the depth and usability required for specialised tasks.

2.5.8 Foundational Concepts and Broader Challenges

Effective collaborative analytics also hinges on addressing human and organisational factors. This includes overcoming resistance to new tools [78], building trust in complex data science processes [70], managing knowledge handovers effectively [93], and streamlin-

ing practical collaboration workflows [46]. The FAIR principles [90, 9, 91] provide crucial guidance for creating reusable and interoperable analytical assets. Methodologies for defining and assessing reusability [43, 100, 69], along with understanding the implications of AI in code generation and reuse [39, 27, 77], are also essential considerations. Architectural frameworks like ADAGE [94] and knowledge-driven design approaches like DASE [6] and BIGOWL4DQ [8] inform the creation of robust and maintainable analytics platforms. Tools like **Data-Juicer** (LLM data prep) [18] are highly specialised and do not offer a generic platform for user-defined pipeline construction with LLM guidance from a diverse microservice ecosystem.

The preceding review of existing systems and foundational concepts is summarised in Table 2.2. To provide a framework for this comparison, the platforms were evaluated against the following criteria, which reflect the core challenges in modern data analytics:

1. **Low-code/No-code:** The primary user interaction model, assessing support for non-programmers via visual tools or natural language.
2. **Modular and Microservice Design:** The architectural foundation for component reusability and extensibility, such as through a microservice or node-based design.
3. **Flexible Programming:** The platform's extensibility for technical experts through custom coding, such as via integrated scripts or APIs.
4. **Knowledge Base:** The system's capability to capture, manage, and reuse analytical knowledge in a central repository, such as component metadata and pipeline templates.
5. **AI/LLM Guidance:** The integration of AI to proactively guide users in the discovery and construction of analytical pipelines.
6. **Role-based Collaboration:** Support for teamwork between diverse user personas (e.g., technical experts, domain experts) through shared, interactive features.
7. **Generic Analytics:** Whether the platform is a general-purpose, domain-agnostic tool or is specialised for a particular field.

This comparative analysis highlights that while many platforms address parts of the collaborative analytics puzzle, no single solution provides a seamless, integrated framework that holistically supports diverse users with AI-driven guidance. The following section will discuss these identified gaps based on the findings.

Tool/ Platform	Low-code/No-code	Modular and microservice Design	Flexible Programming	Knowledge Base	AI/LLM Guidance	Role-based Collaboration	Generic Analytics
DataChat	**YES** Natural language + point-click	**YES** ~50 skills architecture	**YES** Python API + GEL	**PARTIAL** Recipe sharing only	**YES** LLM NL2Code system	**YES** Live collaborative sessions	**YES** Full spectrum analytics
Texera	**YES** Visual DAG interface	**YES** Actor-based operators	**YES** Python UDF support	**NO** No knowledge management	**NO** No AI features	**YES** Real-time Google Docs-like	**YES** Generic workflows
KNIME	**YES** Visual workflow editor	**YES** Node-based modularity	**YES** R/Python/Java support	**YES** Community Hub	**Yes** Newly released in 2023[16]	**PARTIAL** Basic team server	**YES** Comprehensive platform
JarviX	**YES** Voice/text no-code interface	**PARTIAL** H2O-AutoML pipeline	**NO** No programming support	**YES** Vectorised knowledge repo	**YES** Vicuna + GPT-4	**NO** Individual use only	**PARTIAL** Tabular data focus
InstructPipe	**YES** Natural language instructions	**YES** Visual Blocks node-based	**NO** Limited to predefined nodes	**PARTIAL** Predefined node library only	**YES** LLM pipeline generation	**NO** Individual use with AI	**PARTIAL** ML pipeline focus
DataHub	**PARTIAL** (via apps)	**YES** App ecosystem	**YES** Python/R/etc. via Thrift + Notebook	**YES** Dataset version graph	**NO** (Pre-LLM era)	**YES** Novice & Expert roles	**YES** General purpose
Data-Juicer	**YES** Zero/low-code recipes	**YES** Composable operator pool	**YES** Python for new operators	**YES** Recipes & operator library	**YES** AI quality classifiers, HPO	**PARTIAL** Novice to expert users	**PARTIAL** LLM data prep focus
Galaxy	**YES** Web-based workflow builder	**YES** Tool wrapping system	**YES** Multiple language support	**YES** Workflow registry system	**NO** No AI assistance	**YES** Multi-user platform	**PARTIAL** Bioinformatics focus
RapidMiner	**YES** Visual process designer	**YES** Operator-based modularity	**YES** R/Python/ Groovy support	**PARTIAL** Hub for sharing	**Yes** AI-driven automation	**PARTIAL** Server-based sharing	**YES** Enterprise analytics
Alteryx	**YES** Visual workflow designer	**YES** Tool-based modularity	**YES** R/Python integration	**PARTIAL** Workflow sharing	**Yes** Alteryx Copilot	**PARTIAL** Server collaboration	**YES** Data preparation focus
Azure ML Studio	**YES** Drag-and-drop interface	**YES** Component-based system	**YES** Python/R script support	**PARTIAL** Model/dataset sharing	**YES** AI tools for guidance	**PARTIAL** Basic collaboration	**YES** ML platform
Tableau	**YES** No-code visualisation	**NO** Limited modularity	**PARTIAL** R/Python integration	**NO** No knowledge base	**PARTIAL** NL queries only	**PARTIAL** Dashboard sharing	**PARTIAL** BI-focused only
H2O-AutoML	**PARTIAL** Automated ML only	**YES** Pipeline modularity	**YES** Python/R APIs	**NO** No knowledge base	**NO** Rule-based only	**NO** Individual use only	**PARTIAL** ML-focused only
Jupyter	**NO** Code-based only	**NO** Cell-based limited	**YES** Multi-kernel support	**NO** No knowledge base	**NO** No AI features	**PARTIAL** Real-time collab (new)	**YES** General computation

Table 2.2: Comprehensive feature comparison of analytics tools and platforms.

2.6 Research Gaps in Collaborative Data Analytics

Despite significant advancements in data analytics tools, methodologies, and AI integration, as evidenced by the diverse platforms reviewed above, several critical research gaps persist. These gaps hinder the realisation of truly democratised, efficient, and comprehensively collaborative data analytics ecosystems. The preceding comparative analysis highlights a landscape ripe with opportunity but also marked by fragmentation and unmet needs, particularly in seamlessly supporting the entire collaborative analytics lifecycle for diverse user groups with AI-driven intelligence.

1. **Limited Support for Seamless, End-to-End Collaboration Among Diverse User Roles in Building and Evolving Complex Analytics** The literature confirms that modern data science is an inherently collaborative, multidisciplinary effort involving individuals with varied expertise, including technical developers, domain specialists, and end-users [96, 72]. However, existing platforms often fail to holistically support the entire collaborative lifecycle. While some tools excel in specific collaborative aspects, such as Texera for real-time workflow co-editing [56, 87] or DataHub for data versioning [12], there is a dearth of platforms that seamlessly integrate the contributions of all roles from initial component creation to final analytical insight.

   Significant hurdles remain in bridging the "different language" problem and establishing "common ground" between technical and non-technical stakeholders [58, 70]. Communication challenges [72], difficulties in merging diverse contributions [46], and managing knowledge handovers [93] are persistent pain points. This points to a clear gap for a platform that explicitly models and facilitates the distinct contributions and interactions of a diverse user base—such as Technical Experts defining and implementing microservices, Domain Experts curating them with contextual knowledge, and End Users applying them to solve problems—within a single, traceable, end-to-end workflow.

2. **Insufficient Support for Systematic Reusability and Semantic Knowledge**

Management in Heterogeneous Component Ecosystems The effective reuse of analytical components is paramount for efficiency and consistency [17]. While platforms like Galaxy [82] and BioUML [45] have demonstrated the power of tool repositories in specific domains, a generalised approach to managing and reusing heterogeneous analytical components (akin to microservices) with rich semantic descriptions is less mature. The literature highlights the challenges of "reusability decay" [64] and the difficulty in objectively assessing component reusability based on multifaceted factors like documentation, complexity, and reliability [100, 69].

Adherence to the FAIR principles [90] for research software and workflows—which enhance findability, accessibility, interoperability, and reusability—is gaining traction but is not yet a standard feature in most analytics platforms for their internal components [9, 91]. Many systems still lack a central, semantic Knowledge Base that describes analytical components with the richness required to support systematic reuse. Such a knowledge base must capture not only technical specifications but also provenance, usage contexts, and domain-specific annotations to power effective discovery, validation, and composition of components [94, 6].

3. **Lack of Integrated, AI-Guided Pipeline Construction for Diverse Users in Generic Contexts** While many visual workflow tools offer Low-Code/No-Code (LC/NC) interfaces for pipeline construction, they typically require users to manually select and connect predefined nodes or operators. Proactive AI-driven guidance for *constructing* these pipelines from a broader, extensible component base is often missing or limited to specific sub-tasks like auto-modeling [41]. Newer AI-driven platforms like DataChat [35] and InstructPipe [99] demonstrate the potential of LLMs for translating natural language to workflows or generating pipeline structures. However, DataChat's "skills" are high-level abstractions rather than an open ecosystem of general-purpose microservices, and InstructPipe focuses on ML pipelines within a specific visual environment using a predefined node to generate new codes instead of suggesting existing components. Specialised agents like MRAgent [92] automate fixed scientific workflows but are not designed for generic, user-driven pipeline construction across diverse analytical domains. There remains a clear gap for a **generic-**

purpose analytics platform that deeply integrates an LLM-based recommender system to proactively assist a wide spectrum of users (from novices to experts) in composing complex analytical pipelines from a dynamic and extensible set of modular components (microservices).

These gaps underscore the need for continued innovation in developing data analytics platforms. An ideal platform should not only be powerful and versatile but also accessible and intuitive for users of all skill levels. It must foster seamless, role-based collaboration, promote systematic reuse of well-described and FAIR analytical assets, and intelligently leverage AI to enhance the entire analytics lifecycle, from problem formulation and graph-based pipeline construction to insight generation and knowledge dissemination. Addressing these multifaceted gaps is critical for unlocking the full potential of data analytics in both research and industry, motivating the development of a system like SLEGO.

2.7 Conclusion

This literature review has surveyed the evolving landscape of data analytics, revealing a field rich with innovation yet marked by persistent, systemic challenges. The shift towards modular, low-code systems and the rise of Artificial Intelligence have opened new frontiers. However, as established in the preceding analysis, no existing solution holistically addresses the interconnected challenges of bridging expertise gaps, ensuring systematic knowledge reuse, and overcoming fragmented tooling. The review culminates in the identification of a set of critical, interrelated research gaps:

- Limited end-to-end collaboration frameworks that model diverse user roles.
- Modular analytics components lack adequate support for systematic reusability and semantic knowledge management.
- A lack of integrated, AI-guided pipeline recommendation in general purpose platforms.

The core argument emerging from this review is that these gaps cannot be solved in isolation. A truly effective platform must be architected as an ecosystem designed to foster the cycle of creation, curation, and consumption. Addressing these gaps therefore requires a novel architectural approach which will be discussed in the next Chapter.

Chapter 3

Research Methodology

Based on the findings of the literature review in Chapter 2, this chapter discusses the research methodology followed in the development and evaluation of the **SLEGO** platform. Section 3.1 presents the identified research problems, and Section 3.2 explores their interconnected nature to argue for the necessity of a holistic solution. In Section 3.3, we introduce the design science research methodology (DSRM) and the steps that will be part of the research process. Subsequently, Section 3.4 shows how these DSRM steps are applied throughout this thesis to design, implement, and evaluate SLEGO platform. Finally, Section 3.5 concludes the chapter.

3.1 Identified Research Problems

Building on the literature review summary in Section 2.6, this research addresses three interconnected problems hindering collaborative data analytics effectiveness. Existing platforms tackle individual aspects, but a holistic solution is lacking.

1. **Lack of End-to-End Collaboration Support for Diverse Users:** A fundamental challenge is the lack of platforms that effectively support the entire collaborative

lifecycle for teams with varied expertise. Data science is an inherently multidisciplinary effort involving technical developers, domain specialists, and end-users. However, current tools often create a divide, forcing a trade-off between powerful but complex coding environments and inflexible low-code interfaces. This fragmentation leads to communication barriers and inefficient workflows, as there is a lack of "common ground" between stakeholders. The result is a clear need for a platform that can manage the distinct contributions of diverse user groups within a single, traceable, and end-to-end workflow.

2. **Insufficient Support for a Modular and Extensible Architecture:** Organisations consistently struggle with the systematic management and reuse of analytical assets, which leads to inefficiency and duplicated effort. Many current solutions lack a truly modular architecture, making them difficult to scale and maintain. While platforms in specific domains have shown the value of tool repositories, a generalised approach for managing heterogeneous analytical components with rich semantic descriptions is not yet mature. There is a deficiency in systems that incorporate a central, semantic Knowledge Base to capture not only the technical specifications of components but also their usage context and domain-specific annotations, which is essential for effective discovery and reuse.

3. **Absence of Integrated, AI-Guided Knowledge Discovery:** A significant gap exists in providing intelligent assistance to help users navigate the complexity of modern analytics platforms. While some visual workflow systems offer low-code interfaces, they typically lack proactive, AI-driven guidance for constructing pipelines from an extensible component base. Newer AI-driven tools demonstrate the potential of Large Language Models (LLMs) to translate natural language into workflows; however, they are often specialised and do not operate on an open ecosystem of general-purpose microservices. There is a distinct need for a generic analytics platform that deeply integrates an LLM-based recommender to proactively assist a wide range of users in composing complex pipelines from a dynamic set of modular components.

Based on these problems, there is a clear need for a platform that integrates a low-code interface, a knowledge base for modular microservices, and an intelligent recommender to enhance collaborative analytics.

3.2 The Interconnected Nature of Gaps and the Need for an Integrated Solution

The three research problems identified above are not independent challenges but are rather deeply interconnected. Tackling them one by one misses the deeper issue causing inefficiency in collaborative analytics.

The central argument of this thesis is that a truly effective solution must address these problems holistically. This process creates a compounding effect, where each solution not only solves a problem but also enhances the capacity to solve other problems. This can drive a paradigm shift in how analytics are performed. The core connections are as follows:

1. **Collaboration Enables a Reusable Architecture:** Solving the *Collaboration and User Diversity Problem* is what populates and validates a *Modular and Extensible Architecture*. An effective role-based framework provides the human engine: Technical Experts contribute microservices, Domain Experts enrich them with semantic metadata, and End Users validate their utility through application. Without solving the collaboration problem, a modular architecture remains an empty, theoretical construct.

2. **Architecture and Knowledge Enable Reliable AI:** Solving the *Modularity and Reusability Problem* provides the essential foundation for reliable *AI-Guided Knowledge Discovery*. An LLM operating on a rich, structured Knowledge Base—filled with well-defined and curated components—can provide recommendations that are verifiable, trustworthy, and aligned with best practices. In contrast, unconstrained models that risk "hallucinating" unworkable solutions.

3. **AI Guidance Enables Collaboration at Scale:** Solving the *Intelligent Guidance Problem* closes the loop by making the platform accessible to a wider range of users. When novice users are empowered by effective AI guidance, the barrier to adoption plummets. This increased usage generates more analytical assets and provides valuable feedback, thus feeding the collaborative engine and further enriching the knowledge base. This makes the entire ecosystem more intelligent and effective over time.

This interconnectedness reveals that a new architectural approach is needed—one that is designed from the ground up to foster this virtuous cycle. The following sections will therefore introduce and propose **SLEGO**, a novel platform designed as a direct architectural response to this system of challenges, with the goal of fostering a more accessible, reusable, and truly collaborative data analytics ecosystem.

3.3 Design Science Research Methodology

The challenges outlined in Section 3.1 can be approached as **design questions**, necessitating the creation of a novel IT artefact—in this case, a new platform. Accordingly, we adopt the Design Science Research Methodology (DSRM) proposed by Peffers et al. [71], which aims to:

- Provide a structured, iterative process for building and evaluating innovative solutions in information systems.
- Ensure that the resultant artefact aligns with real-world problems, draws on relevant literature and technical foundations, and systematically measures its efficacy.

As illustrated in Figure 3.1, the main phases of DSRM are:

i. **Identify Problem and Motivation:** Clearly define the research problem and explain why an innovative solution is required.

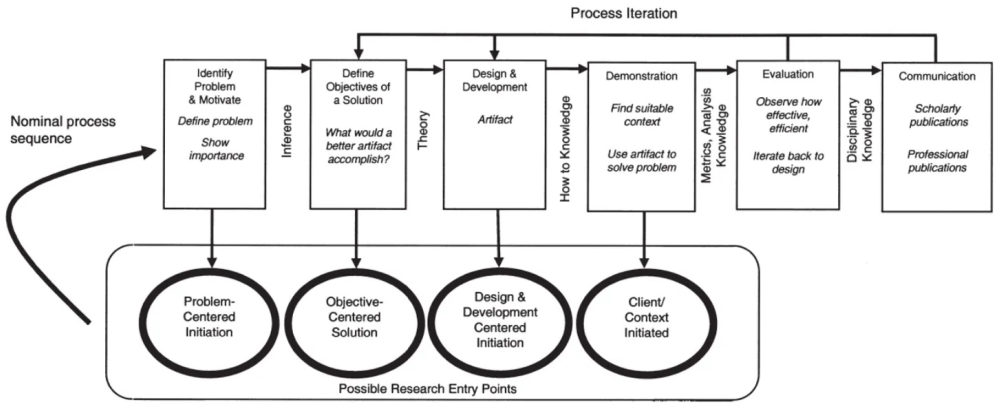

Figure 3.1: General DSRM Process Model (adapted from [71]).

ii. **Define Objectives for a Solution:** Translate the identified problem into measurable objectives and specify how a new artefact will meet these objectives.

iii. **Design & Development:** Propose the design of the artefact—conceptual models, system architecture, or prototypes—and develop it using appropriate software engineering methods.

iv. **Demonstration:** Show how the artefact solves the identified problem in one or more scenarios, often through *case studies* or *simulations*.

v. **Evaluation:** Compare the actual performance of the artefact with the objectives, using *quantitative* (e.g., platform performance) or *qualitative* (e.g., feedback or heuristics) metrics.

vi. **Communication:** Document and present the research contributions, methodology, and artifact performance to both scholarly and practitioner communities.

3.4 Research Approach Used in this Thesis

This section describes how each DSRM step is applied to address the research problems identified in Section 3.1, leading to the design, implementation, and validation of the

SLEGO platform.

3.4.1 Identify Design Objectives

Following the identification of the core research problems in Section 3.1, we define three primary design objectives to guide the development and evaluation of the SLEGO platform. These objectives directly address the identified gaps:

- **O1: End-to-End Collaboration Support:** To develop an analytics framework and a platform that support seamless, end-to-end collaborative workflows among both novice and expert users.
- **O2: Modular & Extensible Architecture:** To design a modular and microservices-based architecture, supported by a shared knowledge base, that facilitates the systematic reuse and extension of analytical components.
- **O3: AI-Guided Knowledge Discovery:** To integrate an LLM-based recommender system that automates the discovery and construction of analytical pipelines, making complex analytics more accessible.

These objectives systematically address the identified research problems and serve as the benchmarks for evaluating the effectiveness of the proposed solution.

3.4.2 Design and Development of SLEGO

In this thesis, the proposed solution to the identified research problems is a low-code collaborative analytics platform called SLEGO (Software-LEGO). SLEGO is designed as an integrated system comprised of the following key components:

I. **Low-Code User Interface:** An intuitive graphical environment that enables non-technical users of all skill levels to construct, configure, and execute analytics pipelines.

It features drop-down menus, parameter textboxes, and user-friendly dashboards to lower technical barriers and promote hands-on collaboration.

II. **Modular Microservices Architecture:** A flexible architecture where analytical workflows are composed from discrete microservices. Each microservice handles a single, well-defined task (e.g., data ingestion, feature engineering), and pipelines are defined through a formal configuration that specifies each step and its data dependencies. This structured approach ensures components are reusable and easy to integrate.

III. **Unified Cloud Storage for Data and Artifacts:** A centralized repository for all data and analytical artifacts, including microservice code, pipeline results, and models. This ensures consistent, shared access for all users across the platform.

IV. **Shared Knowledge Base:** A central metadata repository that captures domain-specific insights, technical specifications, and usage scenarios for all pipelines and microservices. It serves as the "brain" of the platform, transforming tacit knowledge into a reusable and governable asset.

V. **LLM-Based Recommender:** A retrieval-augmented large language model component that leverages the Knowledge Base to interpret user queries. It suggests relevant, pre-vetted pipelines and microservices, thereby automating the discovery and initial configuration of workflows.

These components are integrated into a holistic system that supports the entire collaborative analytics lifecycle. The artifact was developed using standard software engineering practices, and the resulting software has been open-sourced on GitHub.

3.4.3 Evaluation

The effectiveness of SLEGO was validated through a series of **three experiments**, each designed to simulate a key phase in a collaborative analytics lifecycle. These experiments

involved different data analytics tasks and data modalities, allowing for a comprehensive assessment of the platform's ability to meet its core design objectives.

1. **Experiment 1:** This experiment assesses the foundational collaborative workflow by simulating the creation of an analytical asset from scratch. It provides the primary validation for **Objective O1** by demonstrating how the architecture can seamlessly orchestrate the distinct contributions of Technical Experts, Domain Experts, and End Users.

2. **Experiment 2:** This experiment focuses on the platform's adaptability as the Knowledge Base grows. It provides the primary validation for **Objective O2** by demonstrating how an existing pipeline can be reused for a new task and extended with new, technologically diverse microservices.

3. **Experiment 3:** This experiment evaluates the platform at a mature stage, testing its versatility and the efficiency of its AI-driven guidance. It provides the primary validation for **Objective O3** by using quantitative tests to measure the LLM-recommender's accuracy in identifying relevant pipelines from a complex, populated Knowledge Base.

Collectively, these experiments provide practical demonstrations of SLEGO's architectural principles, offering insights into its strengths and potential areas for development. The outcomes of these evaluations are discussed in detail in Chapter 6.

3.4.4 Communication of Results

The final step of DSRM involves communicating outcomes and reflecting on the broader impact of SLEGO. Several channels facilitate this dissemination:

- **Academic Publications:** Conference and journal submissions highlighting system architecture, AI-driven components, and results from simulation-based evaluations.

- **Technical Documentation:** Developer and user guides describing microservice, knowledge base schema, and system configurations.
- **Open Source Release**: Making the source code publicly available via a GitHub repository to foster community engagement, enable reproducibility, and encourage further development by others.
- **Workshops and Demonstrations:** Presenting SLEGO at academic and industry venues to encourage adoption and collect critical technical feedback.

3.5 Conclusion

In this chapter, we presented the **Design Science Research Methodology (DSRM)** as the guiding strategy for the design, implementation, and evaluation of SLEGO. We identified the core issues motivating the need for a collaborative analytics platform and described how SLEGO's design objectives, architecture, and demonstration scenarios address these problems. In the chapters that follow, we elaborate on the DSRM phases in more detail.

Chapter 4

Design and Architecture of the SLEGO System

Software-LEGO (SLEGO) is a platform that streamlines the creation, sharing, and execution of analytics workflows. Its design is grounded in microservice modularity, role-based collaboration, and AI-driven guidance. This chapter details SLEGO's core *principles*, *architecture*, *user roles*, and *operational workflows*. It also provides a rigorous definition to formally describe pipeline composition and execution, illustrating these concepts with examples of analytics pipelines from the preceding sections.

4.1 Basic Principles as Design Objectives

SLEGO tackles the difficulties faced by diverse teams working together on intricate analytics projects. Conventional solutions frequently struggle to satisfy both the demand for extensibility sought by skilled developers and the need for user-friendliness desired by inexperienced analysts. To address these challenges effectively, SLEGO's design and creation are guided by a set of fundamental design objectives (which are also detailed with their corresponding platform components in Section 4.7, Table 4.3). These guiding

objectives are:

- **End-to-End Collaboration Support:** This objective is met by designing SLEGO to support the entire collaborative analytics life cycle through a role-oriented approach. It acknowledges that developing analytics pipelines is a team effort by recognising and catering to distinct user roles:
  - *Technical Experts (e.g., data scientists, software engineers):* Implement and validate microservices, ensuring rigorous documentation and correct I/O formats.
  - *Domain Experts (e.g., finance, healthcare professionals):* Contribute domain-specific annotations and example pipelines to the knowledge base, guiding novices in adopting best practices.
  - *End Users (e.g., business analysts, researchers):* Build and execute analytic workflows through the low-code interface or the *LLM-based recommender*, focusing on business goals rather than low-level coding details.

  This role-oriented design facilitates effective, traceable hand-offs and contributions.
- **Modular & Extensible Architecture:** SLEGO achieves this by decomposing analytics tasks into discrete *microservices* with standardised inputs and outputs. Each microservice targets a single transformation step (e.g., data ingestion, data cleaning, model training), making it straightforward to reuse or replace components across different domains. This design ensures a highly reusable and extensible system.
- **AI-Guided Knowledge Discovery:** SLEGO integrates AI, particularly through its *LLM-based recommender*, to streamline workflows. This system provides intelligent guidance, rapidly surfacing best-fit pipelines and components, thereby slashing duplicate effort and reducing the time-to-insight for all users.

By harmonising these core design objectives, SLEGO lowers the barrier to advanced analytics, allowing users from various backgrounds to collaborate effectively and scale up shared insights.

4.2 Analytics Pipeline Theoretical Foundations, Representation, and Execution

At the core of SLEGO lies the *Analytics Pipeline Execution Engine*, which manages data flow through a series of *microservices*. This section establishes the theoretical underpinnings of how these microservices interact in analytics pipelines by emphasizing *graph-theoretic representation*, *execution ordering constraints*, and *parallelization opportunities*.

SLEGO's organizing principle for analytics pipelines is the **Directed Acyclic Graph (DAG)**—a mathematical structure consisting of nodes (each representing a microservice) connected by directed edges (indicating data dependencies) with no cycles. This DAG representation serves multiple crucial purposes:

1. **Explicit Dependency Modeling:** The edges capture how intermediate outputs feed into subsequent microservices, creating a clear roadmap of the data transformation process.

2. **Guaranteed Execution Ordering:** The acyclic property ensures a valid sequence of processing steps always exists. Consequently, the system can systematically determine the order in which microservices must be executed.

3. **Parallel Execution Opportunities:** Independent or disconnected subgraphs in the DAG can be executed simultaneously. By identifying microservices that do not depend on each other, SLEGO maximizes resource utilization and reduces overall runtime.

4. **Traceability and Debugging:** When errors occur, the DAG structure allows precise localization of the issue. Because data dependencies are explicitly defined, the system can pinpoint the microservice or data flow that caused the failure.

By modelling analytics pipelines as DAGs, SLEGO grounds complex data transformations in a *rigorous yet intuitive* formalism. The following sections detail how this graph-based

paradigm is operationalized, including the scheduling algorithms used to organize microservice execution and the mechanisms for integrating new microservices into the pipeline.

4.2.1 Directed Acyclic Graphs (DAGs) as a Foundation

A *Directed Acyclic Graph (DAG)* is a type of directed graph characterized by vertices (nodes) connected by directed edges (arcs), with the defining constraint that it contains no directed cycles.

Formally, let $G = (V, E)$ represent a directed graph, where:

- $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of **vertices**. In the context of computational workflows, each vertex $v \in V$ typically corresponds to a distinct task, operation, or processing stage.
- $E \subseteq V \times V$ is a set of **directed edges**. An edge $e = (u, v)$, where $u, v \in V$, represents a directed connection from vertex u (the *tail*) to vertex v (the *head*). This directionality is crucial as it typically signifies a prerequisite relationship or a flow dependency; for instance, task v cannot commence until task u has completed, or data produced by u is required by v.

A **path** in G from vertex u to vertex v is a sequence of vertices $\langle v_0, v_1, \ldots, v_k \rangle$ such that $v_0 = u$, $v_k = v$, and for all $0 \leq i < k$, the edge $(v_i, v_{i+1}) \in E$. The **length** of such a path is k, the number of edges it comprises.

The graph G is **acyclic** if and only if there exists no vertex $v \in V$ such that there is a path of length $k \geq 1$ starting and ending at v. In simpler terms, following the direction of the edges, one can never return to a previously visited vertex. This acyclicity property is fundamental for modeling execution workflows because it guarantees the absence of circular dependencies, which would otherwise lead to logical impossibilities or deadlocks in execution.

A key consequence of acyclicity is the existence of a **topological sort** (or topological ordering) of the vertices. A topological sort is a linear ordering of all vertices in V such that for every directed edge $(u, v) \in E$, vertex u comes before vertex v in the ordering. This ordering provides at least one valid sequence for executing the tasks represented by the vertices while respecting all dependencies defined by the edges. The ability to determine such an order makes DAGs exceptionally well-suited for scheduling tasks, resolving dependencies, and orchestrating complex processes in areas ranging from computational pipelines and build systems to project management.

4.2.2 Graph-Theoretic Model of Microservices and Pipelines in SLEGO

In SLEGO, each data analytics pipeline is formally represented as a *Directed Acyclic Graph (DAG)* $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the set of vertices representing individual microservices, and $E \subseteq V \times V$ comprises directed edges representing data dependencies. Each edge $(u, v) \in E$ indicates that microservice v requires output produced by microservice u as input—establishing a precedence relationship. The acyclic property guarantees a well-defined execution order without circular dependencies, enabling reliable scheduling and execution.

Data States and Diverse Artifacts. Each edge (u, v) in the DAG corresponds to a data dependency where microservice v consumes artifacts produced by microservice u. These data artifacts, denoted as $D_i \in \mathcal{D}$ (where $\mathcal{D}$ is the space of all possible data objects), represent intermediate states in the analytics workflow. The system handles a rich variety of data types, including but not limited to:

- Tabular data (CSV, Parquet files)
- Semi-structured data (JSON, XML)
- Unstructured data (text documents, images, audio)
- Machine learning models (serialized .pkl, .h5 files)

- Visualization artifacts (HTML, static images, interactive plots)

Formally, a microservice $f \in V$ operates as a transformation function $f : \{D_{\text{in}}\} \to \{D_{\text{out}}\}$, consuming input artifacts to produce output artifacts. Each microservice maintains a strict input-output contract defined independently of any specific pipeline, ensuring clear specifications of requirements and deliverables across different execution contexts.

Pipeline Structures: Chaining, Branching, Merging, and Multiple Outputs. The DAG structure naturally models various workflow patterns observed in analytics. The following examples illustrate common patterns, though in practice, pipelines often involve complex combinations and variations of these fundamental structures:

Chaining: A linear sequence $f_1 \to f_2 \to \cdots \to f_{n+1}$ represents sequential processing, where each $f_i \in V$ is a microservice node in the DAG. The directed edges between these nodes represent data dependencies, with each edge $(f_i, f_{i+1}) \in E$ indicating that f_{i+1} requires the output of f_i. Formally, data flows through these nodes as follows:

$$\text{Source} \xrightarrow{D_0} f_1 \xrightarrow{D_1} f_2 \xrightarrow{D_2} \cdots \xrightarrow{D_n} f_{n+1} \xrightarrow{D_{n+1}} \text{Sink}, \tag{4.1}$$

where Source and Sink denote the initial and final nodes of the pipeline, respectively. Each microservice f_i transforms the artifact D_{i-1} into D_i, establishing a chain of dependencies in which every node can only be executed after its predecessor has completed. Maintaining these node–edge–node relationships ensures pipeline reliability.

Branching: Sometimes, a single microservice f_i produces an artifact D_i consumed by multiple subsequent microservices $f_{i+1}, f_{i+2}, \ldots$, creating parallel paths within the DAG. Formally, this can be represented as:

$$\text{Source} \xrightarrow{D_0} \cdots \xrightarrow{D_{i-1}} f_i \xrightarrow{D_i} \begin{cases} f_{i+1} \xrightarrow{D_{i+1}} \cdots \to \text{Sink} \\ f_{i+2} \xrightarrow{D_{i+2}} \cdots \to \text{Sink} \end{cases} \tag{4.2}$$

Once f_i completes, the artifact D_i becomes available to all dependent microservices, which can then run *in parallel*. Each downstream microservice ultimately produces its own result D, leading to faster overall execution when branching is feasible.

Merging: Conversely, a microservice f_m may require inputs produced by multiple upstream nodes f_j and f_k. If f_j emits $D_{j\to m}$ and f_k emits $D_{k\to m}$, then f_m merges these parallel outputs:

$$\begin{cases} \text{Source} \to \cdots \to f_j \xrightarrow{D_{j\to m}} \\ \text{Source} \to \cdots \to f_k \xrightarrow{D_{k\to m}} \end{cases} \quad f_m \xrightarrow{D_m} \cdots \to \text{Sink} \tag{4.3}$$

By explicitly modelling this convergence in the DAG, we ensure data integrity and correct execution order, even when multiple parallel branches flow into a single node.

Multiple Outputs from One Microservice: Finally, a single microservice f_i can emit *multiple* output artifacts. For example, f_i could consume $\{D_{i-1}^1, \ldots, D_{i-1}^k\}$ and produce $\{D_i^1, D_i^2, \ldots, D_i^r\}$. In a purely linear chain, you could illustrate multiple outputs as follows:

$$\text{Source} \xrightarrow{D_0} f_1 \xrightarrow{(D_1^A, D_1^B)} f_2 \xrightarrow{D_2} \cdots \xrightarrow{D_n} f_{n+1} \xrightarrow{D_{n+1}} \text{Sink}. \tag{4.4}$$

In this notation, f_1 produces two distinct artifacts (D_1^A, D_1^B), each of which may be consumed by f_2. This ability to handle diverse and multiple outputs (e.g., different data files, model artifacts, or reports) enables the pipeline to capture the rich array of possible transformations a single node might perform.

These patterns, as demonstrated across pipelines , represent the most common structures encountered in analytics pipelines, and real-world applications often feature sophisticated combinations and nested arrangements of these basic forms. SLEGO's DAG-based representation is sufficiently expressive to model arbitrary compositions of these patterns, enabling the construction of complex workflows while preserving the benefits of explicit dependency tracking and parallel execution opportunities.

Microservice Parameters and Configuration. Beyond the core data dependencies captured by the DAG edges, the behavior of individual microservices $f \in V$ is often tunable through a set of configuration parameters. These parameters might include hyperparameters for algorithms (e.g., learning rates, window sizes), thresholds, operational

modes, or references to external resources not explicitly modelled as data artifacts in the primary DAG flow. Formally, let Π_f denote the set of admissible parameter names for a microservice f. A specific execution instance of f operates under a configuration π, which is typically a mapping assigning values to a subset of parameters in Π_f. We can represent the configured microservice execution mathematically by explicitly including the parameter configuration π alongside the data inputs:

$$f : \{D_{\text{in}}\} \mid \pi \rightarrow \{D_{\text{out}}\} \tag{4.5}$$

Here, $\{D_{\text{in}}\} \subseteq \mathcal{D}$ is the set of input data artifacts, $\{D_{\text{out}}\} \subseteq \mathcal{D}$ is the set of produced output artifacts, and π represents the specific parameter configuration used for this execution instance (e.g., $\pi = \{\texttt{'window\_size'} : 20, \texttt{'learning\_rate'} : 0.01\}$). The pipe symbol ($|$) explicitly distinguishes the data inputs from the configuration parameters. **Crucially, while the configuration π alters the *content* or *values* within the output artifacts $\{D_{\text{out}}\}$, the fundamental *structure* or *schema* of these outputs is expected to remain consistent with the microservice's defined contract**. This structural consistency is vital for ensuring reliable data flow to downstream microservices (i.e., consumers of $\{D_{\text{out}}\}$), enabling robust validation, and maintaining overall pipeline predictability and component reusability.

Modularity, Reuse, and Evolution. Viewing each microservice f as an independent node $v \in V$ inherently supports modularity. A microservice is defined by its function signature and input/output contract, independent of the specific pipeline (graph G) in which it participates. This allows the same microservice node definition to be modular and reusable across multiple different pipelines, potentially with different parameterisations, significantly reducing development effort. Furthermore, the graph model provides a formal basis for pipeline evolution: modifications such as adding a step (adding a node and related edges), removing a step (deleting a node and edges), or changing a dependency (modifying edges) correspond directly to standard graph update operations.

Parallel Computation. Two nodes can run concurrently whenever no directed path exists between them in the DAG—that is, neither u is an ancestor of v nor v of u. This situation arises in two ways:

1. *Disjoint sub-graphs:* u and v belong to separate branches of the pipeline and share no common parent; once their own dependencies are met, they execute independently.
2. *Branch fan-out:* one parent f produces an artifact consumed by multiple children $\{f_1, \ldots, f_k\}$. After f finishes, the in-degree of every child drops to 0, so the scheduler can launch $\{f_1, \ldots, f_k\}$ in parallel.

SLEGO's graph-based approach, leveraging DAGs, provides a formal and flexible foundation for composing, executing, and evolving complex analytics pipelines involving diverse data artifacts and workflow patterns. By ensuring reliability, modularity, and opportunities for performance optimization through parallelism, it becomes the cornerstone of an efficient and maintainable analytics ecosystem.

4.2.3 Theoretical Design Guidelines for Microservices and Pipelines

Building on the graph-theoretic model in Section 4.2.2, the following **six guidelines** ensure that every SLEGO pipeline is *modular*, *reproducible*, and *evolvable*.

G1. Single-Responsibility Microservices (Modularity): Each vertex $v \in V$ implements *exactly one* transformation $f : \{D_{\text{in}}\} \to \{D_{\text{out}}\}$. Fine-grained tasks are easier to test, reuse, and swap out independently.

G2. Explicit Data Contracts (Integrity): Every edge $(u, v) \in E$ conveys an artefact whose schema and semantics are part of the contract between the producing service u and the consuming service v. Contract validation prevents silent data corruption.

G3. Pure Parameterisation (Determinism & Flexibility): A configured service, driven only by $\{D_{\text{in}}\}$ and parameters π, behaves as a pure function. Hidden global state or non-deterministic calls should be avoided or surfaced as explicit parameters.

G4. Stateless & Retry-Safe Execution (Reliability): A microservice must retain no persistent state between invocations; all information required for its computation is supplied through its inputs $\{D_{\text{in}}\}$ and parameter set π. It must not modify global variables, write to shared scratch space, or depend on previously cached data.*Retry-safety.* The service must never mutate its inputs. When re-executed with identical parameters it must either (i) deterministically overwrite the same declared output paths, or (ii) write to a uniquely derived path (e.g., timestamp or hash) and return that path via the pipeline's artefact map. These rules ensure safe retries, fault tolerance, and horizontal scaling.

G5. Acyclic Data-Flow (Scheduling Soundness): The complete pipeline must form a DAG $G = (V, E)$. Acyclicity guarantees at least one valid topological order, eliminating dead-locks and enabling parallel scheduling.

G6. Clear Documentation (Transparency & Discoverability): Every microservice ships with human-readable metadata covering its purpose, parameters, expected inputs, generated outputs, and failure modes. Rich documentation is essential for discoverability and reusability.

Adhering to guidelines G1–G6 keeps the SLEGO ecosystem robust and extensible, while giving pipeline schedulers the guarantees they need for safe, efficient execution.

4.2.4 Example SLEGO Pipeline: Formal Model

This section introduces the formal graph-theoretic model for a sample pipeline. A pipeline in SLEGO is represented as a configured Directed Acyclic Graph (DAG), which can be expressed with a data-flow equation.

In this abstract model, each vertex is a configured microservice instance, denoted as a pair (f_i, π_i), where f_i is the underlying microservice function and π_i is its specific parameter configuration. Each edge is a data artefact D that connects vertices. The concrete declarative representation of this model, which defines the specific function names and parameters, is detailed in the subsection 4.2.5.

$$\text{Source} \xrightarrow{D_{\text{initial}}} \begin{cases} f_1|\pi_1 \xrightarrow{D_A} f_3|\pi_3 \xrightarrow{D_C} \\ f_2|\pi_2 \xrightarrow{D_B} \end{cases} \quad f_4|\pi_4 \xrightarrow{D_{\text{final}}} \text{Sink} \tag{4.6}$$

Mathematically, this corresponds to the formal graph $G = (V, E)$ where:

$$\boxed{G = \Big(\{v_1, v_2, v_3, v_4\}, \{(v_1, v_3), (v_2, v_4), (v_3, v_4)\}\Big).}$$

The sets V and E are constructed as follows:

- **Vertex set V.** Each configured instance (f_i, π_i) becomes a vertex v_i. For this example, the set is $V = \{v_1, v_2, v_3, v_4\}$.
- **Edge set E.** Edges are inferred from data-flow dependencies. The output of v_1 is an input to v_3, creating the edge (v_1, v_3). Similarly, the data flow creates edges (v_2, v_4) and (v_3, v_4).

4.2.5 Declarative Representation of the Example Pipeline

To implement the formal graph model from subsection 4.2.4, SLEGO captures the entire analytics pipeline in a single, declarative *JSON format*. This format serves as the canonical, executable specification from which the formal graph is derived, as shown in Listing 4.1.

```
{
  "microservice_1_transform_A": {
    "inputs":  { "initial_data": "dataspace/initial_input.csv" },
    "outputs": { "data_out": "dataspace/intermediate_A.csv" },
```

```
    "parameter_1": 0.5
  },
  "microservice_2_transform_B": {
    "inputs":  { "initial_data": "dataspace/initial_input.csv" },
    "outputs": { "data_out": "dataspace/intermediate_B.csv" },
    "parameter_2": "mode_x"
  },
  "microservice_3_aggregate_A": {
    "inputs":  { "input_stream": "dataspace/intermediate_A.csv" },
    "outputs": { "aggregated_out": "dataspace/intermediate_C.csv" },
    "parameter_3": 200
  },
  "microservice_4_merge_all": {
    "inputs":  {
      "stream_B": "dataspace/intermediate_B.csv",
      "stream_C": "dataspace/intermediate_C.csv"
    },
    "outputs": { "final_report": "dataspace/final_output.json" }
  }
}
```

Listing 4.1: A pipeline specification where parameters are defined as top-level keys.

This JSON structure provides the concrete details for the abstract model:

- **Vertices ($v_i = (f_i, \pi_i)$):** The system maps the JSON to the formal model.
  - The microservice function f_i is identified by the top-level string key (e.g., `"microservice_1_transform_A"`).
  - The parameter configuration π_i consists of all other key-value pairs within that microservice's object, excluding `"inputs"` and `"outputs"`. For the first vertex v_1, this means its configuration is $\pi_1 = \{\texttt{"parameter\_1"} : 0.5\}$.
- **Artefacts (D) and Edges (E):** The file paths in the `"inputs"` and `"outputs"` dictionaries represent the data artefacts. The system induces a directed edge by

matching an output path from one microservice to an input path of another. For instance, the artefact `".../intermediate_A.csv"` links `microservice_1_transform_A` to `microservice_3_aggregate_A`, creating the edge (v_1, v_3) in the graph G.

This compact JSON representation provides a complete, self-contained definition of the workflow. The system parses this file to automatically construct the formal graph G and derive the data-flow dependencies shown in Equation (4.6). This structure explicitly reveals the workflow's dependencies, including the parallel start of the first two microservices and the final merge. The responsibility for interpreting this structure and determining a valid execution order lies with a topological sorting algorithm, as detailed in the next section.

4.2.6 Kahn's Algorithm for Topological Sorting

To produce a valid execution sequence, SLEGO uses **Kahn's algorithm** [37] for topological sorting. The algorithm (see Algorithm 1)identifies all zero in-degree nodes (i.e., microservices that depend on no others), processes them, and repeatedly updates the remaining graph. If all vertices are eventually processed, the resulting list is a valid topological order. Kahn's algorithm runs in $O(|V| + |E|)$ time, suitable for large graphs in real-world data pipelines.

Algorithm 1 Kahn's Algorithm for Topological Sorting (Part 1)

Require: A DAG $G = (V, E)$ where each vertex $v \in V$ represents a microservice, and each directed edge $(u \to v) \in E$ denotes a dependency.

Ensure: *ExecutionOrder*, a valid topological ordering of the microservices, or failure state.

1: // Phase 1: Initialize In-Degree Counts
2: **for** each vertex $v \in V$ **do**
3: $inDegree[v] \leftarrow 0$ ▷ Initialize incoming dependency count for all nodes to zero
4: **end for**
5: // Phase 2: Calculate Actual In-Degrees from Edges
6: **for** each edge $(u \to v) \in E$ **do** ▷ Iterate through all defined dependencies
7: $inDegree[v] \leftarrow inDegree[v] + 1$ ▷ Increment the count for the node the edge points to
8: **end for**
9: // Phase 3: Initialize Queue with Source Nodes
10: Let Q be an empty queue (or list) ▷ Queue to hold nodes with zero in-degree (ready to process)
11: **for** each vertex $v \in V$ **do**
12: **if** $inDegree[v] = 0$ **then** ▷ If a node has no incoming dependencies...
13: Enqueue v into Q ▷ ...add it to the queue as it can be scheduled first
14: **end if**
15: **end for**

Algorithm 1 Kahn's Algorithm for Topological Sorting (Part 2)

// Phase 4: Process Nodes and Build Execution Order
Initialize an empty list *ExecutionOrder* ▷ This will store the final sorted sequence
while Q is not empty **do** ▷ While there are nodes ready to be processed...
    $v \leftarrow$ Dequeue a vertex from Q ▷ Get the next node with no unmet dependencies
    Append v to *ExecutionOrder* ▷ Add the processed node to the result list
    ▷ For all nodes that depend on the processed node 'v'...
    for each edge $(v \rightarrow w) \in E$ **do** ▷ Iterate through 'v''s outgoing edges
        $inDegree[w] \leftarrow inDegree[w] - 1$ ▷ Decrement neighbor's in-degree (dependency from 'v' is met)
        if $inDegree[w] = 0$ **then** ▷ If 'w' now has no other pending dependencies...
            Enqueue w into Q ▷ ...it becomes ready to be processed, add to queue
        end if
    end for
end while
// Phase 5: Validation and Return
if length(*ExecutionOrder*) $\neq$ number of vertices in V **then** ▷ Check if all nodes were processed
    fail("Graph contains a cycle") ▷ Cycle detected
else
    return(*ExecutionOrder*) ▷ Return the valid sequence
end if

4.2.7 Validation and Data Integrity Checks

SLEGO enforces a two-phase validation procedure to ensure both data integrity and parameter consistency before running any microservices.

Phase 1: DAG Assembly and Topological Sort.

1. **Pipeline Specification Parsing:** The pipeline specification, which is provided in a defined format (e.g., JSON, YAML, or another structured representation), is read and interpreted into an internal data structure. This structure captures the details for each microservice, including its defined `inputs`, `outputs`, and configuration `parameters`.

2. **DAG Construction:** A directed graph $G = (V, E)$ is formed by:

   - Adding one node $v \in V$ for each microservice defined in the specification.
   - Creating directed edges $(u \rightarrow v) \in E$ whenever the outputs declared for microservice u match any inputs declared for microservice v, establishing data dependencies.

3. **Topological Sort and Cycle Check:** A topological sort algorithm (Kahn's algorithm, see Section 4.2.6) is applied to the constructed DAG G. This process serves two critical functions:

   - It checks for cycles (circular dependencies) within the graph. If a cycle is detected, the pipeline definition is invalid, and SLEGO halts execution immediately, reporting the circular dependency.
   - If the graph is acyclic, it computes a valid linear ordering of the microservices. This order ensures that nodes with no incoming edges (i.e., no prerequisites) appear first, guaranteeing that each microservice is processed only after all its dependencies have successfully completed.

Phase 2: Data and Parameter Validation. Once SLEGO has determined a valid topological order (let's denote it *execOrder*), it proceeds to validate each microservice's data requirements and parameters sequentially according to this order:

1. **Validation of Inputs for Initial Nodes:** For any microservice f identified as an initial node in the topological sort (i.e., having zero in-degree), all its required input

resources (e.g., data files, database tables, cloud storage objects) must exist and be accessible *before* execution begins. SLEGO checks:

- Are the specified input resources present at the given locations (paths, URIs, etc.)?
- Does the data appear to be in the expected format or type, if verifiable beforehand (e.g., checking file extensions or basic structure)?

If any required initial input is missing, inaccessible, or seems incorrectly formatted, SLEGO reports the specific error and aborts the pipeline execution.

2. **Validation of Inputs for Subsequent Nodes:** For any microservice that depends on outputs from one or more upstream microservices (as defined by the DAG edges), SLEGO confirms:

   - The correct mapping of specific upstream outputs to the corresponding inputs.
   - That each required input artifact is expected to be generated by its designated predecessor microservice within the pipeline flow.

   If there's a mismatch in the expected data flow (e.g., an input references an output that is never produced according to the DAG), SLEGO raises a structural error and terminates.

3. **Parameter Consistency:** Concurrently with input validation, the parameters specified for each microservice are verified against its defined signature or contract. These checks typically include:

   - **Type/Format Check**: Ensuring that provided parameter values match the expected data types (e.g., integer, float, string, boolean, list).
   - **Required vs. Optional**: Verifying that all mandatory parameters have been provided in the specification. Optional parameters might be checked for validity if provided, or default values might be assigned if applicable.

   Any parameter mismatch, missing required parameter, or invalid value triggers an immediate failure with a descriptive error message.

Executing the topological sorting *prior* to the detailed data and parameter validation guarantees that the dependency structure is robust and acyclic. Validating resources and parameters in topological order ensures that each microservice is checked only when its position in the workflow is confirmed. This approach eliminates redundant checks and establishes a clear, sequential validation flow. If all validation steps pass successfully, SLEGO proceeds to execute the pipeline; otherwise, it halts and provides comprehensive error feedback to facilitate rapid debugging and correction of the pipeline specification or environment setup.

4.2.8 Overall Execution Lifecycle

SLEGO executes analytics pipelines through a structured lifecycle designed for correctness and efficiency. The core stages are:

1. **Parse Pipeline Definition:** Read and interpret the user-provided pipeline specification (e.g., from a configuration file). This defines the microservices (tasks), their required inputs, expected outputs, and configuration parameters.

2. **Build and Analyze Dependency Graph:** Construct a Directed Acyclic Graph (DAG) where nodes represent microservices and edges represent data dependencies between them. Check this graph for cycles (circular dependencies); if found, report an error and halt. If acyclic, determine a valid execution sequence (topological order).

3. **Pre-flight Validation:** Before starting any execution, perform critical checks: verify that necessary initial input data/resources are available and accessible, and validate that the parameters specified for each microservice are consistent with its requirements (e.g., correct types, valid values). Halt with an error if any validation fails.

4. **Execute Tasks:** Run the microservices according to the valid execution order derived from the DAG. Microservices whose dependencies are met can be executed,

potentially in parallel to improve performance. The system manages the flow of data outputs from completed tasks to the inputs of subsequent tasks. Halt if any microservice encounters an unrecoverable runtime error.

5. **Finalize Execution:** Once all tasks complete successfully, the pipeline finishes. Final output artifacts are made available. Record execution status, logs, and potentially performance metrics for monitoring and auditing.

This lifecycle ensures dependencies are respected, configurations are validated upfront, and parallel execution is leveraged where possible. Algorithm 2 provides a high-level pseudocode representation of this process.

Algorithm 2 SLEGO Pipeline Execution (Part 1)

Input: A pipeline specification describing microservices, inputs/outputs, and parameters.
Output: Final artifacts produced by the microservices, plus logs/status of execution.
// Phase 1: Parse Specification and build DAG
pipelineSpec ← PARSESPECIFICATION(*pipeline_file*) ▷ read and parse pipeline specification file
G ← CONSTRUCTDAG(*pipelineSpec*) ▷ build DAG of microservices
// Phase 2: Topological sorting (cycle detection)
if CYCLEDETECTED(G) **then** ▷ detect cycles in the DAG
    fail("Cycle detected in pipeline configuration.")
end if
execOrder ← TOPOLOGICALSORT(G) ▷ obtain valid topological order
// Phase 3: Validate resources and parameters for ALL microservices (Pre-flight)
for m **in** *execOrder* **do** ▷ iterate through microservices in topological order
    if not VALIDATEINPUTS(m, *pipelineSpec*) **then** ▷ ensure inputs exist/are correct type/format
        fail("Missing or invalid inputs for microservice: " + m)
    end if
    if not VALIDATEPARAMETERS(m, *pipelineSpec*) **then** ▷ ensure parameters are type/value correct
        fail("Invalid parameters for microservice: " + m)
    end if
end for
▷ All microservices passed pre-execution validation

Algorithm 2 SLEGO Pipeline Execution (Part 2)

```
// Phase 4: Execute microservices (with parallelism if applicable)
for m in execOrder do                          ▷ execute tasks following topological order
    status ← EXECUTE(m, pipelineSpec)          ▷ Invoke microservice logic
    if status is FAILURE then
        fail("Execution failed for microservice: " + m)
    end if
    if not CHECKOUTPUTS(m, pipelineSpec) then  ▷ verify expected outputs exist after execution
        fail("Expected outputs missing or invalid after execution for microservice: " + m)
    end if
end for
return("Pipeline completed successfully; outputs generated.")
```

4.3 SLEGO Architecture

Having established *what* a SLEGO pipeline is (a validated DAG of microservices) and *how* it is scheduled and verified, we now turn to *where* these concepts live in the running system. SLEGO adopts a **three-layer architecture** consisting of a Storage Layer, a Service Layer, and a UI Layer. Figure 4.1 provides a conceptual overview of these layers and their interconnections.

4.3.1 Storage Layer

- **Cloud Storage:** A centralised repository for all organisational assets, including raw data, microservice code, intermediate outputs, and final results. This unified system ensures consistent access and version control for all user roles.
- **Knowledge Base:** A structured database that catalogues metadata about each

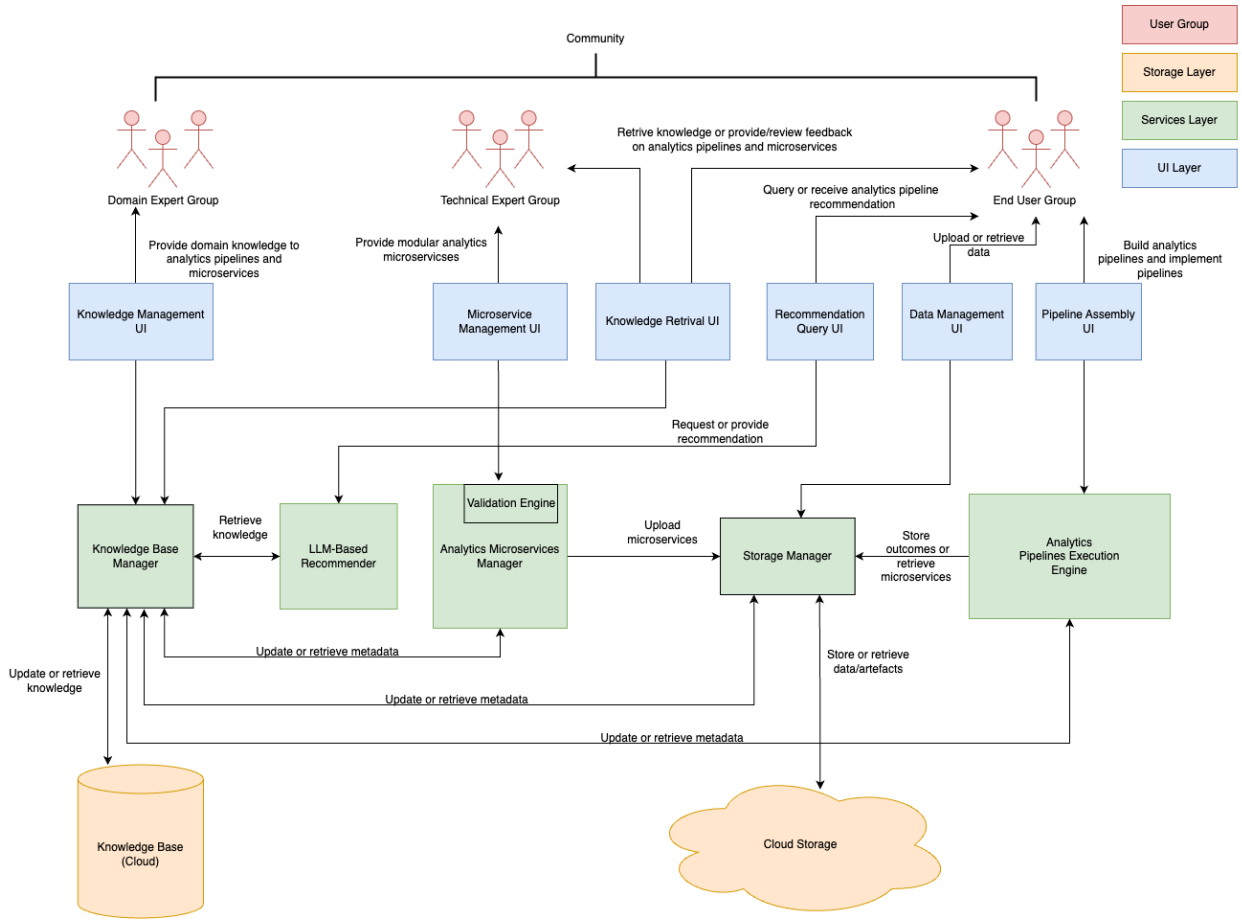

Figure 4.1: High-Level Overview of the SLEGO Architecture

microservice: descriptions, parameters, example pipelines, and vector embeddings for semantic retrieval. The Knowledge Base also stores user-constructed pipelines with domain-specific annotations. Domain experts and technical users continuously update this repository to keep the system's collective intelligence current.

4.3.2 Service Layer

The service layer manages core logic and interactions between storage and the user interface:

- **Analytics Microservices Manager & Validation Engine:** Oversees the lifecy-

cle of new or updated microservices, verifying that each meets SLEGO's guidelines for docstrings, function signatures, and I/O specifications. Compliant microservices are then admitted to the production environment.

- **Storage Manager & Knowledge Base Manager:** Provide standardized APIs for reading/writing from Cloud Storage and updating/searching in the kb. By abstracting these operations, SLEGO ensures consistent data manipulation across the platform.
- **Analytics Pipeline Definition and Execution Engine:** Define and executes analytical pipelines analytical pipelines by loading the configured microservices, passing data between them, and storing results in Cloud Storage.
- **LLM-Based Recommender:** Leverages the knowledge base to suggest relevant microservices or full pipelines based on a user's plain-language query (see Section 4.4). It can also propose parameter values aligned with domain-specific constraints.

4.3.3 UI Layer

The UI Layer acts as the main interface for users to engage with the SLEGO system, enabling seamless pipeline creation, execution, and knowledge management. It includes the following components:

- **Pipeline Assembly UI:** Users assemble analytics pipelines via an intuitive, drag-and-drop graphical interface. This UI operational SLEGO's **consistent low-code workflow** [44, 13], which, as depicted in Fig. 4.2, simplifies analytics into three primary steps:
  1. **Component Selection:** Users visually choose microservices from a library.
  2. **Parameter Input:** Microservices are customised via GUI-based parameter adjustments.

3. **Execution:** Pipelines are run with a single click.

This standardised interaction minimises manual coding. The process can be further streamlined by the LLM-based Recommender, which suggests pipeline configurations from natural language queries [99], allowing users to focus on insights rather than technical complexities [41].

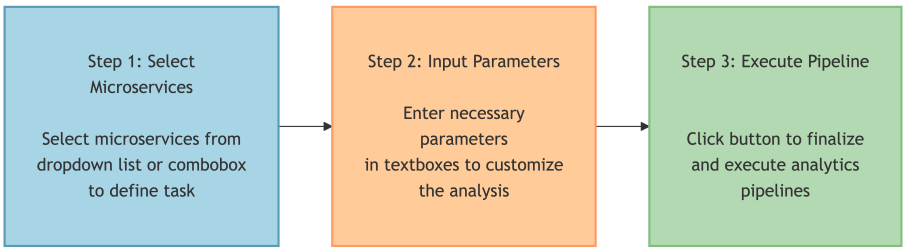

Figure 4.2: The three-step low-code workflow in the Pipeline Assembly UI: Component Selection, Parameter Input, and Execution.

- **Data Management UI:** Through the integrated data management interface, users can seamlessly upload, browse, and manage datasets and analytic outputs. This component provides structured storage, easy retrieval, and visualization of results (e.g., charts, CSV files, and generated models), simplifying data-driven decision-making.

- **Recommendation Query UI:** The recommender system enables users, particularly those lacking advanced technical skills, to express their analytical objectives using a a natural language. The LLM-driven recommender employs semantic search and incorporates domain-specific knowledge to automatically create candidate pipelines. These proposed pipelines can be reviewed, adjusted, and approved by users before being executed, thereby closing knowledge gaps and expediting the development of analytics applications.

- **Microservice Management UI:** Technical experts utilize this interface to submit new microservices, monitor validation results, and manage the lifecycle of existing analytics components. Clear feedback and validation status updates streamline microservice integration into the SLEGO environment.

- **Knowledge Management UI:** Domain experts and authorised technical users maintain the platform's knowledge base with full access directly via the UI. This includes editing microservice metadata, parameter descriptions, recommended default settings, and sample pipeline templates. These updates ensure the platform remains current with best practices and domain-specific guidelines.
- **Knowledge Retrieval UI:** A single discovery interface where *both* non-technical end-users and technical experts can search and browse artefacts in the Knowledge Base's metadata. The same UI supports providing comments and feedback for specific microservices or pipelines, enabling users to exchange ideas, flag issues, and coordinate improvements—conversations are stored with the relevant artefacts to guide future work.

Collectively, these UI elements simplify complex analytics workflows, enhancing user productivity and collaboration across diverse technical and domain expertise levels.

4.3.4 Knowledge-Base Schema

The Knowledge Base (KB) serves as the definitive catalog for all reusable *microservices* and saved *pipelines* within the SLEGO system. It is implemented using two primary SQL tables: `microservices`, detailed in Table 4.1, and `pipelines`, detailed in Table 4.2. The system automatically populates rows in the `microservices` table when Technical Experts upload validated programs.

Table 4.1: Schema of the `microservices` table. Each row represents a specific version of a microservice.

Column	Content	Type
`id`	Internal identifier for joins and lineage tracking.	INTEGER
`module_name`	Source file provided by a Technical Expert.	TEXT
`component_name`	Function or class name implementing the analytic step.	TEXT
`language_type`	Programming language of the microservice (e.g., Python, Java, R).	TEXT
`source_code`	Full language source code for static analysis and auditing.	TEXT
`docstring`	Markdown documentation (purpose, usage notes, examples).	TEXT
`parameters`	JSON schema defining expected parameters, types, and defaults.	JSON
`pipeline_names`	JSON list identifying pipelines that import this microservice.	JSON
`comment`	Feedback from all users	TEXT

Table 4.2: Schema of the `pipelines` table. Domain knowledge is captured in the `description` column via the template in Listing 4.3.

Column	Content	Type
`id`	Internal identifier for each stored pipeline.	INTEGER
`pipeline_name`	Human-readable title assigned via the GUI.	TEXT
`pipeline_description`	Knowledge-prompt record (JSON format) completed by Domain Experts.	TEXT (JSON)
`pipeline_details`	Canonical JSON Directed Acyclic Graph (DAG) defining the pipeline structure.	JSON
`embedding`	High dimensional vector representation of `pipeline_description` for semantic search.	VECTOR
`comment`	Feedback from all users	TEXT

Domain Experts enrich the KB by adding contextual knowledge to the `description` field of each pipeline using a structured template shown in Listing 4.2. This design maintains a loose coupling between `microservices` and `pipelines` semantically, enabling the recommendation system to reason effectively over both aspects.

```
1 {
```

```
  "task_goal": "<Comprehensive objective and business context>",
  "data_context": "<Dataset input/output, modality, size, format>",
  "problem_type": "<regression|visualization|classification|time-series|NLP
    |computer-vision|ETL>",
  "pipeline_process": "<High-level flow, e.g. clean->feature-engineer->
    train->explain>",
  "pipeline_components": "<Exact microservices and key parameters>",
  "domain_keywords": "<finance, healthcare, marketing, ...>",
  "additional_info": "<Constraints: latency, privacy, metrics, deployment
    notes>"
}
```

Listing 4.2: Seven-key JSON template embedded in `pipelines.description`.

This schema design offers several operational advantages:

1 **Complete Provenance** – Each pipeline record encapsulates its analytical intent, data requirements, and domain context within the `description` field, simplifying auditing and replication.

2 **Semantic Retrieval** – The vector embedding of the `description` field enables efficient nearest-neighbor search for LLM (see section 4.4) . This ensures recommended pipelines match the user's query in both topical relevance and technical structure.

3 **Integrated Governance** – Domain Experts can embed regulatory constraints or industry-specific requirements directly into the pipeline's `description`, preserving this information alongside the executable DAG and facilitating compliance reviews.

In summary, this schema effectively separates lean microservice metadata from rich domain expertise. It strategically places domain knowledge within the pipeline records, where it is most actionable for driving specific analytical workflows and informing the recommendation engine.

4.3.5 Facilitating Low-Code Analytics Operations

A cornerstone of the SLEGO system's design philosophy is the empowerment of diverse users, particularly those without extensive programming expertise, through a **low-code operational paradigm**. This approach is crucial for democratizing data analytics and bridging the gap between technical developers and domain-focused end-users [44, 78]. SLEGO's UI Layer (Section 4.3.3) is specifically engineered to facilitate the construction and execution of complex analytics pipelines with minimal to no direct coding required from the end-user. This aligns with the broader trend of Low-Code Platforms (LCPs) which aim to simplify development and increase productivity by integrating traditional tools into more accessible environments [13].

The primary mechanism for low-code operation in SLEGO is its **visual pipeline assembly interface**. Users can select pre-built, validated microservices from the Knowledge Base and connect them in a drag-and-drop or selection-driven manner to form an analytical workflow. This visual paradigm abstracts the underlying code complexity of each microservice, allowing users to focus on the analytical logic and data flow, similar to established visual workflow systems like KNIME [25, 81] or data analytics platforms like DataChat that support natural language and point-and-click interactions [35]. Each microservice, while potentially complex internally, presents a simplified interface to the end-users, primarily through configurable parameters.

Parameter-driven customisation is another key aspect of SLEGO's low-code approach. Instead of modifying code, users adapt microservices to their specific needs by adjusting input parameters via the GUI. This ensures that the core logic of the microservice (developed and validated by Technical Experts) remains intact, promoting robustness and reusability [43, 100, 69].

Furthermore, SLEGO's **LLM-based Recommender System** (described in Section 4.4) pushes the boundary towards a no-code experience for pipeline initiation. By allowing users to describe their analytical goals in natural language, the LLM translates these high-level instructions into concrete pipeline suggestions, drawing upon the structured

information in the Knowledge Base [99, 54, 67]. This significantly lowers the barrier to entry, enabling users who may not even know which microservices exist or how they should be combined to construct sophisticated analytics [77, 98]. The system aims to provide intelligent assistance, mitigating the "blank canvas" problem often faced by novice users in complex software environments.

This low-code operational model is designed to empower a broader range of "citizen data scientists" [62] and domain experts, allowing them to independently conduct analyses that would traditionally require dedicated programming support. By reducing the technical hurdles, SLEGO aims to foster a more inclusive and efficient analytical environment, where the focus shifts from coding mechanics to analytical problem-solving and insight generation [41]. The platform thus acts as an integrated environment that supports the "collaborative assembly and execution of analytical tasks," as envisioned by systems moving beyond traditional, siloed toolsets [88, 56].

4.4 LLM-Based Recommendation System

The SLEGO LLM-Based Recommendation system addresses the challenge of navigating extensive microservice libraries for users without specialised analytics expertise. It functions by translating natural language objectives into specific pipeline suggestions, thereby aiming to reduce the initial learning curve, expedite the experimentation process, and facilitate the adoption of established best practices throughout an organisation.

4.4.1 Motivation

Contemporary analytics platforms often comprise hundreds of modular operators (e.g., data cleaning modules, feature encoding components, and model training algorithms). The selection of an appropriate subset of these operators and their sequential arrangement can present a significant challenge for users unfamiliar with the intricacies of analytics workflows. Large Language Models (LLMs) possess capabilities in mapping natural language

intents to structured outputs [86, 15]. By integrating an LLM within the recommendation loop, SLEGO is designed to:

- Parse the user's stated analytical goal and relevant contextual information.
- Identify and retrieve pertinent pipelines and microservices from a Knowledge Base.
- Generate concise, human-readable explanations for each proposed suggestion.

Through these mechanisms, the recommendation system seeks to lower the barrier to entry for analytics tasks, accelerate the identification of suitable analytics pipelines, and promote the reuse of established best practices. The overarching goal is to enhance the accessibility and transparency of sophisticated analytics for a broader range of users.

4.4.2 Prompt Template for Users

To maximise retrieval accuracy, users are encouraged to structure their queries in JSON format, as illustrated in Listing 4.3. Optional fields such as `additional_info` can capture constraints (e.g., privacy or performance requirements) without cluttering the main goal statement.

```
{
  "task_goal": "<What should the pipeline achieve? e.g. 'predict
   house prices in Sydney'>",
  "data_context": "<Dataset input/output, name, modality, size &
   format>",
  "problem_type": "<regression | Visualization | classification |
   time-series | NLP | computer-vision | ETL>",
  "domain_keywords": "<finance, healthcare, marketing>",
  "additional_info": "<optional constraints, desired metrics,
   deployment notes>"
}
```

Listing 4.3: Recommended prompt schema for the LLM-Based Recommender.

4.4.3 LLM-Based Recommendation System Design

The LLM-Based Recommendation system is built upon a multi-layered architecture designed to provide reliable and context-aware pipeline suggestions by leveraging a Retrieval-Augmented Generation (RAG) [49] approach. Building on the high-level overview Figure 4.1, Figure 4.3 illustrates the high-level conceptual architecture, showing the interaction between the Recommendation Query UI, the core Recommender Controller, the Knowledge Base Manager, and the Knowledge Base itself.

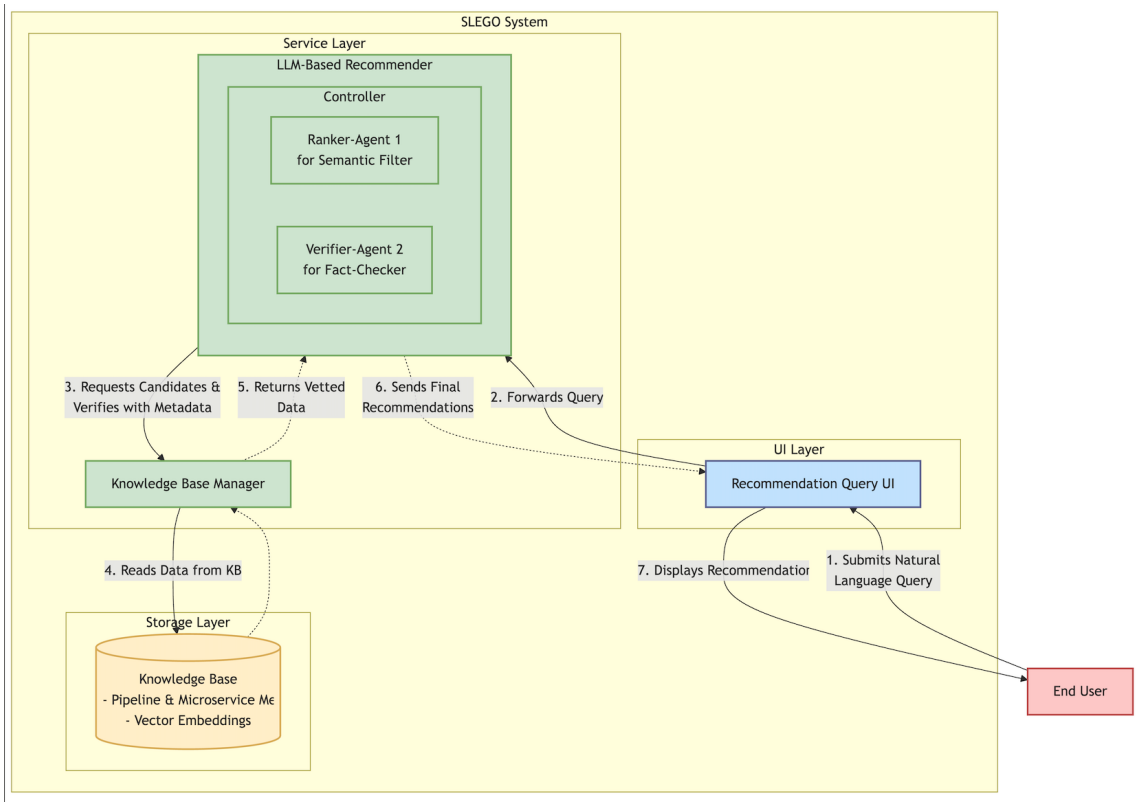

Figure 4.3: SLEGO LLM Recommender Architecture

Figure 4.4 illustrates this recommender's workflow. The process leverages a Knowledge Base (KB), defined as a tuple $KB = (P, M)$, where P is the set of all pipeline specifications and M is the set of all microservice definitions. The recommendation workflow proceeds through the following steps. This process is governed by key parameters, which, while configurable, are assigned specific values in our implementation to illustrate a concrete

operational flow:

Step 1: User Query and Embedding. The process commences with a user query, q, which is transformed into a high-dimensional vector embedding, v_q, by an embedding model, E. For this implementation, we utilise OpenAI's `text-embedding-3-small` model[1].

$$v_q = E(q) \tag{4.7}$$

Step 2: Candidate Retrieval. A set of candidate pipelines, C_{initial}, is retrieved by performing a nearest-neighbour search against the embeddings of all pipelines in P. This step retrieves the top k_{attempt} pipelines. For the initial iteration, our implementation sets the retrieval size $k_0 = 10$. This is formalised as:

$$C_{\text{initial}} = \underset{p_i \in P, k=k_{\text{attempt}}}{\arg\,\text{topk}} \left(\text{sim}(v_q, E(d_i))\right) \tag{4.8}$$

where p_i is an individual pipeline from the set P, d_i is its textual description, and `sim` is a similarity function, typically cosine similarity.

Step 3: Ranking with Agent 1 (The Ranker). The first LLM agent, A_1, receives the query q and candidates C_{initial} to produce a ranked list, C_{ranked}. This interaction, formalised by the prompting function π_1, results in a list containing the TOP_K most relevant candidates. The implementation presented herein sets $TOP\_K = 3$.

$$C_{\text{ranked}} = A_1(\pi_1(q, C_{\text{initial}})) \tag{4.9}$$

where C_{ranked} is an ordered tuple, and each element p'_j (with the prime denoting a ranked candidate) is a member of the initial set C_{initial}.

Step 4: Verification with Agent 2 (The Verifier). The second LLM agent, A_2, audits each ranked pipeline p'_j. The agent is provided with the query q, the pipeline

[1] `https://platform.openai.com/docs/guides/embeddings`

specification, and the metadata of its constituent microservices, which is constructed via the prompting function π_2. The agent's function is to produce a verdict, V_j, that includes an `accept` or `reject` decision and an explanation.

$$\forall p'_j \in C_{\text{ranked}}, \quad V_j = A_2(\pi_2(q, p'_j, \{\text{metadata}(m_k) \mid m_k \in M(p'_j)\})) \tag{4.10}$$

Here, m_k represents an individual microservice, and $M(p'_j)$ is the set of all microservices that constitute the pipeline p'_j. The set of all pipelines that receive an `accept` decision is denoted P_{accepted}.

Step 5: Iterative Refinement. The system evaluates whether the number of accepted pipelines, $|P_{\text{accepted}}|$, meets a predefined success threshold, T_{accept}. Our configuration requires at least one accepted pipeline, thus $T_{\textbf{accept}} = 1$. If this threshold is not met within the maximum number of attempts, $T_{\max}$, the system expands the search space by increasing the retrieval size by an increment, Δk (e.g., to $k_1 = 20$), and repeats Steps 2-4.

$$\begin{gathered}\text{IF } (|P_{\text{accepted}}| < T_{\text{accept}}) \text{ AND } (\text{attempt} < T_{\max}) \\ \text{THEN set } k_{\text{attempt}+1} = k_{\text{attempt}} + \Delta k \text{ and repeat Steps 2-4}\end{gathered} \tag{4.11}$$

Step 6: Final Recommendation Packaging. The workflow concludes by retrieving the full specifications for the final set of recommended pipelines, C_{final}. This produces the final recommendation, R, a ranked list of verified pipelines that includes their executable configuration, `config(`p'_j`)`, and their final verdict V_j.

$$R = \text{rank}\left(\{(p'_j, \text{config}(p'_j), V_j) \mid p'_j \in C_{\text{final}}\}\right) \tag{4.12}$$

The entire recommendation process is consolidated into the procedural logic shown in Algorithm 3.

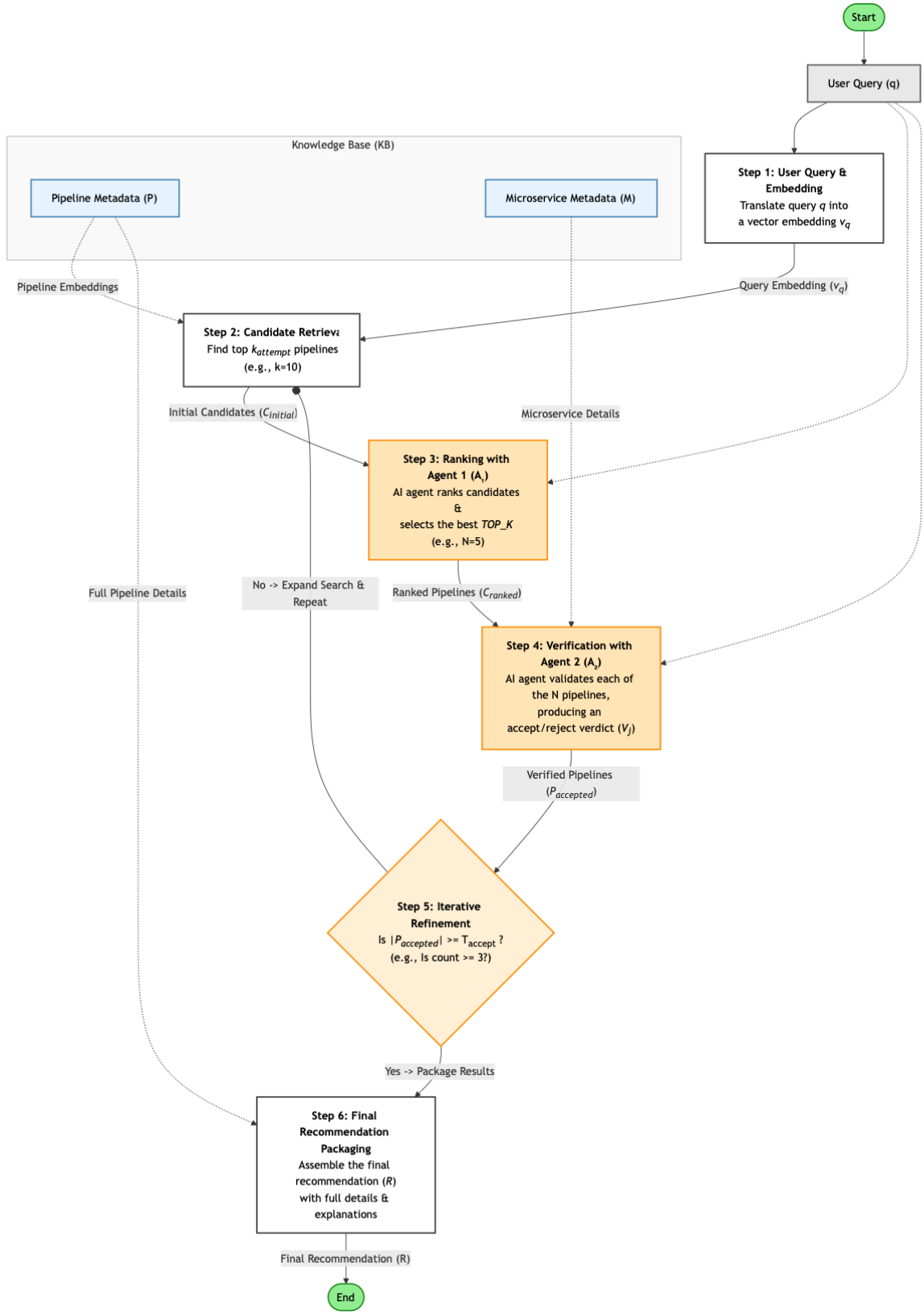

Figure 4.4: Controller's Workflow of the SLEGO LLM Recommender System

Algorithm 3 SLEGO LLM-Based Recommender Algorithm

Require: User query q, Knowledge Base $KB = (P, M)$, embedding model E, ranking agent A_1, verification agent A_2

Require: Hyper-parameters: initial candidate size k_0, ranking cutoff TOP_K, acceptance threshold T_{accept}, maximum attempts $T_{\max}$, candidate increment Δk

Ensure: Ranked list of verified pipelines R

$attempt \leftarrow 0$; $k_{\text{attempt}} \leftarrow k_0$; $P_{\text{accepted}} \leftarrow \emptyset$
while $attempt < T_{\max}$ **and** $|P_{\text{accepted}}| < T_{\text{accept}}$ **do**
  $attempt \leftarrow attempt + 1$
  $v_q \leftarrow E(q)$ ▷ Embed the user query
  $C_{\text{initial}} \leftarrow \textsc{TopKSim}_{k_{\text{attempt}}}(v_q, \{E(d_i)\}_{p_i \in P})$
  $C_{\text{ranked}} \leftarrow A_1(\pi_1(q, C_{\text{initial}}))$ ▷ Keep $\min(TOP\_K, |C_{\text{initial}}|)$
  for all $p'_j \in C_{\text{ranked}}$ **do**
    $V_j \leftarrow A_2(\pi_2(q, p'_j, \{\text{metadata}(m_k) \mid m_k \in M(p'_j)\}))$
    if V_j.decision = `accept` **then**
      $P_{\text{accepted}} \leftarrow P_{\text{accepted}} \cup \{(p'_j, \text{config}(p'_j), V_j)\}$
    end if
  end for
  if $|P_{\text{accepted}}| < T_{\text{accept}}$ **then**
    $k_{\text{attempt}} \leftarrow k_{\text{attempt}} + \Delta k$
  end if
end while
$R \leftarrow \textsc{Rank}(P_{\text{accepted}})$
return R

This multi-agent architecture provides a clear separation of concerns, enabling efficient candidate retrieval, context-aware ranking, and fact-grounded verification. While the core parameters (k_{attempt}, TOP_K, T_{accept}) are configurable, the values specified herein demonstrate a robust operational flow.

4.4.4 Summary

By leveraging a semantic search agent and a verification/explanation agent, SLEGO's LLM-Based Recommender translates user requests into executable workflows. This multi-agent design optimises retrieval accuracy and fosters the reuse of best-practice components.

4.5 User Interaction and Use Cases

SLEGO fosters collaboration among three primary user groups: *Technical Experts*, *Domain Experts*, and *End Users*. Each group has distinct responsibilities, yet all users work within a shared environment that integrates microservices, domain metadata, and low-code tools.

4.5.1 Technical Expert Use Cases

Technical Experts (e.g., data scientists, software engineers) are responsible for the development and maintenance of the microservices that form the core of SLEGO's analytics capabilities. Their workflow involves direct interaction with several key system components. The main use case is adding a microservice.

Figure 4.5 shows the sequence diagram that corresponds to this use case. The Technical Expert uploads a new microservice via the Microservice Management UI. The UI triggers automated validation against SLEGO standards (naming, documentation, parameters); any errors are shown for correction. On successful validation, the artefact is stored in Cloud Storage and its metadata is registered in the Knowledge Base, making it immediately discoverable and usable.

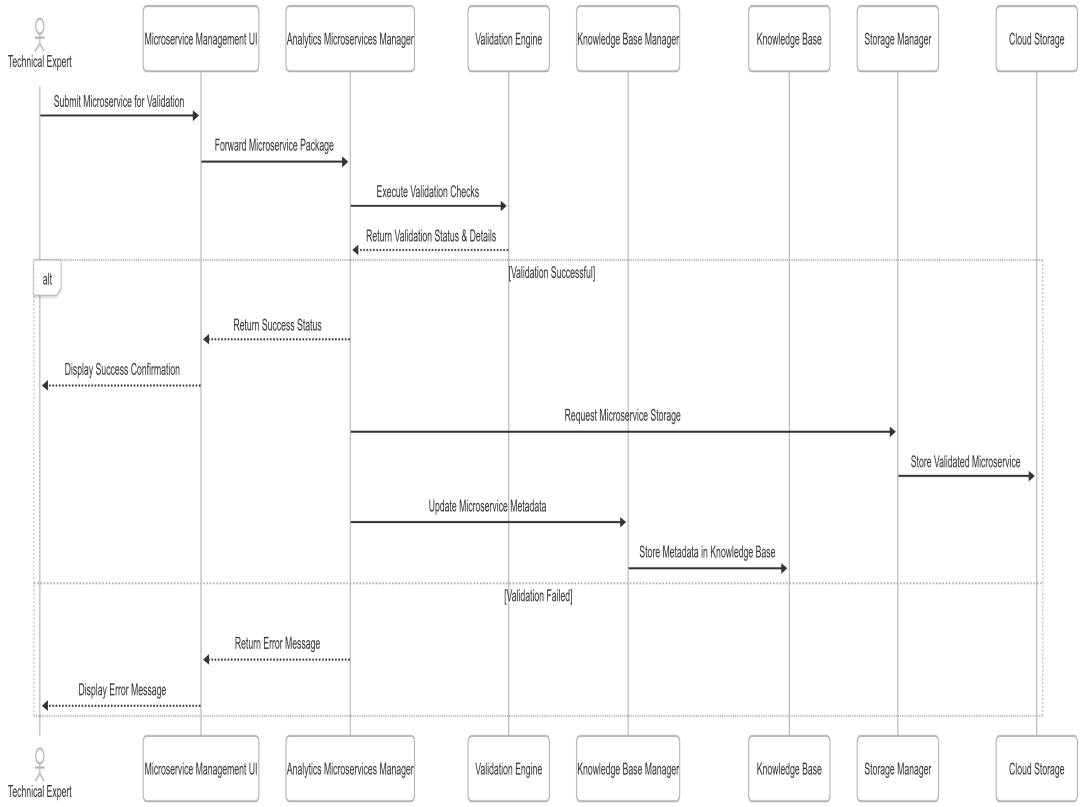

Figure 4.5: Sequence Diagram for adding a microservice by the technical expert.

Beyond the initial creation and validation use case, Technical Experts engage in the **continuous improvement** of the microservice library. This ongoing responsibility includes adding new features, enhancing performance, fixing bugs, or deprecating outdated code as needed. By regularly updating and refining these components, Technical Experts ensure that domain experts and end users have access to a robust, reliable, and up-to-date set of tools for building analytics pipelines within SLEGO.

4.5.2 Domain Expert Use Cases

Domain Experts integrate specialised knowledge, such as industry regulations, common data-cleaning practices, or recommended modelling techniques, into the Knowledge Base. Their use case adding domain specific knowledge involves direct interaction with several

key system components, as illustrated in the sequence diagram in Figure 4.6. The use case operate in two steps:

i. **Knowledge Review:** The Domain Expert uses the Knowledge Management UI to fetch and review details of available microservices and pipeline templates. The UI communicates with the Knowledge Base Manager, which retrieves metadata from the Knowledge Base and presents it to the expert for assessment and identification of opportunities for domain-specific enhancements.

ii. **Annotations and Constraints:** The Domain Expert adds domain-specific annotations (such as valid parameter ranges, recommended defaults, or regulatory caveats) or constraints to microservices or pipelines using the Knowledge Management UI. These updates are submitted to the Knowledge Base Manager, which applies the changes to the Knowledge Base and confirms successful updates back to the expert. This metadata guides users in building valid pipelines that respect domain requirements and industry standards.

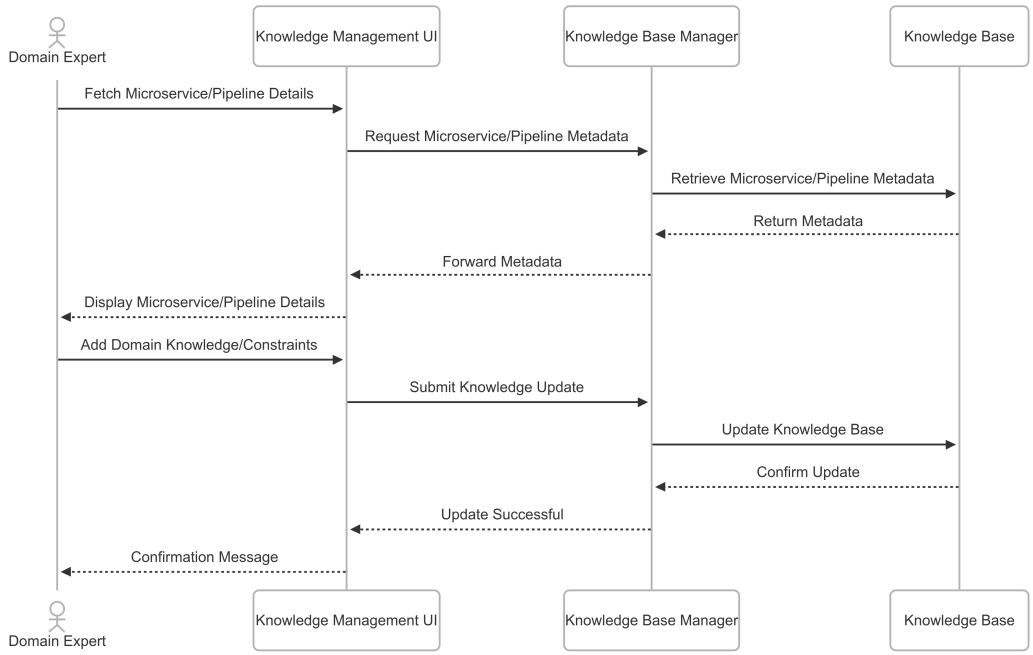

Figure 4.6: Sequence Diagram for adding domain-specific knowledge by the Domain Expert.

While Figure 4.6 emphasises knowledge review and annotations, Domain Experts can also build illustrative pipelines following the same process used by End Users (Figures 4.7–4.10). These expert-created pipelines serve as valuable templates that codify best practices within a specific domain. Any pipeline they create (or adapt from an existing template) can then be saved as a reusable example in the knowledge base (see 4.5.3), thereby helping others address similar domain-specific challenges more effectively.

4.5.3 End User Use Cases

End Users (e.g., business analysts, researchers) utilise the SLEGO system to construct, execute, and manage analytics workflows, focusing on achieving analytical goals with minimal coding. Their use cases involve interaction with various SLEGO components. The typical use cases are:

i. **Data Upload:** End Users typically start by uploading their datasets (e.g., CSV files) via the *Data Management UI*. This UI interacts with the *Storage Manager*, which securely stores the files in *Cloud Storage* and confirms the successful upload.

ii. **Pipeline Exploration (Manual Review):** Users can explore existing solutions by browsing pipeline templates in the *Knowledge Base* using the *Pipeline Assembly UI*. These templates, often created by Domain Experts or other users, represent proven approaches for common analytics tasks. The sequence diagram for this use case is shown in Figure 4.7.

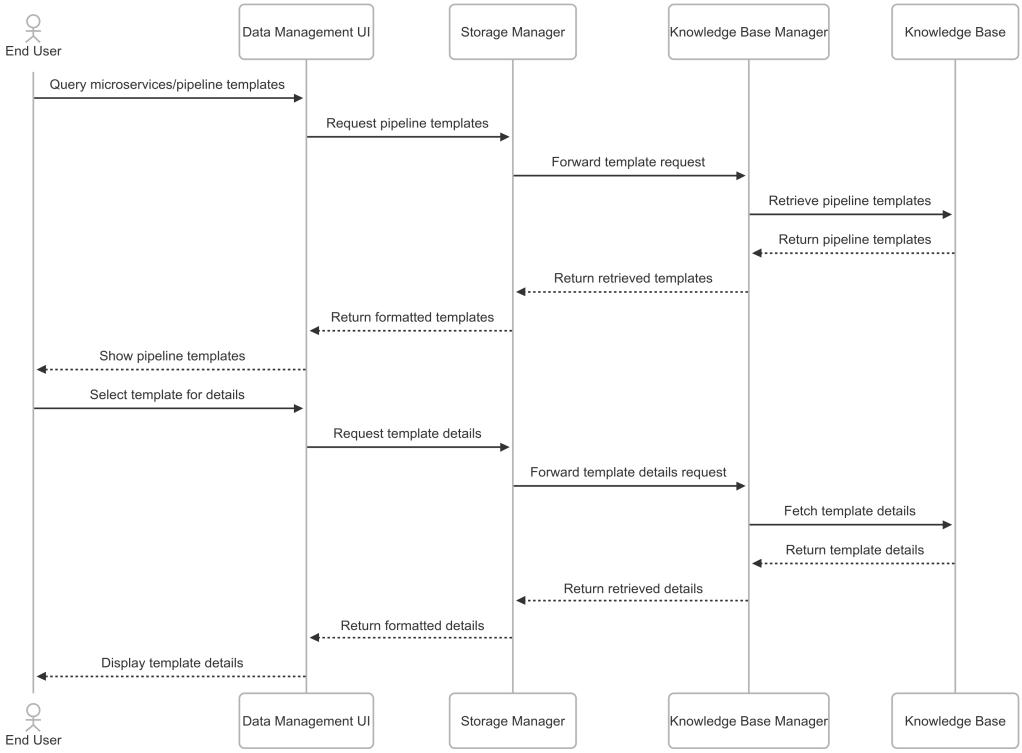

Figure 4.7: Sequence Diagram for Manual Review of Pipeline Templates.

iii. **LLM-Assisted Pipeline Recommendation:** Alternatively, if unsure about the appropriate microservices, users can query the *LLM-Based Recommender* via the *Recommendation Query UI* using natural language. The *LLM-Based Recommender* retrieves relevant components and metadata from the *Knowledge Base* and proposes a suitable pipeline structure. The user can then review, modify, or accept this suggestion. The sequence diagram for this use case is shown in Figure 4.8.

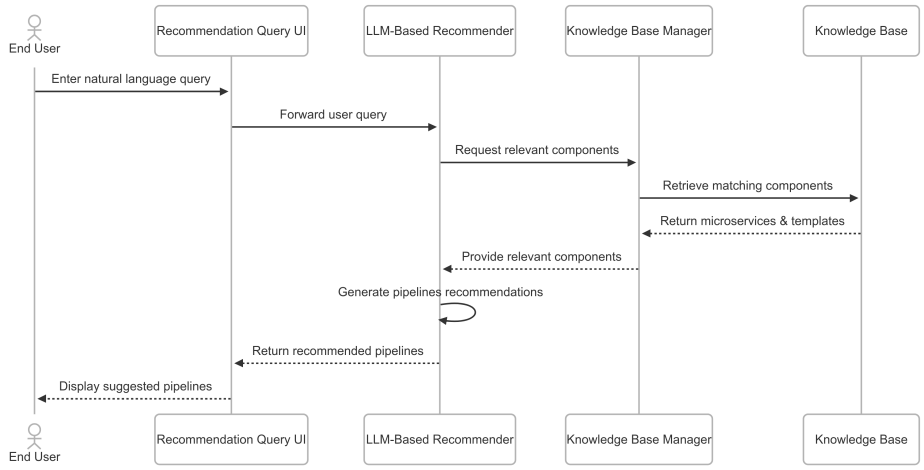

Figure 4.8: Sequence Diagram for LLM-Assisted Pipeline Recommendation.

iv. **Manual Pipeline Assembly and Configuration:** Once a pipeline structure is chosen (from a template, LLM recommendation, or manual design), the End User configures the specific parameters for each microservice using the *Pipeline Assembly UI* (Figure 4.9). This UI allows users to finalise the workflow and adjust settings like hyperparameters or file paths. The sequence diagram is shown in Figure 4.9.

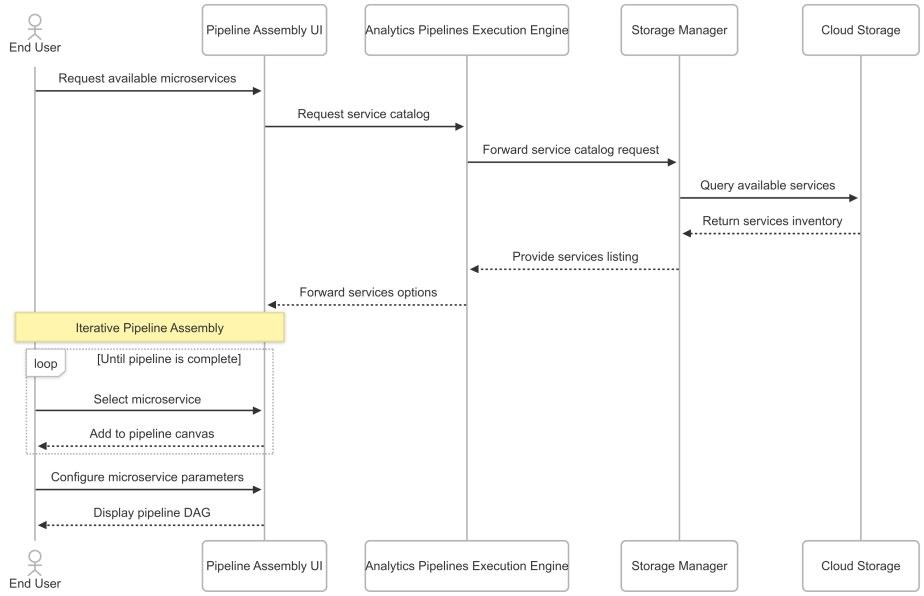

Figure 4.9: Sequence Diagram for Manual Pipeline Assembly and Configuration.

v. **Pipeline Execution and Monitoring:** Finally, the user initiates the pipeline execution via the *Pipeline Assembly UI*, triggering the *Analytics Pipeline Engine*. The engine orchestrates the execution by retrieving the necessary microservices and data, running them in the correct sequence according to the DAG, and managing data flow. Intermediate and final results are stored back in *Cloud Storage* for retrieval and analysis via the *Data Management UI*. The sequence diagram is shown in Figure 4.10.

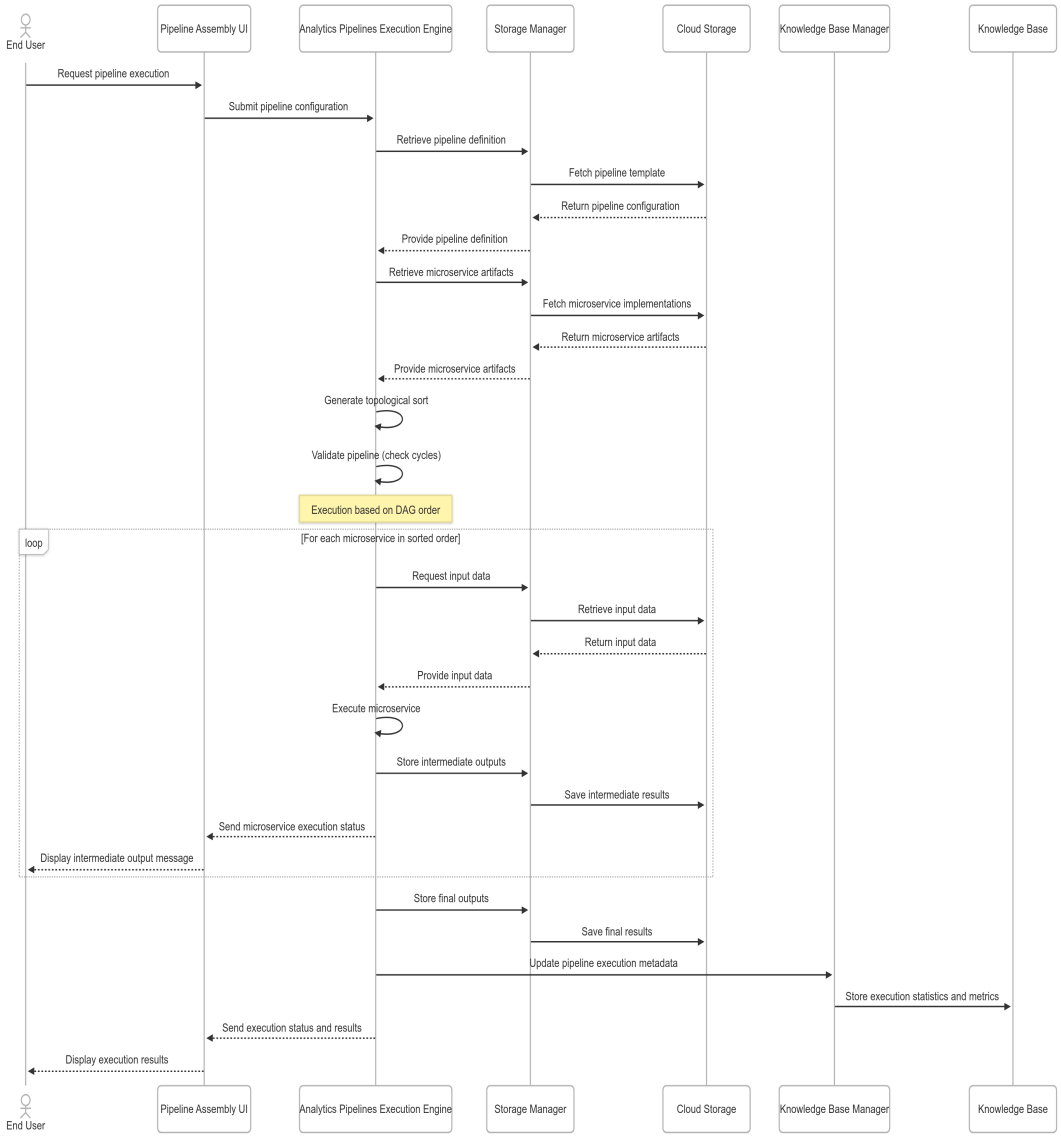

Figure 4.10: Sequence Diagram for Pipeline Execution.

These use cases empower End Users to conduct comprehensive data analysis, from initial data upload to final results, leveraging shared resources and intelligent guidance without requiring deep technical expertise.

4.5.4 End-to-End Collaboration Workflow

SLEGO provides a comprehensive environment for end-to-end collaboration, enabling users with diverse roles and expertise—the End User Group (EUG), Technical Expert Group (TEG), and Domain Expert Group (DEG)—to jointly address data analytics challenges. Figure 4.11 offers a high-level view of this collaborative workflow, illustrating these interactions. At the core of this process is the Knowledge Base (KB), serving as the central repository for shared assets like microservices and pipeline templates. The following list details each numbered step in this collaborative process.

1. **End User Defines Analytical Objective/Problem (Step 1):** The workflow commences when an EUG identifies an analytical need or problem.
2. **End User Seeks Solutions via KB/Recommender (Step 2):** The EUG queries the LLM-based recommender via the Recommendation Query UI (RQU) or directly searches the KB using the Knowledge Retrieval UI (KRU) for existing pipelines or microservices that can address their objective.
3. **Suitable Pipeline Found? (Step 3):** This is a critical decision point.
    - The DEG may optionally provide guidance **(Step 3a)** to the EUG, assisting them in identifying relevant solutions or assessing their applicability.
    - **If YES:** The workflow proceeds along the user-driven application path. The EUG:
        (1) Uploads and inspects input data for the selected pipeline using the Data Management UI (DMU) **(Step 4)**.

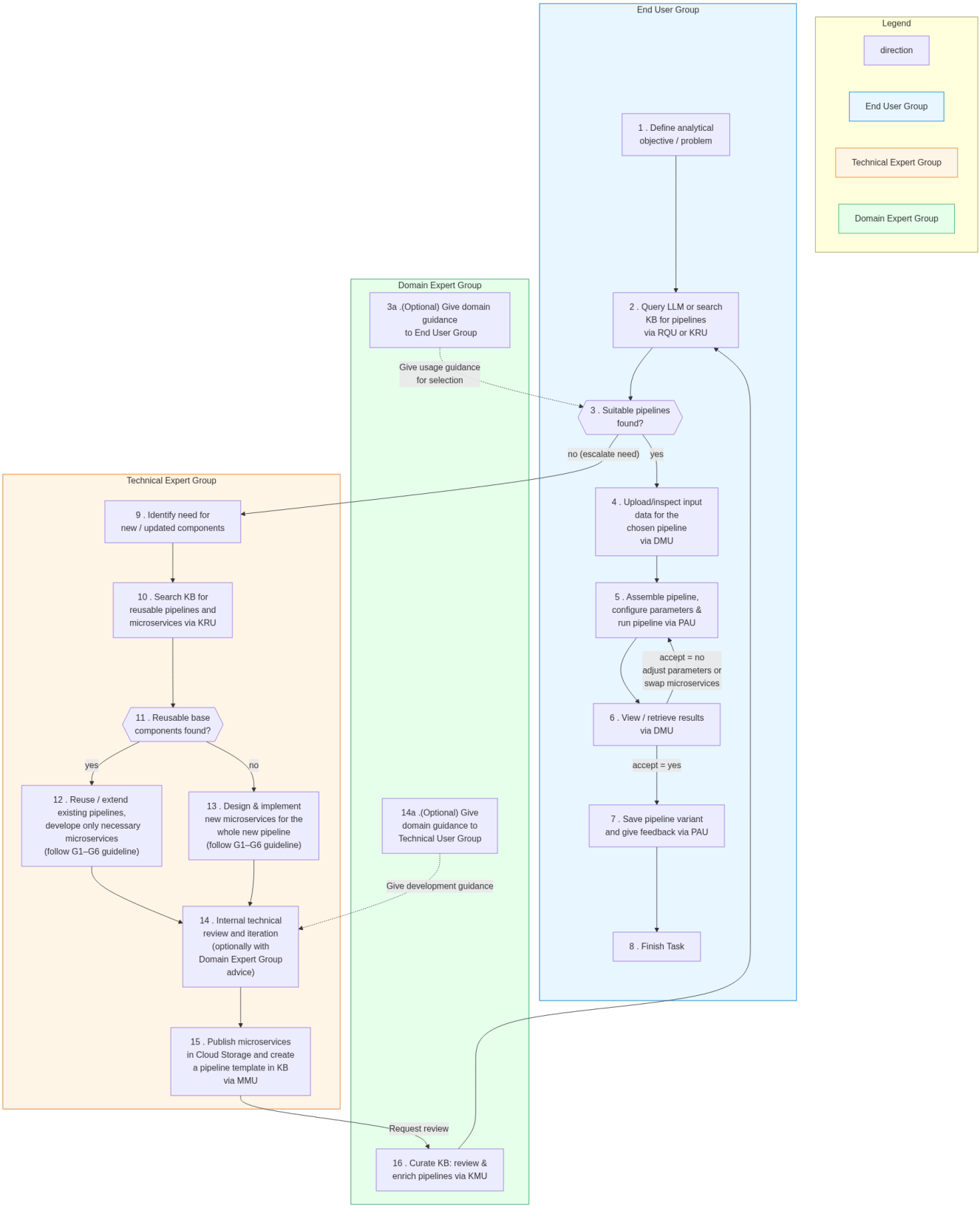

Figure 4.11: End-to-end collaboration workflow in SLEGO.

(2) Assembles and executes the pipeline, adjusting parameters as needed, via the Pipeline Assembly UI (PAU) **(Step 5)**.

(3) Reviews the results of the pipeline execution through the DMU **(Step 6)**. If the results are unsatisfactory, the user can return to the PAU to adjust parameters or swap microservices, initiating an **iterative refinement loop** back to **(Step 5)**. If the results are satisfactory, the user proceeds.

(4) Saves any customized pipeline variant to the KB or provides feedback on components via the PAU, enriching the shared repository **(Step 7)**.

(5) Completes the analytical task using the existing or adapted solution **(Step 8)**.

- **If NO:** If no suitable solution is found, the need is escalated to the TEG, initiating the component creation and enhancement cycle.

4. **TEG Identifies Need for New/Updated Components (Step 9):** The TEG acknowledges the requirement for new or updated analytical components, typically identified through EUG escalation or proactive assessment, to fulfill the user's need.

5. **TEG Searches KB for Reusable Assets (Step 10):** Before building from scratch, TEG members search the KB (e.g., via KRU) for existing microservices or pipeline components that can be reused or extended.

6. **Reusable Base Components Found? (Step 11):** Based on the search, the TEG's path diverges:

   - **If YES:**

     (1) The TEG reuses or extends existing pipelines and microservices **(Step 12)**, developing only the necessary new parts according to SLEGO's development guidelines (G1–G6).

   - **If NO:**

     (1) The TEG designs and implements all required new microservices **(Step 13)**, potentially for an entire new pipeline, from scratch, adhering to development guidelines (G1–G6).

13. **TEG Conducts Technical Review and Iteration (Step 14):** All newly developed or modified components, whether extended or built from scratch, undergo internal technical review and iterative refinement. The DEG may optionally provide input **(Step 14a)** to ensure domain relevance and correctness of the new components.

14. **TEG Publishes Artifacts to Cloud Storage and Updates KB (Step 15):** In this final TEG step, finalized microservices are validated (e.g., by the Validation Engine). They are then published to Cloud Storage (CS), and, crucially, a **baseline pipeline template** incorporating these microservices is created and updated in the KB (e.g., via the Microservice Management UI (MMU)). This ensures a complete, working solution is delivered, making artifacts accessible for EUG use and DEG curation.

15. **DEG Curates and Enriches Knowledge Base (Step 16):** The DEG continuously reviews, annotates, and enriches new and existing pipelines and microservices in the KB (e.g., via the Knowledge Management UI (KMU)). This curation involves adding domain-specific context, best practices, usage examples, and constraints. This transforms technical artifacts into trusted, easily discoverable, and effectively usable solutions, directly benefiting EUGs in subsequent searches (as indicated by the arrow from **Step 16** to **Step 2** in Figure 4.11).

This collaborative framework, with its cycles of creation, curation, and application centered around the Knowledge Base, empowers all user groups within the SLEGO ecosystem. End Users can efficiently address analytical problems by leveraging well-documented, context-rich solutions discovered via search or recommendation, and can adapt or reuse them as needed. Technical Experts accelerate their development by building upon a reusable foundation of microservices and pipeline templates, focusing new efforts where most impactful. Domain Experts ensure the ongoing quality, discoverability, and domain relevance of all shared assets through active curation, annotation, and optional guidance. Collectively, this synergy enables SLEGO to continuously evolve as a robust, reliable, and

ever-improving organizational knowledge repository for analytics.

4.6 SLEGO's Unique Combination

SLEGO integrates all architectural pillars introduced in Section 4.3. Its strength is therefore not a single feature, but the *synergy* of the whole design.

- **Modular Microservices with Strict Design Guidelines**: SLEGO enforces a set of theoretical guidelines (G1-G6) for microservice development, ensuring pure functionality, explicit data contracts, deterministic behaviour, and seamless composability. This creates a reliable ecosystem of reusable components that can be combined into complex pipelines without unexpected interactions.
- **Semantic Knowledge Base and LLM-Powered Recommendation System for Contextual Enrichment and Guidance**: The platform includes a structured knowledge base specifically designed to support LLM-based recommendations, capturing not just technical metadata but also domain-specific annotations, usage guidelines, and task applicability, enabling intelligent discovery and adaptation of components across diverse analytical needs.
- **Role-Based Interfaces, Workflows, and End-to-End Collaboration Loop**: Tailored user use cases for Technical Experts (component creation and validation), Domain Experts (knowledge curation), and End Users (pipeline assembly and execution) facilitate seamless collaboration, ensuring each role contributes effectively without overwhelming complexity. SLEGO supports a continuous cycle of creation (new microservices), curation (metadata enhancement), and consumption (pipeline building and reuse), promoting knowledge accumulation and platform evolution over time.

In summary, SLEGO offers a holistic solution to the collaboration problem in data analytics.

4.7 Summary and Implications

This chapter has introduced the SLEGO system, outlining its design principles, comprehensive architecture, defined user roles, pipeline engine, and the theoretical underpinnings that ensure its reliability and operational efficiency. SLEGO is engineered to tackle prevalent challenges in the realm of collaborative data analytics. It achieves this by systematically integrating a set of core design objectives with specific, tangible platform components.

The following table (Table 4.3) provides a detailed mapping, illustrating how each SLEGO design objective is supported by its principal design components. This linkage underscores the system's capacity to deliver a more inclusive and cumulative approach to data analytics.

Objective	**Concise Definition**	**SLEGO Design Techniques**
End-to-End Collaboration Support	A platform and workflow that facilitates traceable hand-offs among EUG, TEG, and DEG roles throughout the analytics lifecycle.	• Role-oriented workflow support • Low-code UI platform • Shared cloud storage and knowledge base for artefacts and knowledge exchange
Modular & Extensible Architecture	Every analytic step is an independent microservice that can be swapped, reused, or hot-plugged; new capabilities join the platform with no rewrites.	• Microservice and modular architecture • DAG-based pipelines with strict I/O contracts • Design guidelines G1–G6 • Validation engine for pipeline schema checks
AI-Guided Knowledge Discovery	The LLM-based recommender suggests the best-fit pipelines and components, thereby reducing the effort required to search for solutions from the knowledge base.	• LLM-based recommender • Multi-agent RAG design • Structural semantic knowledge base • Prompt templates for users

Table 4.3: SLEGO's three retained design objectives and the design techniques that fulfil each one.

The following chapter details the implementation and deployment of the SLEGO framework. It describes the integration of the theoretical pipeline engine with AI-assisted usability features designed to facilitate the application of advanced data analytics.

Chapter 5

Prototype Implementation

This chapter outlines the implementation of **SLEGO**, with particular attention to its low-code user interface, microservice repository, and integration with a large language-model (LLM) API. Section 5.1 describes the system architecture, its core components, and the associated data stores. Section 5.3 presents the user interface and highlights the features available to the **End User**, **Technical Expert**, and **Domain Expert** roles. Section 5.4 then walks through the construction of an example analytical pipeline to show how these roles collaborate in practice.

The full implementation is open-source and available on GitHub.[1]

5.1 System Overview and Deployment

This section details the system's deployment and components, building upon the overall SLEGO architecture discussed in Section 4.3.

- **UI Development:** The SLEGO front end is built primarily with Python's **Panel**[2]

[1] `https://github.com/alanntl/SLEGO-Project`

[2] `https://panel.holoviz.org`

library, which provides a browser-based dashboard. This framework supports drag-and-drop microservice selection, real-time parameter adjustments, and intuitive visual feedback for users.

- **Microservices:** Each microservice is defined as a Python function that adheres to standardized naming, docstrings, parameters, and I/O conventions. In some cases, the Python function simply acts as a *wrapper* for external code—invoking scripts in other programming languages, running compiled programs stored in a shared data repository, or calling external APIs. This design ensures consistent discovery and reuse across different pipelines, regardless of the technology stack. Approved microservices reside in a shared repository and are indexed in the SLEGO knowledge base.

- **LLM Recommender:** SLEGO integrates with OpenAI's `gpt-4.1-mini`[3] API to guide users in constructing pipelines. When a user issues a query, SLEGO fetches relevant pipeline templates or microservices from the Knowledge Base, provides them to the LLM as context, and refines the model's output to match SLEGO's parameter requirements.

- **Cloud Storage and Knowledge Base:** Raw and intermediate files are stored in a shared cloud folder. An SQL database (SQLite)[4] hosts structured metadata about microservices and user pipelines, forming the core of SLEGO's Knowledge Base.

- **Containerization and Deployment:** The entire SLEGO application is containerized using Docker[5] and can be deployed on Amazon Web Services (AWS)[6]. This setup simplifies infrastructure management and ensures scalability, allowing teams to run SLEGO seamlessly in cloud environments.

[3] `https://platform.openai.com/docs/models/gpt-4.1-mini`

[4] `https://www.sqlite.org/`

[5] `https://www.docker.com/`

[6] `https://aws.amazon.com/`

5.2 Pipeline Representation and Execution

SLEGO implements a pipeline execution engine that transforms the user-created graphical workflows into efficiently executable code. This section explains the internal representation of pipelines and how they are processed for execution.

5.2.1 Internal Pipeline Representation

As discussed in Section 4.2, SLEGO represents analytics pipelines as Directed Acyclic Graphs (DAGs). The implementation translates these conceptual graphs into concrete data structures:

- **Pipeline Structure:** Each pipeline is internally represented as a JSON object containing metadata (name, description, creation date) and a list of microservices with their configurations and dependencies.
- **Graph Construction:** When a user builds a pipeline in the visual interface, SLEGO dynamically constructs an adjacency list representation of the DAG, where each node is a microservice and each directed edge represents a data dependency.
- **Dependency Tracking:** Input/output dependencies are explicitly recorded. For each microservice, SLEGO captures which outputs from preceding microservices serve as its inputs. This information is essential for determining execution order.

The code snippet in Appendix B.1 illustrates the internal JSON representation of a simple analytics pipeline.

5.2.2 Implementation of Microservices as Python Functions

Each SLEGO microservice is a *single Python function*. Table 5.1 shows how the six theoretical guidelines (G1–G6, Section 4.2.3) translate into concrete coding practices. After

the table, a minimal code template demonstrates all rules in action.

Table 5.1: From theory (left) to concrete Python features (middle) and where to see them (right).

Guideline	Python feature(s) that realise it	Seen in template
G1 Single responsibility	Verb-based function name; exactly one transformation in body.	`sma_col = ...` comment # G1
G2 Explicit data contract	First two parameters `inputs, outputs` — both typed `Dict[str,str]`; each path/key documented in docstring.	Signature lines with # G2 Docstring `inputs / outputs` block
G3 Pure parameterisation	All behaviour toggles surfaced as typed parameters with defaults (`column: str="Close"`, `window: int =20`).	Signature lines with # G3
G4 Stateless & retry-safe	Read-only input (`pd.read_csv`); deterministic output write (`to_csv`); no global variables.	Lines tagged # G4
G5 Acyclic flow	Function contains *no calls* to other microservices or the scheduler.	Absence of such calls; comment # G5 in prose above template
G6 Clear documentation	NumPy-style docstring with **Summary**, **Parameters**, **Returns**.	Triple-quoted docstring block; comment # G6

Putting the guidelines into practice. Table 5.1 maps the abstract principles G1–G6 (defined in Section 4.2.3 and revisited in Section 4.2.3) to concrete, line-level Python conventions. Listing 5.1 then instantiates every rule in a 20-line "single-moving-average" (`sma`) microservice. The function header exposes a *data contract* via the typed `inputs` and `outputs` dictionaries (G2) and surfaces all tunable behaviour as explicit, defaulted parameters (G3). Inside the body, exactly one transformation is performed (`rolling().mean()`), satisfying the single-responsibility rule (G1), while read-only input and deterministic output ensure stateless, retry-safe execution (G4). No other microservices are invoked, keeping the overall pipeline acyclic (G5), and a short NumPy-style docstring documents purpose, parameters, and returns (G6). Because every approved microservice follows this pattern, SLEGO can validate, version, and compose them automatically, guaranteeing that larger pipelines inherit the same modularity and reproducibility properties.

```python
import pandas as pd
from typing import Dict

def sma(
    inputs: Dict[str, str],            # G2
    outputs: Dict[str, str],           # G2
    column: str = "Close",             # G3
    window: int = 20                   # G3
) -> str:
    """Simple moving average (G6).

    Parameters
    ----------
    inputs : {"csv": "..."}          # contract (G2)
    outputs: {"csv": "..."}
    column  : target column name
    window  : rolling window length
    """
    df = pd.read_csv(inputs["csv"])   # G4 read-only
    sma_col = f"SMA_{column}"
    df[sma_col] = df[column].rolling(window).mean()  # G1
    df.to_csv(outputs["csv"], index=False)           # G4 deterministic
    return f"{sma_col} written to {outputs['csv']}"
```

Listing 5.1: Minimal SMA microservice instantiating guidelines G1–G6

From Microservice Implementation to Pipeline Execution. In practice, the SLEGO user interface constructs a pipeline by assembling references to these standardised Python microservices. As detailed in the theoretical representation (subsection 4.2.4), the user's choices are compiled into a JSON format input (also compatible to Python Dictionary format). This format, which lists the chosen microservices and specifies their `inputs`, `outputs`, and `parameters`, becomes the executable blueprint for the entire workflow. The execution engine then parses this JSON, infers the DAG, and orchestrates the execution of the individual microservices in the correct order.

5.2.3 Execution Engine

At runtime SLEGO executes every pipeline in four consecutive steps:

1. **Graph construction** – the JSON specification pipeline is turned into a directed graph using Python `NetworkX`[7] library. Each microservice is a node $v \in V$; every declared data dependency adds an edge $(u, v) \in E$.
2. **Topological sort** – Kahn's algorithm (Algorithm 1) runs on the graph pipeline to produce a valid execution order and to stamp *layers* (all nodes dequeued in the same iteration share a layer and can execute concurrently).
3. **Validation** – for the sorted list the engine applies the two-phase procedure of Section 4.2.5: (i) check that every initial input artefact exists and matches the documented schema, (ii) verify that all parameters are present, type-correct, and satisfy domain constraints stored in the Knowledge Base. Any failure aborts the run with a detailed report.
4. **Execution** – Tasks are handed to Dask[8], which runs them in parallel when multiple cores are available or one-by-one on a single core, while still queuing work, retrying failures, and reporting clear errors.

These four stages—*NetworkX graph, Kahn sort, stringent validation, and Dask (or sequential) execution*—guarantee both correctness and scalability for SLEGO pipelines.

5.3 SLEGO User Interfaces

Built with the **Panel** library, SLEGO's interface accommodates three primary user roles: end users, technical experts, and domain experts. Sections 5.3.1–5.3.3 describe how the interface is adapted for each role.

[7] `https://networkx.org/`

[8] `https://dask.org/`

5.3.1 End User Interface

End Users work with a straightforward, low-code dashboard to construct and execute analytics pipelines. Figure 5.1 highlights the main interface:

1. **Data Management UI:** A dataspace browser allows End Users to upload new files or search for existing ones. These files are automatically recognized as potential inputs for downstream microservices.

2. **Pipeline Assembly UI:** This interface enables pipeline creation and execution through several components:

   a. **Assembly:** By dragging and dropping microservice blocks (or selecting them from dropdown menus), End Users can assemble a pipeline. Parameter widgets such as text boxes or sliders enable them to specify necessary settings.

   b. **Execution and Monitoring:** When End Users click the *Run Button*, SLEGO executes each microservice in real time. A log panel displays progress or errors, and partial outputs are shown immediately.

   c. **Results and Saving:** Final pipeline outputs can be viewed, downloaded, or shared through the Data Management UI. Users can also save pipeline configurations for future reuse.

3. **Recommendation Query UI:** A dedicated query box lets End Users type natural language requests (e.g., "Calculate moving averages of stock prices"). Based on this input, SLEGO automatically suggests a sequence of microservices with recommended parameter values.

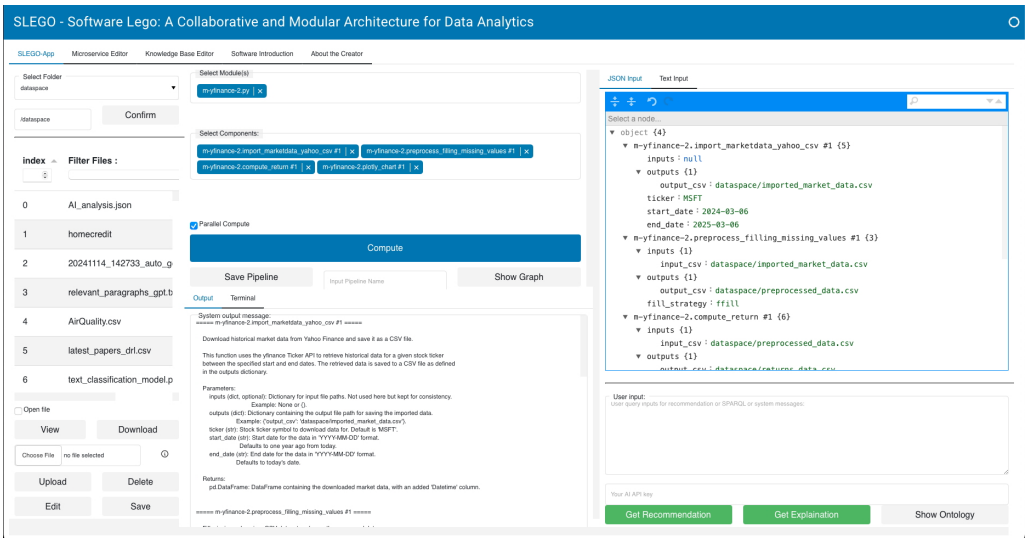

Figure 5.1: Data Management UI (left sidebar) and Pipeline Assembly UI overview.

5.3.2 Technical Expert Interface

Technical Experts, such as data scientists or software engineers, extend SLEGO's functionality by creating or refining microservices through the Microservice Management UI. Figure 5.2 provides an overview:

1. **Microservice Upload:** Technical Experts can upload Python files (modules) containing new microservices along with example datasets for testing. These modules become available for selection in the End User interface dropdown menus. SLEGO immediately verifies compliance with standardized docstring formats, parameter naming conventions, and input/output references.

2. **Validation and Testing:** The system automatically executes the uploaded microservice with the provided example datasets to verify functionality. Upon successful validation, SLEGO extracts the function signature, docstring, and parameter specifications, then updates the knowledge base with both the microservice and its verified test cases.

3. **Deployment Confirmation:** Validated microservices become available in the shared

catalog for all users, with messages showing they have passed quality checks and example use cases. Each Python file uploaded by Technical Experts appears as a separate module in the End User interface, making its microservices available for pipeline creation.

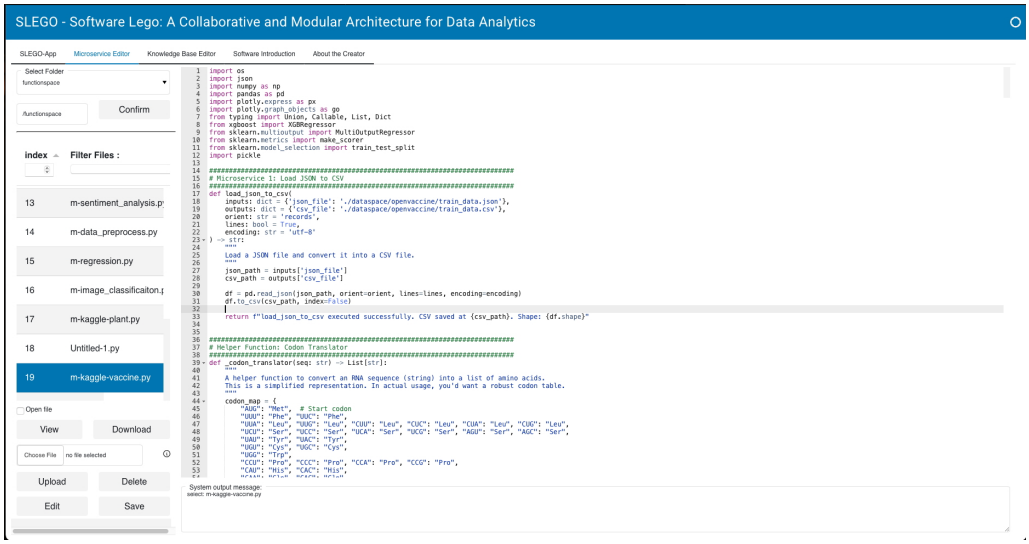

Figure 5.2: Microservice Management UI overview.

5.3.3 Domain Expert Interface

Domain Experts contribute specialized domain knowledge by annotating existing microservices and creating domain-focused pipeline templates. Figure 5.3 shows a configurable table widget enabling the following capabilities:

1. **Knowledge Base Table View:** Domain Experts can browse all existing microservices or pipelines in a table and add or update annotations (e.g., parameter defaults, domain-specific details).

2. **Pipeline Enrichment:** Experts can load an existing pipeline and attach guidelines, recommended transformations, or usage notes, creating "Verified Templates" that others can adopt.

3. **Shared Templates:** Once saved, these verified or annotated pipelines appear prominently for End Users, integrating best practices into subsequent workflows.

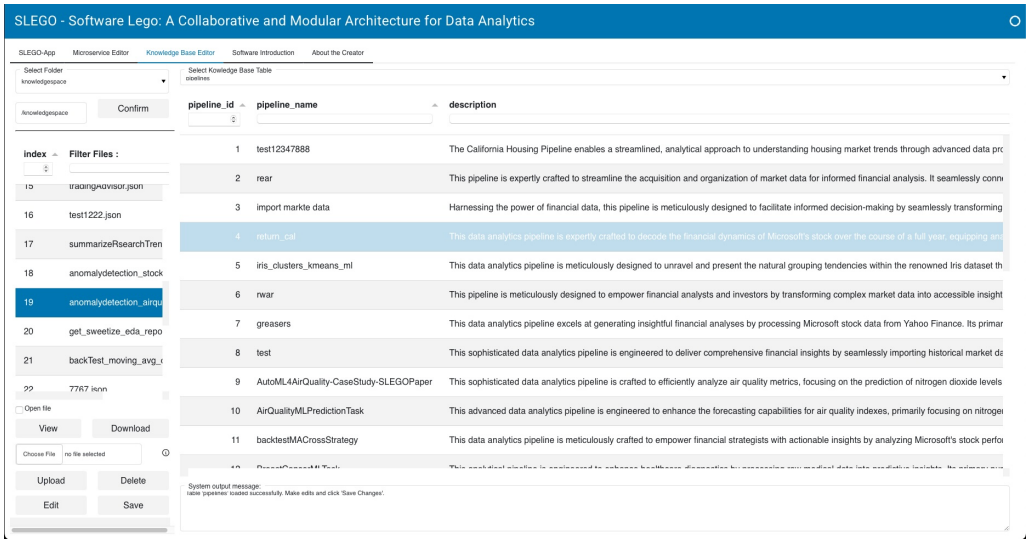

Figure 5.3: Knowledge Management UI overview

By unifying these role-specific interfaces into a single platform, SLEGO fosters streamlined collaboration among team members with varying expertise.

5.4 Illustration of Building Analytical Pipelines with SLEGO

This section demonstrates how an end user builds a basic financial data analysis pipeline to calculate and visualize stock returns. We assume the technical expert has already contributed financial analysis microservices to the repository (see Appendix B.1 for details), and the domain expert has annotated relevant metadata in the knowledge base with appropriate default values for financial time series analysis (such as recommended filling methods for missing stock data and optimal visualization settings for financial charts).

1. Accessing the SLEGO Interface. In the main SLEGO dashboard, the user chooses *SLEGO-App* from the tab (see Figure 5.4). A data management panel (see Figure 5.5)

presents existing files and an option to upload a new CSV (`imported_market_data.csv`), which will be used in the pipeline.

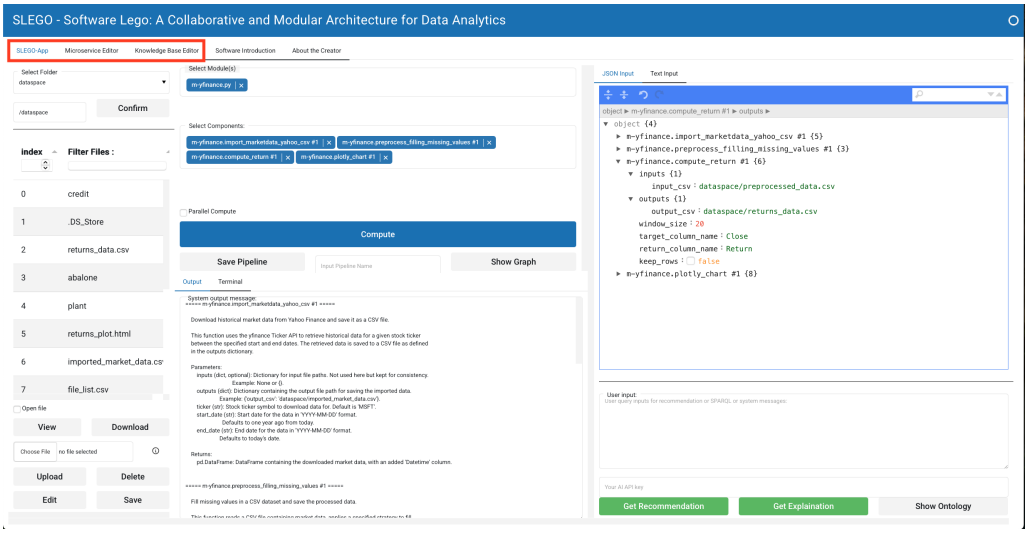

Figure 5.4: Selecting `SLEGO-App` in the main interface.

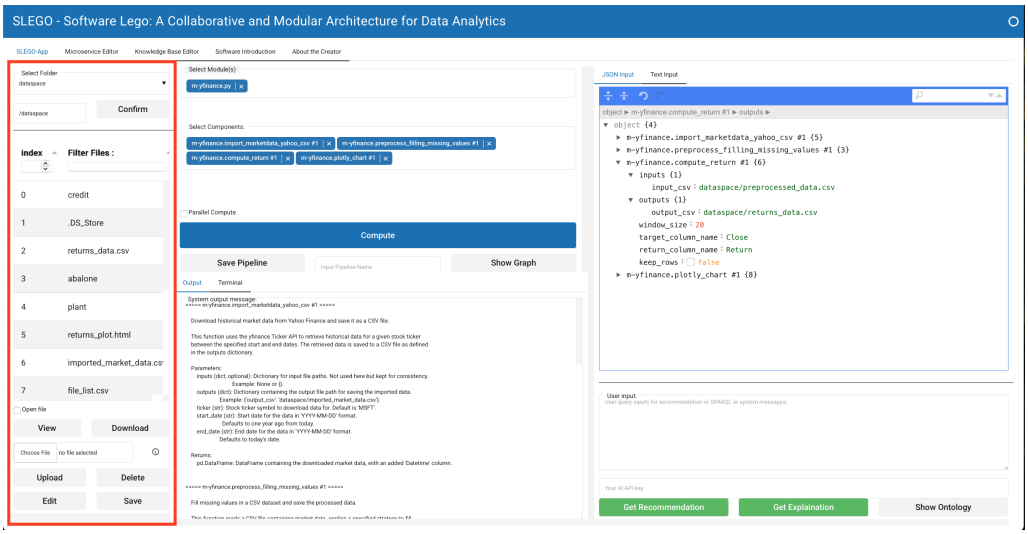

Figure 5.5: Data management panel for uploading or downloading files.

2. Selecting a Module. At the top of the interface, a *Module* dropdown (see Figure 5.6) lists Python modules (Python files containing microservices that were previously uploaded by technical experts through the Microservice Management UI) contributed by Technical

Experts. In this example, the user chooses `m-yfinance.py`, which provides microservices for financial data analysis.

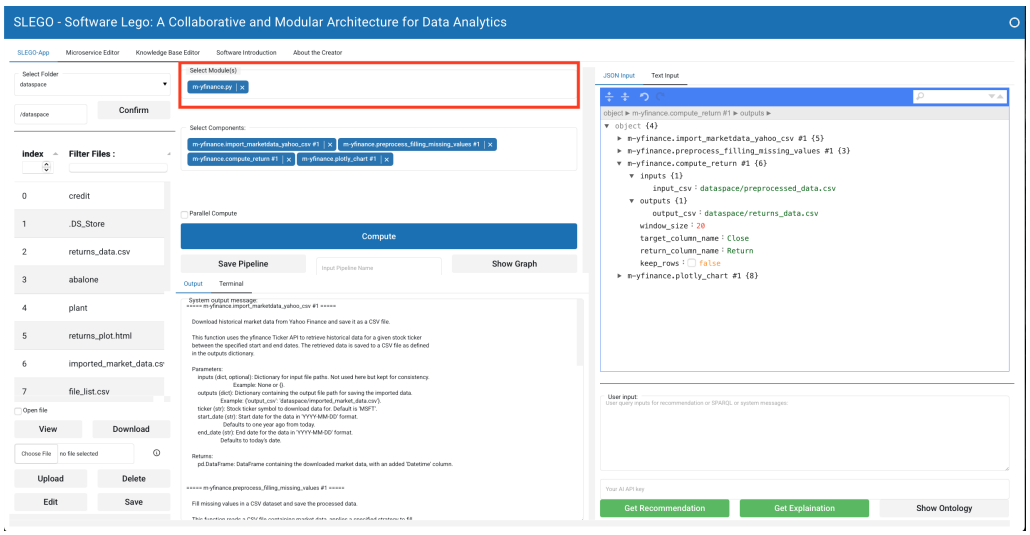

Figure 5.6: Selecting the `m-yfinance.py` module.

3. Choosing Microservices. After loading `m-yfinance.py`, a *Microservices* dropdown menu (see Figure 5.7) shows available microservices. Three are selected in sequence:

1. `preprocess_filling_missing_values`
2. `compute_return`
3. `plotly_chart`

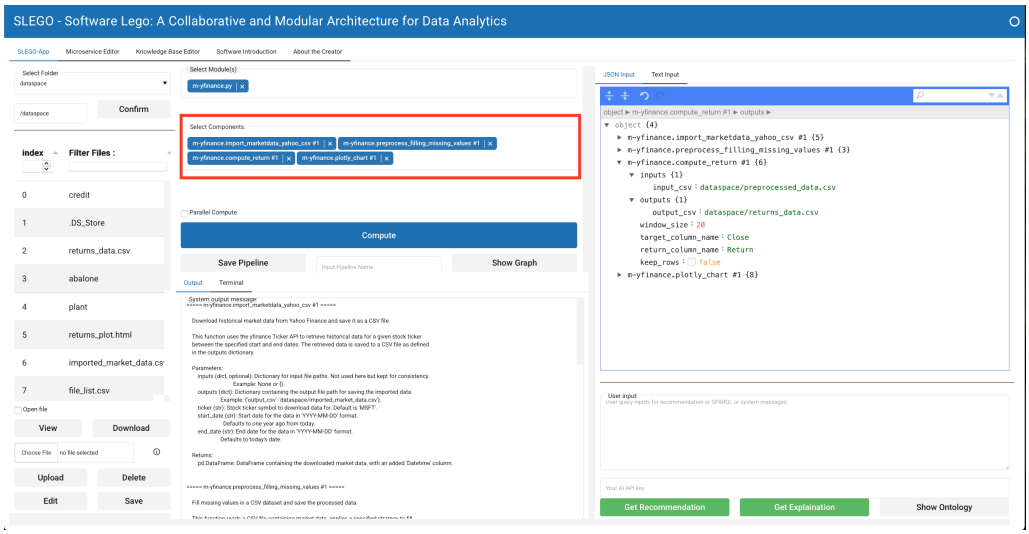

Figure 5.7: Selecting three microservices from `m-yfinance.py`.

4. Configuring the Pipeline. The selected microservices appear in a right-side box (Figure 5.8). Clicking any microservice expands a parameter form that users can adjust as needed. Alternatively, users may switch to a *Text Input* tab to edit the JSON configuration directly (see Appendix B.1).

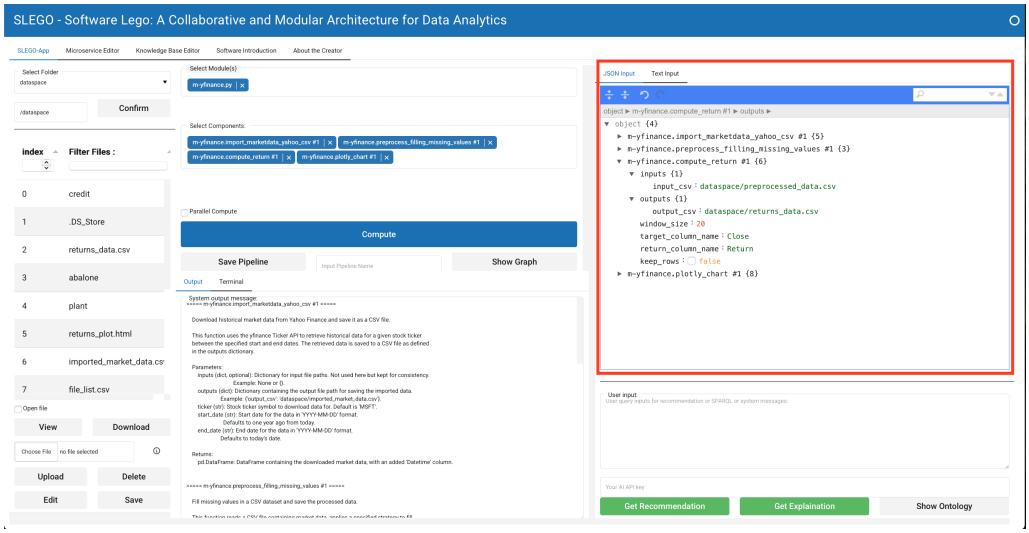

Figure 5.8: Adjusting parameters for each microservice in the pipeline.

5. Executing the Pipeline. After parameter configuration, clicking *Compute* (Figure 5.9) runs the pipeline. A real-time log indicates microservice progress or errors.

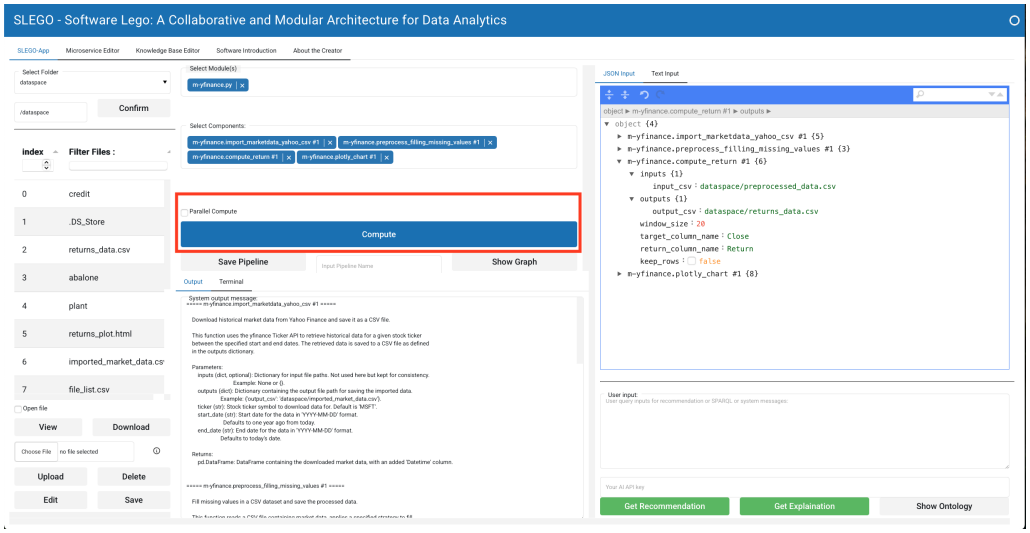

Figure 5.9: Executing the configured pipeline.

If the pipeline completes successfully, users can preview outputs (Figure 5.10) such as plots, partially processed data, or the final CSV.

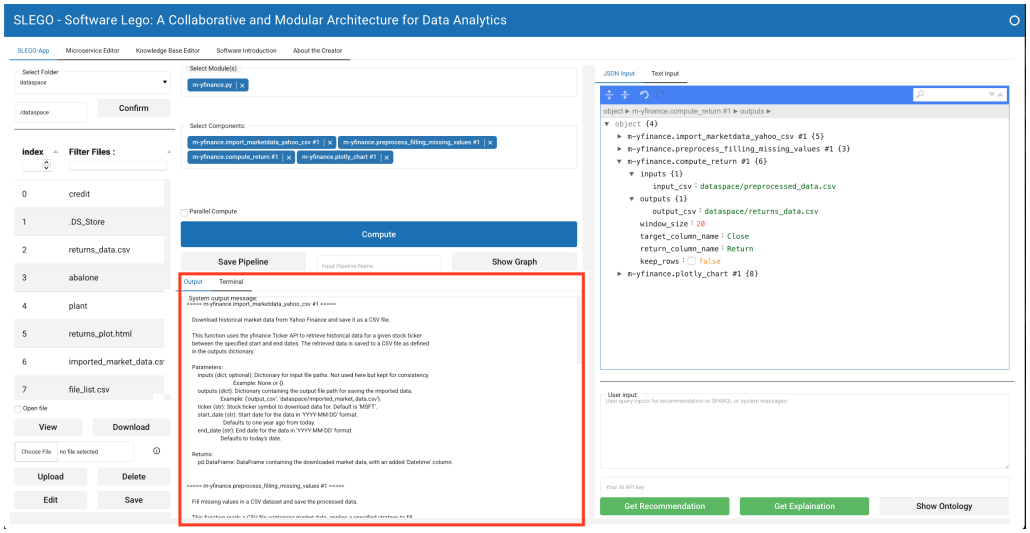

Figure 5.10: Real-time logs and output previews.

6. Saving and Visualizing Pipelines. Users can save the entire pipeline to the knowledge base under a chosen name, using the *Save Pipeline* button (Figure 5.11). They can also select *Show Graph* to display a Directed Acyclic Graph (DAG), offering a bird's-eye view of the pipeline structure. The show graph function can be performed before executing the pipeline.

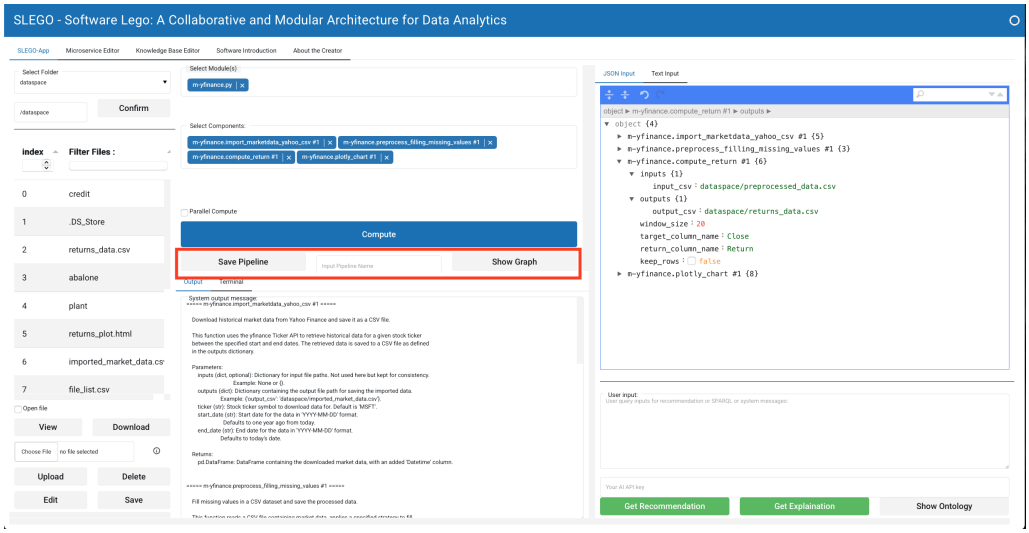

Figure 5.11: Saving a completed pipeline and viewing its DAG.

7. Semantic Query and LLM-Based Recommendations. Optionally, an *LLM Recommender* helps users discover relevant microservices (Figures 5.12–5.13). By typing an analytic goal and clicking *Get Recommendation*, the GPT-4 agent suggests an initial pipeline configuration.

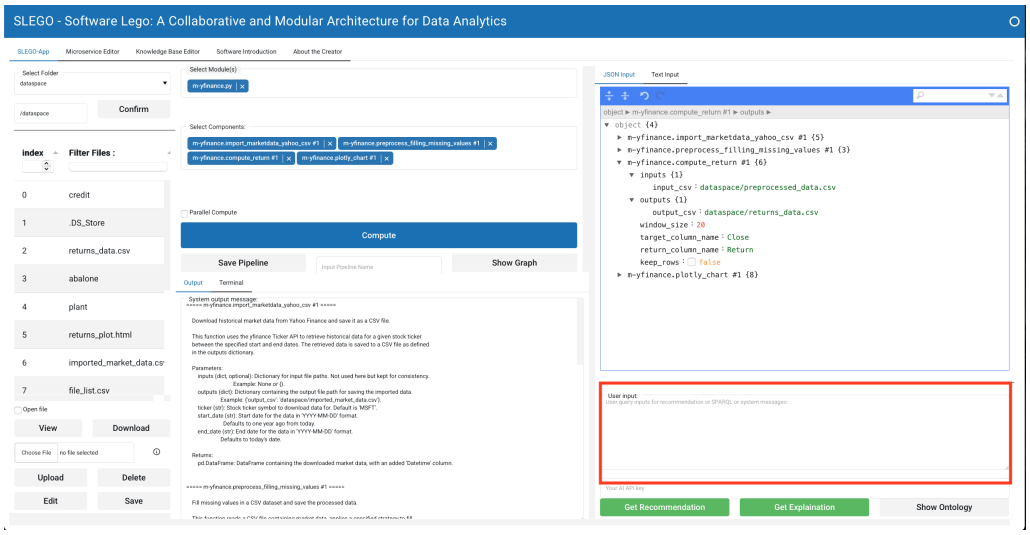

Figure 5.12: Entering a semantic query for LLM-based pipeline suggestions.

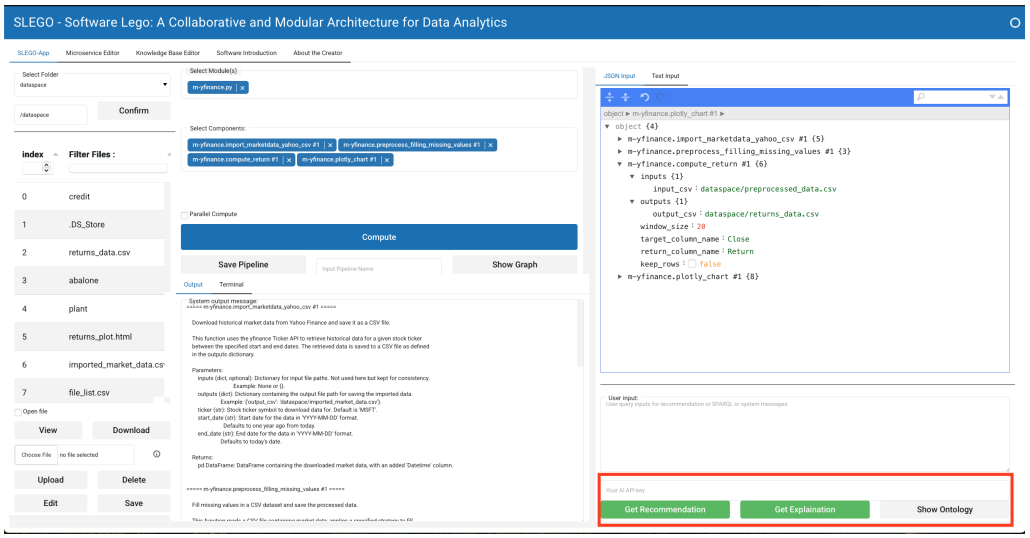

Figure 5.13: Recommendations generated by the LLM-based agent.

Through this process, SLEGO integrates domain knowledge, microservices, and LLM-based recommendations to facilitate efficient, user-friendly pipeline construction.

Chapter 6

Experiments and Evaluation of SLEGO

This chapter presents a series of three experiments structured to evaluate the SLEGO platform against its core design objectives. SLEGO aims to provide a collaborative data analytics architecture, augmented by an LLM powered recommender, tailored for diverse users. In accordance with the research objectives, the architecture emphasises modularity, reusability, low-code accessibility, systematic knowledge management, AI-driven guidance, and a comprehensive end-to-end collaboration workflow. The experiments detailed herein demonstrate these capabilities across a range of real-world data analytics and machine learning scenarios.

All experimental assets—datasets, pipeline definitions, diagrams and analysis scripts—are openly available on GitHub.[1]

[1] `https://github.com/alanntl/SLEGO-Project`

6.1 Evaluation Strategy

6.1.1 SLEGO Design Objectives for Evaluation

The design of the SLEGO platform was guided by three key objectives, as detailed in Section 4.1, aimed at addressing prevalent challenges in collaborative data analytics. As outlined by Hevner et al. (2004) [32], a critical component of Design Science Research is the rigorous evaluation of the designed artefact against its intended purpose. The demonstration of the SLEGO platform in Chapter 5 serves as a crucial step in this research, and it is followed by the comprehensive evaluation presented in this chapter. The experiments in this chapter are structured to assess SLEGO's performance against these objectives, summarised in Table 6.1.

Table 6.1: Consolidated SLEGO Design Objectives, their Evaluation Focus, and Collaborative Significance

ID	Design Objective	Evaluation Focus	Evaluation Rationale
O1	**End-to-End Collaboration**	Demonstrate the architectural feasibility of supporting traceable workflows among TEG, DEG, and EUG roles, showing that complex analytics pipelines can be constructed through the low-code interface following the designed collaborative process.	Validates that the multi-agent LLM architecture can effectively match user intent to appropriate analytical assets, providing the technical foundation for intelligent assistance in pipeline discovery.
O2	**Modular & Extensible Architecture**	Demonstrate successful reuse of microservices across different domains and the integration of technologically diverse components, while showing that the Knowledge Base can store and retrieve component metadata and pipeline configurations.	Validates that the modular design enables component reusability and extensibility across contexts, with the Knowledge Base providing a foundation for accumulating analytical assets.
O3	**AI-Guided Knowledge Discovery**	Evaluate the LLM-based recommender's accuracy using synthetic queries against a seeded Knowledge Base, demonstrating its ability to retrieve relevant pipelines with a resonable precision.	Validates that the multi-agent LLM architecture can effectively match user intent to appropriate analytical assets, providing the technical foundation for intelligent assistance in pipeline discovery.

The subsequent sections will detail experiments specifically designed to evaluate SLEGO's

performance against these objectives.

6.1.2 Experimental Setup and Datasets Selection

To rigorously evaluate the SLEGO platform, a series of experiments were conducted using publicly available datasets and well-defined analytical tasks. Kaggle competitions and associated machine learning tasks offer a valuable framework for assessing a system like SLEGO due to:

- **Representative Complexity:** Kaggle datasets often encapsulate real-world complexities (missing values, diverse data types) demanding for both novices and experts.
- **Broad Coverage of Data Analytics Tasks:** They cover a wide spectrum of the analytics lifecycle, from preprocessing to model evaluation and visualisation.
- **Open-Access Data and Metrics:** Publicly available datasets with clear evaluation metrics (e.g., AUC, RMSE) ensure transparent and reproducible experimental setups.
- **Community Benchmarks:** Public leaderboards allow for objective comparison of SLEGO-constructed pipeline outcomes.

The experiments utilize a variety of datasets and machine learning tasks, primarily sourced from Kaggle competitions, as detailed in Table 6.2 and Table 6.3.

Table 6.2: Kaggle Tasks Utilised in SLEGO Evaluation.

Task ID	Kaggle Competition Name	Primary ML Task	Official Metric	Relevant Experiment(s)
T1	House Prices: Advanced Regression Techniques [65]	Regression	RMSLE	Exp. 1, 3
T2	Regression with an Abalone Dataset [75]	Regression	RMSLE	Exp. 2, 3
T3	OpenVaccine: COVID-19 mRNA Degradation[19]	Multi-output Reg.	MCRMSE	Exp. 3
T4	Home Credit Default Risk [66]	Binary Class.	ROC AUC	Exp. 3
T5	M5 Forecasting – Accuracy [33]	Time-series Forecasting	RMSSE	Exp. 3
T6	Plant Pathology 2020 – FGVC7 [36]	Image Class.	Mean col-wise ROC AUC	Exp. 3

Table 6.3: Overview of Dataset Used in SLEGO Experimental Studies.

Data ID	Dataset Name	Domain	Input Format	Key Characteristics	Used In
D1	House Prices	Real-estate	CSV	1 460 rows; 79 features	Exp. 1, 3
D2	Abalone Age	Biology	CSV	∼90 k rows; 9 features	Exp. 2, 3
D3	OpenVaccine	Biotech	JSON → CSV	RNA sequences + structure	Exp. 3
D4	Home Credit Default Risk	Finance	Multi-CSV	300 k+ rows; 122 attrs; imbalanced	Exp. 3
D5	M5 Forecasting	Retail	Multi-CSV	3 049 products × 2 000 days	Exp. 3
D6	Plant Pathology	Agriculture	Images + CSV	1 821 images; 4 classes	Exp. 3

These six datasets (T1–T6) are established community benchmarks, ensuring that experimental results are reproducible and comparable to published baselines. **House Prices (T1) [65]** is a canonical tabular regression task. **Abalone Age (T2) [75]** extends the regression template to biological data with significant skew. **OpenVaccine (T3) [19]** involves multi-output regression on RNA sequence data. **Home Credit Default Risk (T4) [66]** is a standard benchmark for binary classification with imbalanced data. **M5 Forecasting (T5) [33]** is a complex, large-scale time-series forecasting challenge. **Plant**

Pathology 2020 (T6) [**36**] is a fine-grained image classification task requiring deep learning methods.

This curated collection provides a rigorous proving ground for SLEGO. The selection was deliberately structured to test the platform's core design objectives (O1-O3) across three key dimensions:

- **Diversity of Domains and Modalities:** The datasets span six distinct fields (real estate, biology, finance, etc.) and four data modalities (tabular, sequential, time-series, and image). This variety is crucial for validating SLEGO's domain-agnostic, modular architecture (O2) and its ability to handle diverse data types.
- **Progressive Complexity:** The tasks follow a complexity gradient. T1 (House Prices) offers an accessible entry point for testing the foundational collaborative workflow (O1) in Experiment 1. T2 (Abalone) provides a basis for evaluating microservice reusability (O2) in Experiment 2. The remaining tasks (T3–T6) introduce advanced challenges such as multi-output regression, severe class imbalance, and deep learning, validating SLEGO's capacity for supporting sophisticated analytics.
- **Targeted Objective-Testing:** Each dataset is chosen to stress-test specific capabilities. For instance, the familiarity of T1 is ideal for demonstrating the end-to-end workflow (O1). The domain shift from T1 to T2 directly assesses component modularity, extendability and reusability (O2). The breadth of all six tasks provides a comprehensive testbed for the AI-powered recommender's ability to match user intent to suitable pipelines (O3).

This strategic selection ensures that the experimental evaluation is both comprehensive and representative of the real-world challenges faced by collaborative data science teams.

6.1.3 Overview of Experiments

Three core experiments were conducted to systematically evaluate SLEGO's capabilities against its design objectives. Each experiment simulates a key phase in the lifecycle of an analytics ecosystem, from initial asset creation to mature, AI-assisted operation across multiple domains.

- **Experiment 1 (E1): Foundational Collaboration & Low-Code Pipeline Construction.** This experiment evaluates the initial collaboration cycle where a new analytical asset is created from scratch. It primarily tests **Objective O1 (End-to-End Collaboration)** by demonstrating the complete, traceable workflow involving Technical Experts, Domain Experts, and End Users.
- **Experiment 2 (E2): Pipeline Reuse, Extension, & Knowledge Evolution.** This experiment focuses on the platform's ability to adapt and evolve. It assesses how an existing pipeline can be reused for a new task and extended with technologically diverse microservices, thereby testing **Objective O2 (Modular & Extensible Architecture)**.
- **Experiment 3 (E3): Cross-Domain Versatility & AI-Driven Efficiency.** This experiment evaluates **Objective O3 (AI-Guided Knowledge Discovery)** after all platform components are in place. Adapting the evaluation methods of Starlinger *et al.* [79] and Gu *et al.* [29], we report three rank-based metrics that match our single-label setting:
  - *Hit@1* – proportion of queries where the correct pipeline is ranked first.
  - *Hit@3* – proportion of queries where the correct pipeline appears within the top 3.
  - *Mean Reciprocal Rank (MRR)* – average inverse rank of the first correct pipeline.

  Each query has exactly one ground-truth pipeline, so recall is identical to Hit@k and precision@k adds limited insight. Hit@1 and Hit@3 capture success at the most relevant cut-offs, while MRR summarises overall ranking quality.

A summary of these experiments, their focus, the objectives they evaluate, and their key outcomes is presented in Table 6.4.

Table 6.4: Summary of Experimental Studies and their Core Focus.

Exp. ID	Experiment Focus	Primary Objective Evaluated	Tasks & Datasets	Key Outcome Demonstrated
E1	**Foundational Collaboration & Low-Code Pipeline Construction**	O1: End-to-End Collaboration	T1, D1	Successful creation of a baseline analytical asset from scratch via a complete collaborative workflow. End Users effectively build and tune pipelines using the low-code interface.
E2	**Pipeline Reuse, Extension, & Knowledge Evolution**	O2: Modular & Extensible Architecture	T2, D2	Seamless reuse of an existing pipeline for a new task and its subsequent extension with diverse, newly developed microservices.
E3	**Cross-Domain Versatility & AI-Driven Pipelines Discovery**	O3: AI-Guided Knowledge Discovery	T1-T6, D1-D6	Effective application of the platform across multiple diverse domains. End Users efficiently discover pipelines using the accurate, AI-guided LLM recommender.

6.2 Experiment 1: Foundational Collaboration and Low-Code Pipeline Construction

6.2.1 Objective

This experiment provides a focused validation of SLEGO's core objective: End-to-End Collaboration Support (O1). It simulates the initial creation cycle of an analytical asset from scratch to demonstrate that the architecture supports a complete, traceable work-

flow that seamlessly integrates the distinct contributions of the Technical Expert (TEG), Domain Expert (DEG), and End User (EUG).

6.2.2 Scenario Setup

The experiment begins with an empty SLEGO Knowledge Base (KB). The analytical task is to predict house prices using the House Prices dataset (D1, T1). This scenario requires the Technical Expert to build and upload the necessary microservices (as described in Section 4.5.1), the Domain Expert to provide initial domain context by annotating these components (Section 4.5.2), and the End User to construct and execute a baseline pipeline using the low-code interface (Section 4.5.3).

6.2.3 Method

This experiment simulates a typical collaborative workflow, using the example illustrated in Figure 4.11. The process involved the following steps:

1. **Problem Definition (EUG, Step 1):** An EUG identifies the need to predict house prices for a new project (T1).
2. **Initial KB Search Fails (EUG, Steps 2 & 3-NO):** The EUG queries the empty KB via the KRU/RQU interfaces, finds no relevant assets, and escalates the requirement to the TEG.
3. **Component Creation (TEG, Steps 9, 10, 11-NO, 13-15):** The TEG, finding no reusable base, develops a suite of seven microservices for a standard regression workflow, adhering to SLEGO's G1-G6 guidelines. The microservices created were:
    - A column remover (‘remove_columns_from_csv‘)
    - A preprocessor for missing values (‘fill_missing_values_in_csv‘)
    - An encoder for categorical features (‘encode_categorical_columns_in_csv‘)

- A data splitter (‘random_split_csv‘)
- A model trainer (‘train_random_forest_regressor‘)
- A data transformer using saved objects (‘transform_data_with_saved_preprocessors‘)
- A prediction generator (‘predict_with_sklearn_model‘)

After validation, these microservices and a baseline pipeline template, `HOUSE_PIPELINE`, are published to the KB (see Appendix C.1.1 for the JSON configuration). This establishes the initial set of modular components.

4. **Knowledge Curation (DEG, Step 16):** The DEG reviews the new assets via the KMU, adding domain-specific annotations such as notes on real estate data, recommended parameter defaults, and usage examples (see Appendix C.1.3). This curation enriches the KB for future users.

5. **Pipeline Construction and Iterative Refinement (EUG, Steps 2, 3-YES, 3a, 4-6):** Guided by the DEG's annotations, the EUG discovers the assets, uploads input data, and uses the low-code PAU to assemble the `HOUSE_PIPELINE`. This process includes an **iterative refinement cycle**: the EUG adjusts parameters (such as the imputation strategy), executes partial workflows to inspect intermediate artifacts (like `train_processed.csv`), and then runs the complete pipeline.

6. **Collaborative Loop Completion (EUG, Steps 7 & 8):** The EUG saves their validated pipeline configuration back to the KB, provides feedback on the components (Step 7), and completes their analytical task (Step 8).

6.2.4 Results and Findings

The experiment successfully demonstrated a typical end-to-end collaborative workflow, validating Objective **(O1)**. The key findings were:

- **Successful Orchestration of a Complex Workflow:** The architecture successfully orchestrated the contributions of all three user roles. This process involved the execution of the 7 microservices, which in turn generated 12 data and model artefacts to produce the final result. This confirms the platform's ability to manage the multi-stakeholder, multi-component interactions central to **O1**.
- **Externally-Validated Outcome:** The final outcome (submission.csv file) generated by the EUG constructed pipeline was submitted to the Kaggle competition. It achieved a score of 0.151, placing it in the 59th percentile ($\approx$ top 41 %) of over 4,800 competitors. This result provides an objective benchmark, confirming that the workflow delivers a meaningful output.
- **Creation of a Reusable Asset for Low-Code Users:** The experiment resulted in a new, fully annotated pipeline template (`HOUSE_PIPELINE`) in the Knowledge Base. This asset was specifically designed to be reused by End Users on the low-code platform. It bundles the seven microservices and the expert knowledge into a single package that can be easily found, loaded, and modified through the graphical interface for future projects.

6.2.5 Discussion

This experiment provides initial validation for the **End-to-End Collaboration (O1)** framework. By demonstrating a single, typical workflow from start to finish, it serves as a proof-of-concept that the architecture can support a traceable, multi-stage process connecting the TEG, DEG, and EUG. The objective result from the Kaggle competition confirms that this structured collaboration can produce a meaningful output.

A key aspect of this workflow is how end-users participate directly through the low-code interface. The EUG was not just a passive recipient; they actively assembled the pipeline, adjusted parameters, and ran parts of the workflow to check intermediate results. This design allows users without coding skills to test different analytical ideas (e.g., "What if we

change the imputation strategy?"), which is intended to speed up the process of generating insights.

The experiment shows that for the workflow to succeed, the different user roles must work together in a codependent system. The tools and knowledge provided by the experts are only useful when the end-user can find and apply them to solve their problem. This mutually reinforcing relationship is a central principle of the architecture. However, this also points to a key requirement: the process works best when all participants have a shared understanding of the task, a condition typically found within a single organisation or team. The framework provides the tools for collaboration, but it relies on the team's existing alignment.

In conclusion, this experiment shows that SLEGO's architecture offers a viable pattern for organising the creation of an analytical asset. It validates the foundational support for **O1** by demonstrating a full cycle of component creation, knowledge curation, and end-user application. The workflow's functionality was confirmed within this controlled scenario, while its performance under conditions of less team alignment or greater problem ambiguity remains a challenge for any collaborative system.

6.3 Experiment 2: Pipeline Reuse, Extension, and Knowledge Evolution

6.3.1 Objective

This experiment evaluates SLEGO's **Modular & Extensible Architecture (O2)** by assessing its adherence to the six theoretical design guidelines (G1-G6) established in Section 4.2.3. Specifically, the experiment demonstrates **modularity** through the reuse of an existing analytical asset for a new problem, and **extensibility** through the integration of new, technologically diverse microservices. The role of the integrated Knowledge Base as a key enabler for these architectural properties is also examined.

6.3.2 Scenario Setup

The experiment begins with the SLEGO Knowledge Base (KB) populated with the assets from Experiment 1: the `HOUSE_PIPELINE` template and its seven constituent microservices. A new analytical task is presented to an End User Group (EUG): predict the age of abalone (Task T2) using a new dataset (D2). The scenario presumes that an initial analysis leads the EUG to require features not present in the baseline pipeline: (1) a dynamic, interactive data visualisation, (2) an automated, LLM-powered explanation of that visualisation, and (3) an AutoML component. These advanced microservices are assumed not to exist yet in the KB.

6.3.3 Method

The experiment simulated a continuous workflow for the Abalone age prediction task (T2), following the collaborative steps from Figure 4.11:

1. **Initial Pipeline Reuse (EUG, Steps 1-5):** The EUG defined their objective: predict abalone age (**Step 1**). They searched the KB and, guided by its metadata, identified the existing `HOUSE_PIPELINE` as a suitable template for a generic regression task (**Steps 2, 3-YES**). Using the low-code PAU, they adapted this pipeline by simply changing the input data path to the Abalone dataset (D2) and updating the target column name parameter to "Rings" (**Steps 4, 5**). This created a new baseline, `ABALONE_PIPELINE_BASE`, demonstrating the immediate reusability of existing analytical patterns defined as a formal DAG.

2. **Identifying Need for Advanced Features (EUG, Steps 6, 9):** While the baseline pipeline worked, its outputs were basic. The EUG reviewed the static plots and results (**Step 6**) and identified the need for the three advanced capabilities. A search confirmed these did not exist in the KB, leading to an escalation to the TEG (**Step 9**).

3. **Extending the Platform (TEG, Steps 10-15):** The TEG received the request and confirmed that most microservices from `HOUSE_PIPELINE` could be reused (**Steps 10, 11-YES**). They then proceeded to develop three new, technologically diverse microservices (**Step 12**):

   - An **interactive plotting microservice** (‘plot_column_distribution_in_csv_-js‘) wrapping a JavaScript library, showcasing integration with different programming languages.
   - An **LLM-explanation microservice** (‘explain_visualization_with_chatgpt‘) using an external OpenAI API, showcasing integration with external web services.
   - An **AutoML microservice** (‘train_pycaret_automl_regressor‘) encapsulating the complex PyCaret library, demonstrating how sophisticated third-party tools can be turned into simple, reusable nodes.

   These were validated and published to the KB, demonstrating the platform's extensibility (**Steps 14, 15**).

4. **Knowledge Curation (DEG, Step 16):** The DEG reviewed the new, advanced microservices. They added crucial usage guidance to the KB, such as API key requirements for the LLM service and notes on interpreting AutoML outputs, ensuring the new tools were well-documented and safe for broader use.

5. **Integration and Final Analysis (EUG, Steps 2-8):** Empowered by the new, curated tools, the EUG created an enhanced pipeline, `ABALONE_PIPELINE_ADVANCED` (see Appendix C.2.1 for JSON and Appendix C.2.2 for the DAG). This new workflow included parallel modelling paths, comparing the original Random Forest with the new AutoML component. After executing this advanced pipeline and reviewing its comprehensive outputs—including interactive plots, textual explanations, and competing model results—the EUG saved the final configuration to the KB as a new template (**Steps 7, 8**).

6.3.4 Results and Findings

For this experiment, a functional implementation of the workflow was generated. The outcomes are quantified across three levels of reuse, which collectively provide evidence for the platform's **Modular & Extensible Architecture (O2)**:

- **Pipeline-Level Reuse:** The highest level of reuse was observed when the EUG successfully adapted the entire `HOUSE_PIPELINE` template to create the functional adaptation, requiring only parameter-level changes, validated the reusability of a complete analytical pattern. The resulting pipeline achieved a competitive RMSLE score of 0.1542 (57.2th percentile) on the Abalone task.
- **Microservice-Level Reuse:** In constructing the final `ABALONE_PIPELINE_ADVANCED`, all 7 of the original microservices from Experiment 1 were reused. These were integrated alongside the 4 newly created microservices. Notably, the success of the new AutoML branch was directly dependent on the outputs of 5 of these reused microservices (from data cleaning to the train-validation split), demonstrating how new capabilities can be built upon an established, trusted foundation.
- **Data-Level Reuse and Performance Improvement:** The pipeline's DAG (see AppendixC.2.2) structure facilitated the reuse of intermediate data artefacts. The `train_split.csv` artefact, for instance, was consumed by two parallel modeling branches. The final extended pipeline managed **18 distinct artefacts** and yielded a performance improvement, with the AutoML branch achieving an RMSLE score of 0.1490 (60th percentile).

6.3.5 Discussion

This experiment validates SLEGO's core design by illustrating how its architectural principles facilitate the reuse and extension of analytical assets, thereby demonstrating its Modular & Extensible Architecture (O2). The platform's success is enabled by the synergistic interplay between its six design guidelines (G1-G6) and the central Knowledge

Base. Table 6.5 details how these guidelines enabled the specific workflow activities in this experiment.

The experiment first demonstrated modularity through the successful cross-domain reuse of the `HOUSE_PIPELINE`. This is technically founded on G1 (Single Responsibility) and G4 (Statelessness), which ensure each microservice is a self-contained, predictable unit. Because these components are chained within a Directed Acyclic Graph (DAG), the entire analytical pattern could be repurposed by simply redirecting the initial input file, provided its schema was compatible as enforced by G2 (Explicit Data Contracts). However, this technical feasibility became practical only because the user could identify the shared analytical context of a regression task via G6 (Clear Documentation) in the Knowledge Base. This confirms that modularity in SLEGO is not merely a technical property but a socio-technical outcome, dependent on both well-engineered components and the user's ability to discover and apply them.

This workflow also highlights an architectural limitation. The smooth reuse of the pipeline was possible because the data schemas were compatible. A microservice's applicability is fundamentally constrained by its G2 (Explicit Data Contract). While this ensures pipeline reliability, it limits reuse to structurally similar data. The primary architectural challenge is thus to design microservices with sufficient generality for wide application. A potential solution, not explored in this work, would be to establish a canonical data model within SLEGO. Microservices could then be designed to operate on this standardised internal model, with dedicated adapter microservices at the beginning of pipelines responsible for transforming various raw input formats into the canonical representation.

In summary, this experiment shows the synergy between the design guidelines and the Knowledge Base in achieving the O2 objective. The technical principles (G1-G5) define the rules for a modular and extensible system, while the documentation principle (G6) enables users to apply those rules effectively. The creation and capture of the `ABALONE_PIPELINE` illustrate this cycle, where the architecture allows for the creation of new assets that compound the platform's utility over time.

Table 6.5: Examples Illustrating the Adherence to Design Guidelines (G1-G6) in the Workflow of Experiment 2.

Guideline	Concrete Example from Experiment 2
G1: Single Responsibility	The `random_split_csv` microservice performs only one function: partitioning a dataset. Its singular focus, separate from cleaning or training, allowed it to be reused directly from the pipeline and to serve as a common input for two distinct, parallel modeling branches (`train_random_forest_regressor` and the new `train_pycaret_automl_regressor`) in the advanced abalone pipeline.
G2: Explicit Data Contract	The new `train_pycaret_automl_regressor` microservice required inputs named `train_file` and `valid_file`. It could be safely integrated because the pre-existing `random_split_csv` microservice produced outputs with precisely these names. This strict I/O contract, enforced by the platform, guaranteed that the new component could be "plugged in" correctly.
G3: Pure Parameterization	The `train_random_forest_regressor` microservice's behavior is controlled exclusively through parameters. The EUG could alter the model's complexity by setting hyperparameters like `n_estimators:300` and `max_depth:10` directly in the low-code UI, separating the generic logic (code) from the specific configuration (parameters) and empowering user-led experimentation.
G4: Stateless & Retry-Safe	The `random_split_csv` microservice produces a persistent artifact, `train_split.csv`. Because the service is stateless, this output is deterministic and reliable. This allowed the EUG to reuse this single artifact as the input for two parallel modeling branches. This also enables efficient re-runs, as the system can skip re-creating this artifact if only downstream nodes are changed.
G5: Acyclic Data-Flow	In the `ABALONE_PIPELINE_ADVANCED`, data flows from the `random_split_csv` node to the two parallel modelling nodes. The Directed Acyclic Graph (DAG) structure enforces this one-way flow, preventing circular dependencies that would cause a logical deadlock and ensuring the pipeline is both executable and visually comprehensible.
G6: Clear Documentation	The EUG, facing a new problem, discovered and repurposed the `HOUSE_PIPELINE1` template from a different domain (real estate). This was possible only because its metadata in the Knowledge Base clearly documented its purpose as a "generic regression pipeline," making its abstract, cross-domain utility discoverable.

6.4 Experiment 3: Cross-Domain Versatility and AI-Driven Operational Efficiency

6.4.1 Objective

This experiment evaluates SLEGO's capacity for **AI-Guided Knowledge Discovery (O3)**. It assesses the LLM-recommender's effectiveness in guiding users to suitable pipelines across a diverse set of analytical tasks. A key aspect of this evaluation is demonstrating how the AI guidance relies upon and, in turn, makes manageable the platform's **organisational versatility**—its ability to support varied domains and data modalities within a single, unified architecture.

6.4.2 Scenario Setup

The SLEGO Knowledge Base (KB) is now populated with a collection of assets from Experiments 1 and 2, including various general-purpose microservices and more specialised pipeline templates like `HOUSE_PIPELINE` (for basic regression) and `ABALONE_PIPELINE` (a regression pipeline enhanced with AutoML and LLM explanations).

This experiment introduces four new, distinct analytical challenges for End User Groups (EUGs):

- **OpenVaccine (D3, T3):** A bioinformatics task involving sequence data (JSON converted to CSV) for multi-output regression to predict RNA degradation rates.
- **Home Credit Default Risk (D4, T4):** A financial task using a large tabular dataset (multi-CSV) for binary classification to predict loan default risk.
- **M5 Forecasting (D5, T5):** A retail task involving extensive time-series data (multi-CSV) to forecast product sales.

- **Plant Pathology (D6, T6):** An agricultural task using image data (images + CSV metadata) for multi-class image classification to identify plant diseases.

To quantitatively assess the AI-driven guidance, an offline evaluation of the LLM recommender was conducted. This involved using a test suite of 60 synthetic query prompts (10 distinct prompts for each of the six primary tasks, T1-T6, addressed across Experiments 1, 2, and 3). The Knowledge Base for this evaluation was seeded with the six corresponding "ground-truth" real pipelines and augmented with 100 additional synthetic "distractor" pipelines to simulate a more complex, mature repository. Hit Rate and Mean Reciprocal Rank (MRR) metrics were then used to measure the recommender's ability to rank the correct ground-truth pipeline highly in response to these queries.

6.4.3 Method

The experiment followed a two-pronged approach for each of the four new, diverse analytical tasks, mapping to the collaborative workflow steps (Figure 4.11):

1. **Expert-Led Component Expansion (as needed, TEG/DEG, Steps 9-16):** For each new domain, if the LLM recommender or initial EUG exploration revealed a lack of essential specialized microservices (e.g., a specific image data generator for T6, or a tailored sequence encoding method for T3) (**Step 9** for TEG), the following occurred:
    - The Technical Expert Group (TEG) designed and implemented these new specialized microservices. Crucially, they aimed to reuse existing general-purpose components from the KB (e.g., data splitters, standard model evaluators, submission file creators) wherever possible, showcasing the platform's inherent modularity (**Steps 10, 11-YES, 12-15**).
    - The Domain Expert Group (DEG) then curated these newly added components, enriching them with domain-specific metadata, usage guidelines, and parameter recommendations relevant to the new task area (e.g., typical image sizes

for plant pathology, relevant sequence features for RNA analysis) (**Step 16**). This continuous enrichment of the Knowledge Base is vital for both platform versatility and the effectiveness of AI-driven guidance.

2. **AI-Guided User-Driven Pipeline Assembly (EUG, Steps 1-8):** For each of the four diverse tasks (T3-T6):

   - An EUG defined their specific analytical objective for the given task (**Step 1**).
   - The EUG then utilized the LLM-recommender, accessed via the Recommendation Query UI (RQU), to find or construct suitable pipelines by providing a natural language description of their goal (**Step 2**). The recommender leveraged the now broader and deeper KB, which includes both general-purpose and newly added specialized components.
   - If the recommender proposed a suitable pipeline template or a coherent set of microservices (**Step 3-YES**), the EUG proceeded. They uploaded the relevant dataset (D3, D4, D5, or D6) using the Data Management UI (DMU) (**Step 4**).
   - Using the low-code Pipeline Assembly UI (PAU), they assembled and configured the pipeline, potentially adapting the recommender's suggestion or building upon a retrieved template. They adjusted parameters specific to the dataset and task (**Step 5**).
   - The pipeline was then executed, and the EUG reviewed the results, iteratively refining the configuration if necessary (**Step 6**).
   - Upon achieving satisfactory results, the EUG saved their completed and validated pipeline configuration back to the KB, making it a new, domain-specific asset for potential future reuse (**Step 7**), and concluded their task (**Step 8**).

The JSON configurations for the final pipelines constructed for all four tasks (OpenVaccine, Home Credit, M5 Forecasting, Plant Pathology) with their corresponding Directed Acyclic Graphs (DAGs) are provided in the GitHub Repository.

6.4.4 Results and Findings

- **Pipeline Construction Across Diverse Domains:** For all four new analytical tasks, the EUGs constructed functional pipelines based on the AI recommender's output. The summary of all six pipelines developed across the experiments (see Table 6.6) further illustrates this versatility. The resulting pipelines achieved the performance metrics detailed in Table 6.7 on their respective Kaggle leaderboards.
- **LLM Recommender Performance:** An offline study evaluated ranking quality on 60 natural-language queries spanning the six task domains examined in earlier case studies. Each query was evaluated against a knowledge base that included the ground-truth pipeline and 100 distractors. The recommender attained *Hit@1* = 0.75, *Hit@3* = 0.92, and *MRR* = 0.82 (Table 6.8), meaning that the correct pipeline is ranked first for 75% of queries and appears within the top three for 92%. These results indicate that the multi-agent, RAG-based recommender meets Objective O3 by translating user intent into suitable analytical workflows with a high top-3 success rate.

Table 6.6: Summary of SLEGO pipelines developed across all experiments.

Pipeline	**Domain**	**dataset**	**Task**	**Model Family**	**Key Preprocessing**
House Prices (T1)	Real-estate	CSV	Regression	Random Forest	One-hot, Impute
Abalone Age (T2)	Biology	CSV	Regression	LightGBM	Scale, Log-transform
OpenVaccine (T3)	Biotech	JSON→CSV	Multi-output Regression	XGBoost	Tokenize, Reshape
Home Credit (T4)	Finance	Large CSV	Binary Classification	LightGBM	Impute, One-hot
M5 Forecasting (T5)	Retail	Multi-CSV	Time-series Forecasting	LSTM	Lag features, Merge
Plant Pathology (T6)	Agriculture	Images+CSV	Image Classification	ResNet-50	Augment, Relabel

Table 6.7: Public-Leaderboard Checkpoints for All Tasks Across Experiments 1, 2, and 3. (Evaluated May 2025)

Competition (Task ID)	**Score**	**Teams**	**Rank**	**Percentile**
House Prices (T1)	0.151	4,835	2855	59.0%
Abalone Age (AutoML)(T2)	0.152	2606	1490	57.2%
OpenVaccine (T3)	0.423	1,636	397	75.7%
Home Credit (T4)	0.692	7,180	6,246	13.0%
M5 Forecasting (T5)	5.446	5,558	2,969	46.6%
Plant Pathology (T6)	0.531	1,317	1,246	5.4%

Table 6.8: Offline Evaluation Metrics for LLM-Based Recommender (60 queries; KB with 6 real + 100 distractor pipelines).

Metric	**Value**
Hit Rate@1	0.7500
Hit Rate@3	0.9167
Mean Reciprocal Rank (MRR)	0.8222

6.4.5 Discussion

The findings from this experiment provide significant evidence for the effectiveness of SLEGO's architecture in enabling AI-Guided Knowledge Discovery (O3), particularly within a mature and diverse analytics ecosystem. This reliable performance results (Table 6.8) from recommender design features (see Section4.4), rather than from the large language model alone.

However, it's important to consider the limitations of the evaluation when interpreting these results. The offline test utilized well-structured, synthetic queries, which may not fully capture the ambiguity or imprecision inherent in real-world user requests. Moreover, the recommender's effectiveness is intrinsically linked to the quality and completeness of the metadata curated in the Knowledge Base. An undocumented or poorly described component ecosystem would compromise its accuracy. Additionally, while SLEGO offers a prompt template to guide users, it assumes that users can articulate their analytical goals

effectively. In practice, users with limited analytical experience may struggle to translate a business problem into a well-defined query, underscoring a potential disconnect between the system's technical capabilities and its practical usability.

Despite these limitations, this experiment reveals an insight: the platform's versatility and its AI-guided discovery are interconnected. As the platform supports more analytical tasks, the Knowledge Base expands into a comprehensive repository of diverse assets. This rich repository serves as the fuel for the AI recommender, enabling it to provide relevant suggestions based on the available information. However, a large repository creates a new problem: it becomes overwhelming for users to navigate manually. At this point, AI guidance transitions from a convenience to a necessity. The AI becomes the critical tool that makes the platform's power manageable and its assets discoverable. This creates a cycle: a more versatile platform enables smarter AI guidance, which in turn drives user adoption and asset contribution, further enhancing the platform's value. In large organizations, this cycle is key to breaking down silos by helping teams find and build upon each other's work.

6.5 Overall Evaluation of SLEGO

In accordance with the Design Science Research Methodology (DSRM) guiding this thesis, this evaluation focuses not on production-level performance, but on the feasibility, coherence, and emergent properties of the proposed architecture. The central aim is to validate the architectural blueprint itself as a viable solution to the identified research gaps.

6.5.1 O1: End-to-End Collaboration Support

6.5.1.1 Architectural Validation

The experiments demonstrated that SLEGO's architecture can facilitate a traceable, end-to-end collaborative workflow. Experiment 1 provided the primary validation for this

objective, showing that different user roles (EUG, TEG, DEG) and their corresponding system interfaces could orchestrate the creation of an analytical pipeline. A key aspect of this collaborative support is the empowerment of non-technical users through the low-code platform, which enables their direct participation in the analytical process. The Knowledge Base acts as the central, shared repository connecting the contributions of all roles.

6.5.1.2 Scope and Limitations

The primary limitation regarding this objective is the evaluation's reliance on simulated user roles and workflows. It confirms the architecture's capacity to structure these interactions but does not measure real-world team dynamics or the nuances of human collaboration. Furthermore, the development costs associated with the Technical Expert adhering to the platform's design guidelines (G1-G6) were not formally quantified.

6.5.2 O2: Modular & Extensible Architecture

6.5.2.1 Architectural Validation

The evaluation provided evidence for the effectiveness of SLEGO's Modular & Extensible Architecture (O2). Experiment 2 demonstrated both modularity through the reuse of a pipeline in a new domain, and extensibility through the integration of technologically diverse microservices. The architectural underpinnings for this is the SLEGO's design guidelines (G1-G6). Additionally, The platform's ability to support tasks across varied domains in Experiment 3 further demonstrated the organisational versatility that emerges from this modular design. This versatility also enables the growth of a governed and cumulative Knowledge Base.

6.5.2.2 Scope and Limitations

This evaluation successfully validated the architectural patterns for modularity. However, it did not include production-level technical scalability benchmarking. Critical aspects such as performance under high concurrency, behaviour with datasets significantly larger than those from Kaggle, and the impact of network latency were not assessed. Similarly, the growth of the Knowledge Base was simulated rather than observed organically, and key enterprise features like comprehensive security and operational maturity were outside the scope of this architectural prototype.

6.5.3 O3: AI-Guided Knowledge Discovery

6.5.3.1 Architectural Validation

Experiment 3 validated the AI-Guided Knowledge Discovery objective by showing that a well-structured knowledge repository can work in concert with an AI recommender. Offline evaluation metrics indicate that the proposed architecture, which combines a structured knowledge base with a multi-agent RAG-based recommender, reliably translates user intent into viable analytical pipelines.

6.5.3.2 Limitations of the Evaluation

The recommender was tested offline with tidy Kaggle datasets, clear synthetic queries, and simulated analyst roles. This setup helps isolate SLEGO's design choices but limits what we can claim about real practice. Day-to-day data are often messy or proprietary, user questions are vague, and company rules can block certain pipelines. In such settings the hit-rate and MRR reported here will probably fall, and tools for cleaning schemas or fixing prompts will be required.

The study also skips long-term and cost concerns. Recommendation quality may drift as the knowledge base grows or the LLM is updated, and frequent LLM calls could become

expensive. In addition, the pipeline library we tested is small compared with what a full deployment would face. Future work should stress-test SLEGO on noisy corporate data, involve real analysts, and track accuracy, stability, and cost as the system scales.

6.5.4 Summary of Evaluation

In summary, these experiments has successfully demonstrated the viability and coherence of the SLEGO architecture against its three primary design objectives. The evaluation validates that the proposed design serves as a foundational blueprint for a new type of analytics platforms capable of fostering collaboration, promoting systematic reusability, and leveraging AI for knowledge discovery. The study also notes several limitations which, viewed through Design Science Research Methodology, point to clear directions for future research and engineering.

Chapter 7

Conclusion and Future Directions

This chapter concludes the thesis by integrating the research conducted to design, implement, and evaluate the SLEGO (Software-Lego) platform. The preceding chapters have presented the theoretical foundations, system architecture, and experimental validations. This chapter brings these elements together to provide a comprehensive overview of the research and its outcomes. It begins with a summary of the work, followed by an outline of the research contributions. It then examines the research's limitations to provide context for the findings and concludes by proposing a roadmap for future work.

7.1 Summary of the Thesis

This research addressed a significant challenge in modern data analytics: the difficulty of collaboration between users with different technical skills. The literature review in **Chapter 2** showed that existing analytics tools often force a choice between complex, code-intensive platforms for experts and simpler, but less flexible, low-code tools for domain specialists. This separation can hinder knowledge sharing and create inefficiencies.

To address this issue, this research employed the **Design Science Research Methodology (DSRM)**, as detailed in **Chapter 3**. The goal was to design and evaluate a novel IT

artifact, the proposed SLEGO platform, intended to provide a more unified and effective environment for a diverse range of users.

Chapter 4 presented the core architectural blueprint of SLEGO, detailing both its conceptual foundations and its concrete system design. The chapter began by establishing the theoretical model for analytics pipelines, representing them as formal Directed Acyclic Graphs (DAGs) where each analytical step is an independent, stateless microservice governed by strict design guidelines (G1-G6).

Building on this foundation, the chapter then described the platform's three-layer architecture, consisting of a Storage Layer for the Knowledge Base and data artefacts, a Service Layer for core logic and execution, and a UI Layer for user interaction. Key components within this architecture were detailed, including the specific schema of the Knowledge Base designed to capture technical and domain-specific metadata. The design of the retrieval-augmented recommender system, which leverages this Knowledge Base to translate natural language queries into pipeline suggestions, was presented. Finally, the chapter defines the distinct user roles—Technical Expert, Domain Expert, and End-User—and outlines their collaborative workflows within the system.

Chapter 5 detailed the practical implementation of the SLEGO prototype, utilizing technologies such as Python, Panel dashboard library, SQLite, and LLM model API to build a platform to demonstrate the architectural design.

Chapter 6 shows that the validation of this architecture was conducted through a series of three experiments that simulated the lifecycle of a collaborative analytics ecosystem:

- **Experiment 1 (End-to-End Collaboration)** demonstrated the initial creation of an analytical asset from scratch. It validated the end-to-end workflow, from the EUG's request to the TEG's component creation and the DEG's curation, culminating in the EUG successfully building a pipeline using the low-code interface.
- **Experiment 2 (Modular & Extensible Architecture)** tested the platform's adaptability. It showed how an existing pipeline from Experiment 1 could be seam-

lessly reused for a new domain and extended with technologically diverse microservices (wrapping JavaScript, an external API, and an AutoML library), thereby validating the architecture's modularity and the cumulative nature of the KB.

- **Experiment 3 (AI-Guided Knowledge Discovery)** evaluated the mature platform's ability to handle a wide array of analytical tasks across bioinformatics, finance, and image analysis. It confirmed that the unified architecture prevents tool silos and quantitatively validated the high accuracy of the LLM recommender in guiding users to find the correct analytics pipeline.

Collectively, the findings confirm that the SLEGO architecture provides a viable and coherent blueprint for a new type of analytics platforms—one that systematically fosters collaboration, enhances reusability, and democratizes access to advanced analytics for all users.

7.2 Research Contributions

This research makes several contributions to the field of collaborative data analytics, with their primary novelty lying in their synergistic integration.

1. **A Context-Enriched Graph-Theoretic Model for Pipelines:** This research extends the common use of Directed Acyclic Graphs (DAGs) for pipeline orchestration. It introduces a **context-enriched graph model** where pipeline nodes are formally linked to rich metadata, context, and I/O contracts stored in a central repository. This approach is designed to transform the DAG from a simple scheduler into a more semantically-aware, verifiable, and reusable analytical asset.

2. **An Role-Based Framework for Collaboration:** Moving beyond abstract definitions, this work implements and validates a three-part user model (Technical Expert, Domain Expert, End User). This framework provides tailored interfaces and defined workflows for component creation, knowledge curation, and pipeline consumption,

demonstrating a method for making the collaborative process more traceable and structured.

3. **A LLM(AI)-based Recommender System for Pipeline Discovery:** The thesis details a recommender system with a LLM verification mechanism. In contrast to unconstrained code generation, its retrieval-augmented design is constrained by the factual contents of the system's Knowledge Base. The experiments show this approach can improve the reliability of suggestions for analytical pipelines.

4. **A Structured Knowledge Base:** This research defines and implements a structured, queryable Knowledge Base that serves as the system's central repository. It provides a formal schema for capturing both the technical specifications of microservices (I/O contracts, parameters) and the domain context of pipelines (task goals, data characteristics), enabling systematic validation and discovery.

5. **An Open-Source Artifact for Community-Driven Research:** A key contribution is the delivery of the SLEGO prototype as a tangible, **open-source research artifact**. By making the code publicly available, this thesis provides a foundation for other researchers to validate, extend, and build upon this work, fostering advancement in the field.

6. **An Integrated Architectural Blueprint for Collaborative Analytics:** The primary contribution is a novel architectural blueprint that holistically integrates a microservice ecosystem, a structured knowledge base, a role-based collaboration model, and a natural language recommender. In a field with many specialized tools, this thesis demonstrates—both in design and through a working prototype—how these disparate elements can be synergistically combined into a single, coherent framework to support the entire collaborative analytics lifecycle for diverse users.

7.3 Research Limitations

While this research validates SLEGO's architectural blueprint, a critical analysis of its limitations is important for future work. The limitations arise from the focused scope of a DSRM-based thesis, which prioritizes architectural feasibility over production-level hardening.

7.3.1 Evaluation Within a Simulated Context

A primary limitation of the research is that the validation was conducted within a simulated context. The evaluation's scope did not extend to a real-world enterprise setting with its inherent complexities and representative users.

First, the platform was tested using public Kaggle datasets and well-defined analytical tasks (T1-T6). This controlled approach, while necessary for isolating and testing the core architectural workflows, does not account for the challenges of a genuine operational environment. Such settings typically involve proprietary data with unique quality issues, ambiguous or evolving business requirements, and complex organizational dynamics (e.g., conflicting priorities, legacy systems) that were not modeled.

Second, and more significantly, the study relied on simulated user roles (EUG, TEG, DEG) rather than conducting task-based evaluations with actual professionals. Consequently, the research can validate the architectural feasibility but cannot make empirical claims about the platform's practical usability or the effectiveness of the human-computer interaction (HCI) for each user persona. The current findings lack the qualitative data that would emerge from observing real data scientists, domain experts, and business analysts interacting with the system to solve authentic problems. Such an empirical user study would be required to identify friction points in the collaborative workflow and assess the cognitive load associated with the platform's use, moving beyond architectural proof-of-concept to practical utility.

7.3.2 Architectural vs. Production-Level Scalability

The thesis demonstrates organizational versatility and architectural scalability, showing the platform can handle diverse problem types without requiring new tool silos. However, it did not include rigorous, production-level performance benchmarking. Critical aspects like system behavior under high concurrency (many users executing pipelines simultaneously), performance with massive datasets (far exceeding Kaggle's scale), network latency impacts, and database optimisation under heavy load were not tested. The prototype validates the pattern, not its performance limits under stress.

7.3.3 Constrained Evaluation of the LLM Recommender

The LLM recommender was evaluated offline using a curated set of 60 queries against a seeded KB. This demonstrated high accuracy in a controlled environment. However, this does not address robustness in an unconstrained production setting. Key un-tested challenges include: handling ambiguous, malformed, or wildly out-of-scope user queries; managing prompt drift as the KB and user-base evolve; and the real-time latency and operational costs of LLM API calls in a high-throughput, interactive system.

7.4 Future Directions

The limitations identified above naturally chart a course for future research, moving from architectural validation to empirical testing and production-level hardening.

- **Conduct Task-Based User Studies with Representative Users:** The current evaluation relied on simulated user roles. A crucial next step is to conduct qualitative user studies with real-world participants who embody the target personas: data scientists (TUG), domain experts (DUG), and business analysts (EUG). These studies would involve assigning participants realistic, collaborative analytical tasks

and observing their interactions with the platform. Employing methods such as think-aloud protocols, direct observation, and post-task interviews would provide rich, actionable data on the platform's practical usability, the effectiveness of the UI for each role, and any friction points in the collaborative workflow. This would move beyond architectural validation to directly assess the human-computer interaction and provide a data-driven basis for iterative design improvements.

- **Integration of Ontologies:** Investigate the practical integration of domain-specific ontologies with the Knowledge Base. This could enable features like automated validation of pipeline semantics against domain rules, more context-aware recommendations, and improved interoperability between microservices based on formal knowledge representation.

- **Establish a Mature Governance Framework for the Generative AI Era:** Perhaps the most significant future direction is to position SLEGO as a critical governance layer for the new paradigm of LLM-driven analytics. As purely prompt-driven analysis raises concerns about reliability and reproducibility, a hybrid architecture is essential. Future work should operationalize and test a model where:

  - **Generative AI serves as a creation assistant**, rapidly scaffolding microservices and pipeline drafts under expert supervision.
  - **SLEGO serves as the governance and validation engine**, providing the indispensable framework for human-in-the-loop verification, refinement, and execution against the verifiable contracts stored in its Knowledge Base.

The proposed architecture provides a clear path to harness the creative power of generative AI while ensuring that analytical processes remain reliable, auditable, and trustworthy. In conclusion, SLEGO offers a robust architectural blueprint for a new generation of data analytics platforms—systems designed not just for processing data, but for empowering the diverse teams of people who turn that data into insight.

7.5 Summary

In summary, this thesis makes the case that collaborative data analytics does not require sacrificing rigour, and rigorous analytics need not exclude non-experts. By treating low-code platform, microservices, knowledge base and large language (AI) models as complementary rather than competing ingredients, SLEGO establishes a middle ground where humans and machines can co-evolve their analytical capabilities. It is the author's hope that the principles outlined here will inform both scholarly discourse and industrial practice, ultimately helping organisations turn raw data into reliable and actionable insight, together.

Appendix A

Design and Architecture of the SLEGO System

A.1 Domain Knowledge Base for SLEGO Pipelines

The domain experts contributed the following knowledge for the House Price Prediction pipeline, which was integrated into SLEGO's Knowledge Base:

```
{
    "task_goal": "This pipeline is designed for automated data
   preprocessing, model training, evaluation, and prediction in the
    context of a real estate price prediction task using the Kaggle
    House Prices dataset. The pipeline aims to streamline and
   standardise the process of building a regression model that
   predicts house sale prices based on a variety of features,
   ensuring robust data handling, repeatability, and reliable
   results. The overall workflow adheres to best practices for
   tabular data science projects in machine learning, including
   exploratory data analysis, cleaning, feature engineering, model
   development, and prediction file generation for competition
   submission.",
```

```
3     "data_context": "The input dataset is the classic 'House Prices
      : Advanced Regression Techniques' dataset from Kaggle, provided
      as CSV files. The main training data (./dataspace/house/train.
      csv) contains multiple columns representing house features (
      numerical and categorical) and the target variable 'SalePrice' (
      continuous). Output datasets include: processed CSVs for
      training, validation, and test sets, imputer and encoder objects
       (stored as .pkl files for numerical and categorical
      preprocessing reproducibility), visualization plots (e.g.,
      distribution of SalePrice), final prediction files ready for
      Kaggle submission (.csv), as well as evaluation results (.json).
       Data modalities are tabular. The dataset size typically ranges
      from hundreds to thousands of rows, each with 80+ columns. All
      files are handled in standard CSV format for compatibility and
      ease of use.",
4     "problem_type": "regression",
5     "pipeline_process": "The process flow includes: (1)
      Visualization of SalePrice distribution for initial exploratory
      data analysis; (2) Removal of identifier columns not useful for
      modeling; (3) Imputation of missing values in both numeric (
      using mean) and categorical (using most frequent value) columns;
       (4) Encoding of categorical features for model compatibility;
      (5) Splitting of data into training and validation sets using a
      randomised split; (6) Training of a Random Forest Regressor on
      the training set and evaluating its performance on the
      validation set; (7) Applying the same preprocessing (imputation
      and encoding) to the test data using saved preprocessors; (8)
      Making predictions on the processed test set with the trained
      model; (9) Outputting submission-ready predictions.",
6     "pipeline_components": [
7         "m-kaggle-house.plot_column_distribution_in_csv_js",
8         "m-kaggle-house.remove_columns_from_csv",
9         "m-kaggle-house.fill_missing_values_in_csv",
10        "m-kaggle-house.encode_categorical_columns_in_csv",
```

```
        "m-kaggle-house.random_split_csv",
        "m-kaggle-house.train_random_forest_regressor",
        "m-kaggle-house.transform_data_with_saved_preprocessors",
        "m-kaggle-house.predict_with_sklearn_model"]
    "domain_keywords": "housing, real estate, regression, machine
    learning, tabular data, data preprocessing, feature engineering,
     random forest, Kaggle",
    "additional_info": "The pipeline is tailored for model
    reproducibility and reliability by saving all imputation and
    encoding strategies. All random processes, such as data
    splitting and model training, use fixed seeds for consistency.
    The Random Forest Regressor can be easily configured for
    hyperparameter tuning. Evaluation outputs are stored in JSON for
     easy integration with tracking tools. The pipeline structure
    supports straightforward extension or adaptation for other
    tabular regression tasks. Submission output is generated in the
    format required by Kaggle competitions, with all IDs and
    predicted SalePrices preserved."
}
```

A.2 Detailed Workflow and Component Mapping

This appendix provides supplementary details to the end-to-end collaboration workflow described in Section 4.5.4.

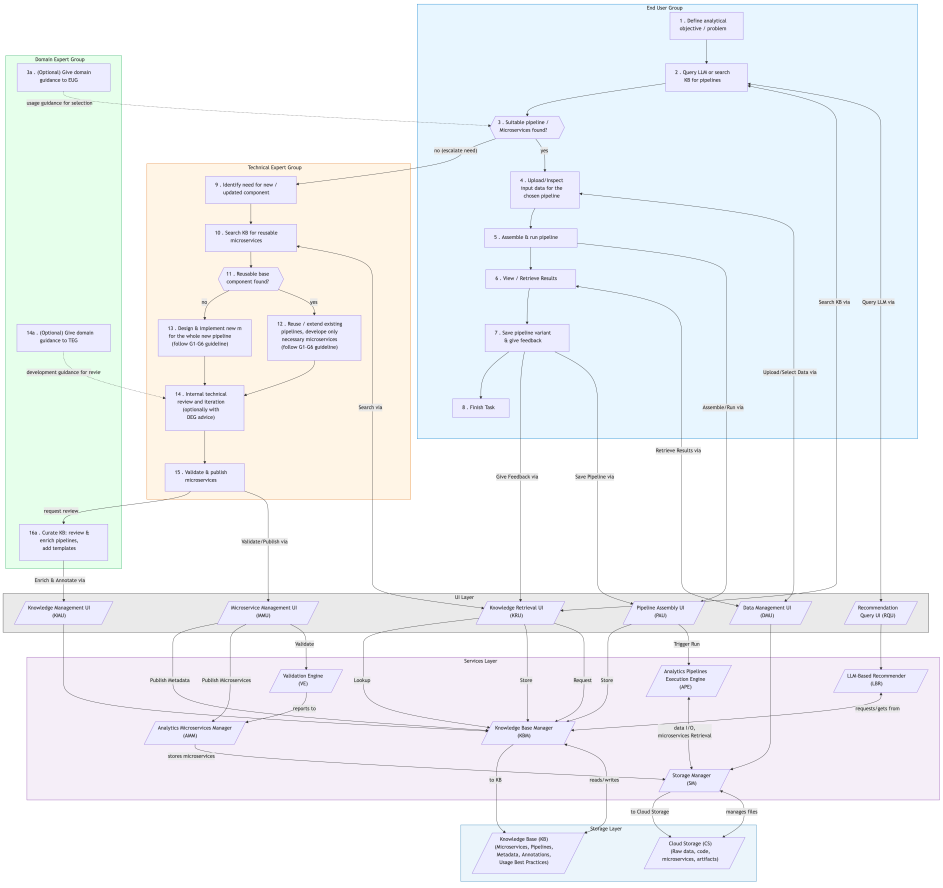

Figure A.1: Detailed end-to-end collaboration loop in SLEGO.

Table A.1: SLEGO Architecture Components and Notations.

Layer/Group	**Component Name**	**Notation**
User Groups		
	Community	COM
	Domain Expert Group	DEG
	Technical Expert Group	TEG
	End User Group	EUG
UI Layer		
	Knowledge Management UI	KMU
	Microservice Management UI	MMU
	Knowledge Retrieval UI	KRU
	Data Management UI	DMU
	Recommendation Query UI	RQU
	Pipeline Assembly UI	PAU
Services Layer		
	Validation Engine	VE
	LLM-Based Recommender	LBR
	Analytics Microservices Manager	AMM
	Knowledge Base Manager	KBM
	Storage Manager	SM
	Analytics Pipelines Execution Engine	APE
Storage Layer		
	Knowledge Base	KB
	Cloud Storage	CS

Appendix B

Prototype Implementation

B.1 Plot Return

```
{
    "m-yfinance.preprocess_filling_missing_values #1": {
        "inputs": {
            "input_csv": "dataspace/imported_market_data.csv"
        },
        "outputs": {
            "output_csv": "dataspace/preprocessed_data.csv"
        },
        "fill_strategy": "ffill"
    },
    "m-yfinance.compute_return #1": {
        "inputs": {
            "input_csv": "dataspace/preprocessed_data.csv"
        },
        "outputs": {
            "output_csv": "dataspace/returns_data.csv"
        },
        "window_size": 20,
        "target_column_name": "Close",
        "return_column_name": "Return",
```

```
        "keep_rows": false
    },
    "m-yfinance.plotly_chart #1": {
        "inputs": {
            "input_csv": "dataspace/returns_data.csv"
        },
        "outputs": {
            "output_html": "dataspace/returns_plot.html"
        },
        "index_col": 0,
        "x_column": "Datetime",
        "y_column": "Close",
        "title": "Data Plot",
        "legend_title": "Legend",
        "mode": "lines"
    }
}
```

This example pipeline:

1. `preprocess_filling_missing_values`: Uses forward fill to handle missing values.
2. `compute_return`: Calculates returns on the specified `Close` column.
3. `plotly_chart`: Generates an HTML line chart of the resulting data for easy visualization.

With these steps, SLEGO streamlines both the data preparation and visualization aspects of analytics, minimizing manual coding effort while maintaining transparency and reproducibility.

Appendix C

Experiments

C.1 Experiment 1

C.1.1 Experiment 1 - HOUSE_PIPELINE1

```
1
2 {
3
4     "remove_columns_from_csv #1": {
5         "inputs": {"csv_file": "./dataspace/house/train.csv"},
6         "outputs": {"csv_file": "./dataspace/house/train_dropped.csv"},
7         "columns_to_drop": ["Id"]
8     },
9     "fill_missing_values_in_csv #2": {
10        "inputs": {"csv_file": "./dataspace/house/train_dropped.csv"},
11        "outputs": {
12            "csv_file": "./dataspace/house/train_imputed.csv",
13            "num_imputer": "./dataspace/house/num_imputer.pkl",
14            "cat_imputer": "./dataspace/house/cat_imputer.pkl"
15        },
16        "label_column": "SalePrice"
17    },
18    "encode_categorical_columns_in_csv #3": {
```

```
        "inputs": {"csv_file": "./dataspace/house/train_imputed.csv"},
        "outputs": {
            "csv_file": "./dataspace/house/train_processed.csv",
            "encoder": "./dataspace/house/encoder.pkl"
        },
        "label_column": "SalePrice"
    },
    "random_split_csv #4": {
        "inputs": {"csv_file": "./dataspace/house/train_processed.csv"},
        "outputs": {
            "train_file": "./dataspace/house/train_split.csv",
            "valid_file": "./dataspace/house/valid_split.csv"
        },
        "test_ratio": 0.3
    },
    "train_random_forest_regressor #5": {
        "inputs": {
            "train_file": "./dataspace/house/train_split.csv",
            "valid_file": "./dataspace/house/valid_split.csv"
        },
        "outputs": {
            "model_file": "./dataspace/house/rf_model.pkl",
            "evaluation_file": "./dataspace/house/evaluation_rf.json"
        },
        "label_column": "SalePrice"
    },
    "transform_data_with_saved_preprocessors #6": {
        "inputs": {
            "csv_file": "./dataspace/house/test.csv",
            "num_imputer": "./dataspace/house/num_imputer.pkl",
            "cat_imputer": "./dataspace/house/cat_imputer.pkl",
            "encoder": "./dataspace/house/encoder.pkl"
        },
        "outputs": {"csv_file": "./dataspace/house/test_processed.csv"},
        "id_column": "Id"
    },
    "predict_with_sklearn_model #7": {
```

```
        "inputs": {
            "model_file": "./dataspace/house/rf_model.pkl",
            "csv_file": "./dataspace/house/test_processed.csv"
        },
        "outputs": {"csv_file": "./dataspace/house/submission.csv"},
        "id_column": "Id"
    }
}
```

C.1.2 Experiment 1 - Detailed DAG

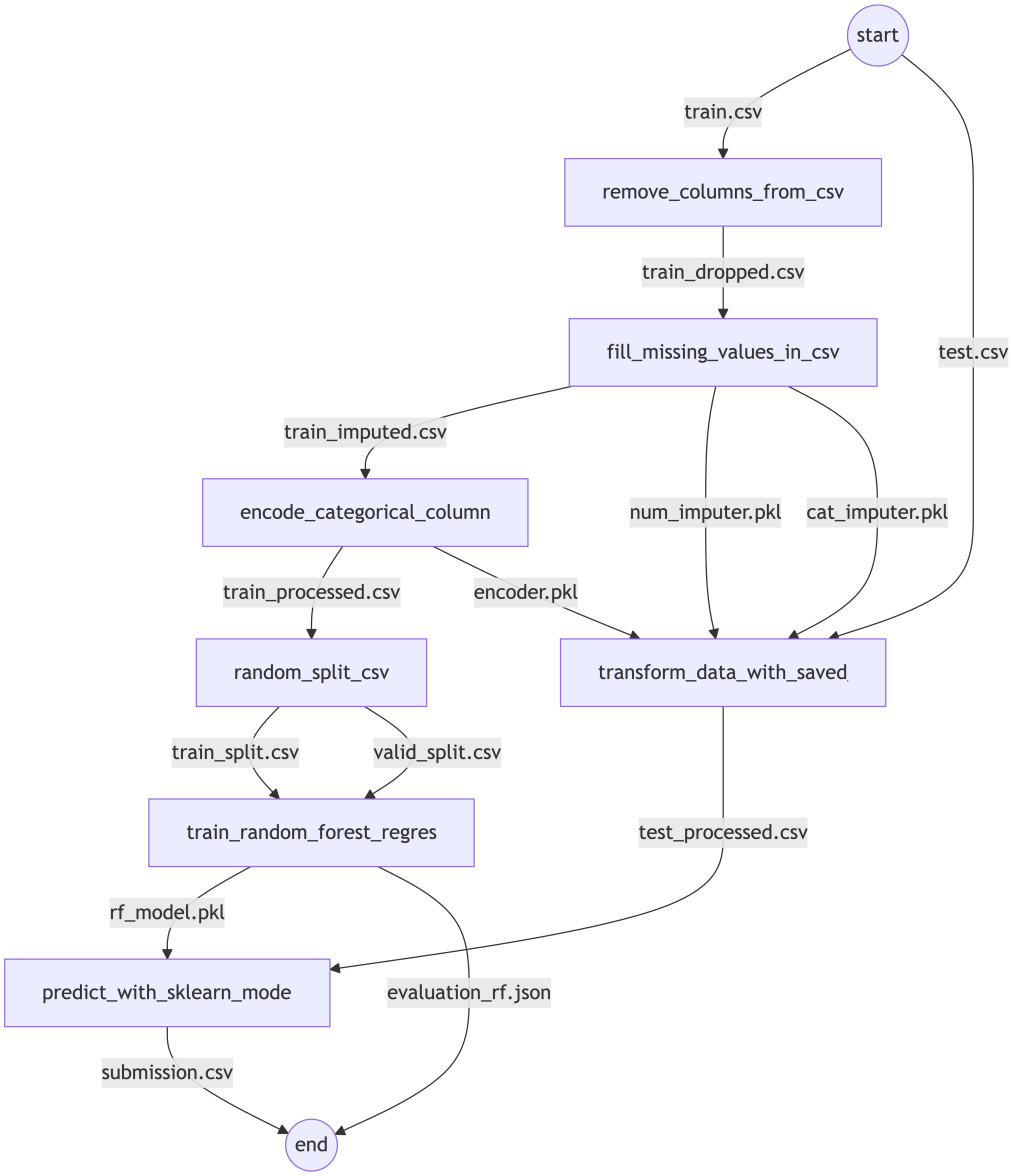

Figure C.1: Directed-acyclic graph of the *OpenVaccine* RNA-degradation pipeline.

C.1.3 Experiment 1 - Doman Expert's Annotation in Kowledge Base

```
1
2 {
3 "task_goal": "This pipeline is designed to prepare and analyse housing data
```

```
    to predict house sale prices using a machine learning model. It
   performs data cleaning, preprocessing, exploratory data analysis, model
    training, validation, and prediction, aimed at building a robust
   regression model to estimate house prices based on various features
   from the dataset.",
"data_context": "The input dataset is a CSV file named 'train.csv'
   containing housing data with multiple features including the target
   variable 'SalePrice'. The dataset includes both numeric and categorical
    columns. The data undergoes various transformations producing
   intermediate CSV outputs such as 'train_dropped.csv', 'train_imputed.
   csv', 'train_processed.csv', and final processed test data in '
   test_processed.csv'. The dataset size is typical for Kaggle housing
   challenges, stored in CSV format, and final outputs include plots,
   trained model files (pickle format), and prediction submissions as CSV.
   ",
"problem_type": "ETL and Regression",
"pipeline_process": "The pipeline initiates by visualizing the distribution
    of the 'SalePrice' column to understand its statistical properties. It
    then removes irrelevant columns (like 'Id'), imputes missing values in
    both numeric and categorical features using mean and most frequent
   strategies respectively, and encodes categorical variables for machine
   learning suitability. After preprocessing, the dataset is split into
   training and validation sets. A Random Forest regressor model is
   trained on the training set and validated. The saved imputers and
   encoders are then applied to preprocess test data, followed by
   prediction generation using the trained model, producing a final
   submission file.",
 "pipeline_components": {
 "plot_column_distribution_in_csv_js": {
 "role": "Generates a histogram plot of the target variable 'SalePrice' to
   visually inspect data distribution.",
 "parameters": {
 "column": "SalePrice",
 "bins": 50,
 "color": "blue",
 "alpha": 0.4
 }
```

```
 },
"remove_columns_from_csv": {
 "role": "Removes unnecessary columns such as 'Id' to avoid them
    interfering with modeling.",
 "parameters": {
 "columns_to_drop": "Id"
 }
 },
 "fill_missing_values_in_csv": {
 "role": "Fills in missing values with a numerical strategy (mean) for
    numeric columns and most frequent value for categorical columns,
    outputting imputers that save these strategies for reproducibility.",
 "parameters": {
 "label_column": "SalePrice",
 "numeric_strategy": "mean",
 "categorical_strategy": "most_frequent"
 }
 },
 "encode_categorical_columns_in_csv": {
"role": "Encodes categorical variables into numeric format using a saved
    encoder to prepare for model consumption.",
"parameters": {
"label_column": "SalePrice"
}
},
"random_split_csv": {
"role": "Splits the processed dataset into training and validation sets
    with a 70-30 ratio to enable model evaluation.",
"parameters": {
"test_ratio": 0.3,
"random_state": 42
}
},
"train_random_forest_regressor": {
"role": "Trains a Random Forest regression model on the training data and
    evaluates on validation data, storing the model and evaluation results.
    ",
```

```
"parameters": {
"label_column": "SalePrice",
"n_estimators": 50,
"max_depth": null,
"random_state": 42
}
},
"transform_data_with_saved_preprocessors": {
"role": "Applies previously saved imputers and encoders to the separate test dataset to ensure consistent preprocessing before prediction.",
"parameters": {
"id_column": "Id"
}
},
"predict_with_sklearn_model": {
"role": "Uses the trained Random Forest model to generate sale price predictions for the test dataset, producing a submission file.",
"parameters": {
"id_column": "Id",
"prediction_column": "SalePrice"
}
}
},
"domain_keywords": "real estate, housing market, regression, data preprocessing, machine learning, Random Forest, ETL",
"additional_info": "Key parameters impacting results include the number of estimators (50) in the Random Forest model, the chosen imputation strategies (mean for numerics and most frequent for categoricals), and the train-validation split ratio (0.7/0.3). The pipeline is focused on maintaining reproducibility with saved imputers and encoders, and includes data visualization for better understanding of data distribution."
}
```

C.2 Experiment 2

C.2.1 Experiment 2 - Abalone Pipeline

```
{

    "plot_column_distribution_in_csv_js": {
        "inputs": {
            "csv_file": "./dataspace/abalone/train.csv",
            "js_file": "./dataspace/abalone/plot_column_distribution.js"
        },
        "outputs": {"plot_file": "./dataspace/abalone/data_dist.png"},
        "column": "Rings",
        "bins": 100,
        "color": "blue",
        "alpha": 0.4
    },
        "explain_visualization_with_chatgpt": {
        "inputs": {"plot_file": "./dataspace/abalone/data_dist.png"},
        "outputs": {"explanation_file": "./dataspace/abalone/
    viz_explanation.txt"},
        "prompt": "Explain the key insights from the distribution plot of
    Rings."
    },
    "remove_columns_from_csv": {
        "inputs": {"csv_file": "./dataspace/abalone/train.csv"},
        "outputs": {"csv_file": "./dataspace/abalone/train_dropped.csv"},
        "columns_to_drop": ["id"]
    },
    "fill_missing_values_in_csv": {
        "inputs": {"csv_file": "./dataspace/abalone/train_dropped.csv"},
        "outputs": {
            "csv_file": "./dataspace/abalone/train_imputed.csv",
            "num_imputer": "./dataspace/abalone/num_imputer.pkl",
            "cat_imputer": "./dataspace/abalone/cat_imputer.pkl"
        },
```

```
        "label_column": "Rings"
    },
    "encode_categorical_columns_in_csv": {
        "inputs": {"csv_file": "./dataspace/abalone/train_imputed.csv"},
        "outputs": {
            "csv_file": "./dataspace/abalone/train_processed.csv",
            "encoder": "./dataspace/abalone/encoder.pkl"
        },
        "label_column": "Rings"
    },
    "random_split_csv": {
        "inputs": {"csv_file": "./dataspace/abalone/train_processed.csv"},
        "outputs": {
            "train_file": "./dataspace/abalone/train_split.csv",
            "valid_file": "./dataspace/abalone/valid_split.csv"
        },
        "test_ratio": 0.3
    },
    "train_random_forest_regressor": {
        "inputs": {
            "train_file": "./dataspace/abalone/train_split.csv",
            "valid_file": "./dataspace/abalone/valid_split.csv"
        },
        "outputs": {
            "model_file": "./dataspace/abalone/rf_model.pkl",
            "evaluation_file": "./dataspace/abalone/evaluation_rf.json"
        },
        "label_column": "Rings",
        "n_estimators" : 300,
        "max_depth": 10,
    },
    "transform_data_with_saved_preprocessors": {
        "inputs": {
            "csv_file": "./dataspace/abalone/test.csv",
            "num_imputer": "./dataspace/abalone/num_imputer.pkl",
            "cat_imputer": "./dataspace/abalone/cat_imputer.pkl",
            "encoder": "./dataspace/abalone/encoder.pkl"
```

```
        },
        "outputs": {"csv_file": "./dataspace/abalone/test_processed.csv"},
        "id_column": "id"
    },
    "predict_with_sklearn_model": {
        "inputs": {
            "model_file": "./dataspace/abalone/rf_model.pkl",
            "csv_file": "./dataspace/abalone/test_processed.csv"
        },
        "outputs": {"csv_file": "./dataspace/abalone/submission.csv"},
        "id_column": "id"
    },

    "random_split_csv": {
        "inputs": {"csv_file": "./dataspace/abalone/train_dropped.csv"},
        "outputs": {
            "train_file": "./dataspace/abalone/train_split.csv",
            "valid_file": "./dataspace/abalone/valid_split.csv"
        },
        "test_ratio": 0.3,
        "random_state": 42
    },

    "train_pycaret_automl_regressor": {
        "inputs": {
            "train_file": "./dataspace/abalone/train_split.csv",
            "valid_file": "./dataspace/abalone/valid_split.csv"
        },
        "outputs": {
            "model_file": "./dataspace/abalone/pycaret_automl_model",
            "evaluation_file": "./dataspace/abalone/pycaret_evaluation.json
    "
        },
        "label_column": "Rings",
        "session_id": 42
    },

```

```
    "predict_pycaret_automl_regressor": {
        "inputs": {
            "model_file": "./dataspace/abalone/pycaret_automl_model.pkl",
            "csv_file": "./dataspace/abalone/test.csv"
        },
        "outputs": {"csv_file": "./dataspace/abalone/submission_pycaret.csv
    "},
        "id_column": "id",
        "prediction_column": "Rings"
    }
}
```

C.2.2 Experiment 2 - Detailed DAG

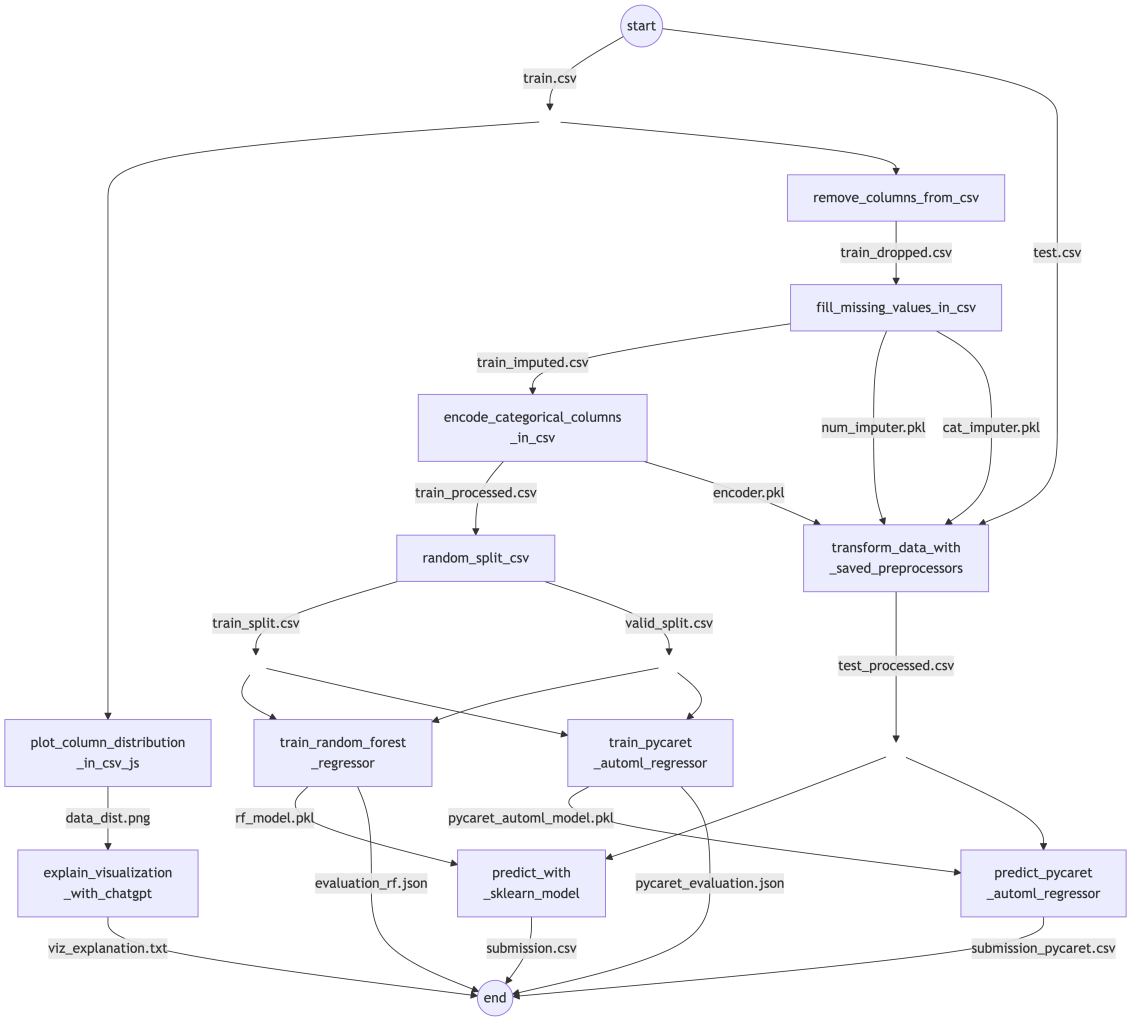

Figure C.2: Directed-acyclic graph of the *OpenVaccine* RNA-degradation pipeline.

References

1. Agarwal, J., Gupta, S., H, A.P.: Software Metrics for Assessing Reusability of Component Based Software System. In: 2023 13th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 287–291. IEEE, Noida, India (2023). https://doi.org/10.1109/Confluence56041.2023.10048812. https://ieeexplore.ieee.org/document/10048812/ (visited on 05/14/2025)

2. Akbulut, A., Perros, H.G.: Performance Analysis of Microservice Design Patterns. IEEE Internet Comput. **23**(6), 19–27 (2019). https://doi.org/10.1109/MIC.2019.2951094. https://ieeexplore.ieee.org/document/8890660/ (visited on 05/14/2025)

3. Alteryx, Alteryx: Data Science and Analytics Automation Platform, (2025). https://www.alteryx.com/. Accessed: 2025-01-18.

4. Antunes, A.L., Cardoso, E., Barateiro, J.: Incorporation of Ontologies in Data Warehouse/Business Intelligence Systems - A Systematic Literature Review. International Journal of Information Management Data Insights **2**(2), 100131 (2022). https://doi.org/10.1016/j.jjimei.2022.100131. https://linkinghub.elsevier.com/retrieve/pii/S266709682200074X (visited on 05/14/2025)

5. Badampudi, D., Usman, M., Chen, X.: Large scale reuse of microservices using DevOps and InnerSource practices – A longitudinal case study, (2023). https://doi.org/10.48550/arXiv.2309.15175. arXiv: 2309.15175[cs]. http://arxiv.org/abs/2309.15175 (visited on 05/14/2025).

6. Bandara, M., Rabhi, F.A., Bano, M.: A knowledge-driven approach for designing data analytics platforms. Requirements Eng **28**(2), 195–212 (2023). https://doi.org/10.1007/s00766-022-00385-5. https://link.springer.com/10.1007/s00766-022-00385-5 (visited on 05/14/2025)

7. Banerjee, S., Chandra, M.G.: A Software Framework for Procedural Knowledge Based Collaborative Data Analytics for IoT. In: 2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT), pp. 41–48. IEEE, Montreal, QC, Canada (2019). https://doi.org/10.1109/SERP4IoT.2019.00014. https://ieeexplore.ieee.org/document/8900706/ (visited on 05/14/2025)

8. Barba-Gonzalez, C., Caballero, I., Varela-Vaca, A.J., Cruz-Lemus, J.A., Gomez-Lopez, M.T., Navas-Delgado, I.: BIGOWL4DQ: Ontology-driven approach for Big Data quality meta-modelling, selection and reasoning. Information and Software Technology **167**, 107378 (2024). https://doi.org/10.1016/j.infsof.2023.107378. https://linkinghub.elsevier.com/retrieve/pii/S0950584923002331 (visited on 05/14/2025)

9. Barker, M., Chue Hong, N.P., Katz, D.S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L.J., Gruenpeter, M., Martinez, P.A., Honeyman, T.: Introducing the FAIR Principles for research software. Sci Data **9**(1), 622 (2022). https://doi.org/10.1038/s41597-022-01710-x. https://www.nature.com/articles/s41597-022-01710-x (visited on 05/14/2025)

10. Berisha, B., Mëziu, E., Shabani, I.: Big data analytics in Cloud computing: an overview. Journal of Cloud Computing (Heidelberg, Germany) **11** (2022). https://api.semanticscholar.org/CorpusID:251369860

11. Bhardwaj, A., Bhattacherjee, S., Chavan, A., Deshpande, A., Elmore, A.J., Madden, S., Parameswaran, A.G.: DataHub: Collaborative Data Science & Dataset Version Management at Scale, (2014). https://doi.org/10.48550/arXiv.1409.0798. arXiv: 1409.0798[cs]. http://arxiv.org/abs/1409.0798 (visited on 05/14/2025).

12. Bhardwaj, A., Deshpande, A., Elmore, A.J., Karger, D., Madden, S., Parameswaran, A., Subramanyam, H., Wu, E., Zhang, R.: Collaborative data analytics with DataHub. Proc. VLDB Endow. **8**(12), 1916–1919 (2015). https://doi.org/10.14778/2824032.2824100. https://dl.acm.org/doi/10.14778/2824032.2824100 (visited on 05/14/2025)

13. Bock, A.C., Frank, U.: Low-Code Platform. Bus Inf Syst Eng **63**(6), 733–740 (2021). https://doi.org/10.1007/s12599-021-00726-8. https://link.springer.com/10.1007/s12599-021-00726-8 (visited on 05/14/2025)

14. Brennan, S.E., Mueller, K., Zelinsky, G., Ramakrishnan, I., Warren, D.S., Kaufman, A.: Toward a Multi-Analyst, Collaborative Framework for Visual Analytics. In: 2006 IEEE Symposium On Visual Analytics Science And Technology, pp. 129–136 (2006). https://doi.org/10.1109/VAST.2006.261439

15. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language Models are Few-Shot Learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems, pp. 1877–1901. Curran Associates, Inc. (2020). https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

16. Callinan, C., Scott, M., Ojo, A., Whelan, E.: How to Create Public Value Through Open Data Driven Co-Creation: A Survey of the Literature. In: Proceedings of the 11th International Conference on Theory and Practice of Electronic Governance, pp. 363–370. ACM, Galway Ireland (2018). https://doi.org/10.1145/3209415.3209457. https://dl.acm.org/doi/10.1145/3209415.3209457 (visited on 05/14/2025)

17. Capilla, R., Gallina, B., Cetina, C., Favaro, J.: Opportunities for software reuse in an uncertain world: From past to emerging trends. J Software Evolu Process **31**(8), e2217 (2019). https://doi.org/10.1002/smr.2217. https://onlinelibrary.wiley.com/doi/10.1002/smr.2217 (visited on 05/21/2025)

18. Chen, D., Huang, Y., Ma, Z., Chen, H., Pan, X., Ge, C., Gao, D., Xie, Y., Liu, Z., Gao, J., Li, Y., Ding, B., Zhou, J.: Data-Juicer: A One-Stop Data Processing System for Large Language Models. In: Companion of the 2024 International Conference on Management of Data, pp. 120–134. ACM, Santiago AA Chile (2024). https://doi.org/10.1145/3626246.3653385. https://dl.acm.org/doi/10.1145/3626246.3653385 (visited on 05/14/2025)

19. Das, R., Wayment-Steele, H., Kim, D.S., Choe, C., Tunguz, B., Reade, W., Demkin, M.: OpenVaccine: COVID-19 mRNA Vaccine Degradation Prediction, https://kaggle.com/competitions/stanford-covid-vaccine (2020). Kaggle.

20. Dong, Y., Jiang, X., Jin, Z., Li, G.: Self-Collaboration Code Generation via ChatGPT. ACM Trans. Softw. Eng. Methodol. **33**(7), 1–38 (2024). https://doi.org/10.1145/3672459. https://dl.acm.org/doi/10.1145/3672459 (visited on 05/14/2025)

21. Douville, S., Grandjean Targos, P.T., Jones, N.D., Knight, C., Azzam, T.: Data Visualization Expert Lessons Learned: Implications for Program Evaluators. The American journal of evaluation (2025)

22. Du, Y., Li, S., Torralba, A., Tenenbaum, J.B., Mordatch, I.: Improving Factuality and Reasoning in Language Models through Multiagent Debate, en (2023). https://doi.org/10.48550/arXiv.2305.14325. http://arxiv.org/abs/2305.14325 (visited on 05/22/2025). arXiv:2305.14325 [cs].

23. Duan, Y., Edwards, J.S., Dwivedi, Y.K.: Artificial intelligence for decision making in the era of Big Data – evolution, challenges and research agenda. International Journal of Information Management **48**, 63–71 (2019). https://doi.org/10.1016/j.ijinfomgt.2019.01.021. https://linkinghub.elsevier.com/retrieve/pii/S0268401219300581 (visited on 05/14/2025)

24. Dzulhikam, D., Rana, M.E.: A Critical Review of Cloud Computing Environment for Big Data Analytics. 2022 International Conference on Decision Aid Sciences and Applications (DASA) (2022). https://api.semanticscholar.org/CorpusID:248517148

25. Fillbrunn, A., Dietz, C., Pfeuffer, J., Rahn, R., Landrum, G.A., Berthold, M.R.: KNIME for reproducible cross-domain analysis of life science data. Journal of Biotechnology **261**, 149–156 (2017). https://doi.org/10.1016/j.jbiotec.2017.07.028. https://linkinghub.elsevier.com/retrieve/pii/S0168165617315651 (visited on 05/14/2025)

26. Franken, S., Kolvenbach, S., Prinz, W., Alvertis, I., Koussouris, S.: CloudTeams: Bridging the Gap Between Developers and Customers During Software Development Processes. Procedia Computer Science **68**, 188–195 (2015). https://doi.org/10.1016/j.procs.2015.09.234. https://linkinghub.elsevier.com/retrieve/pii/S1877050915030793 (visited on 05/14/2025)

27. Fried, D., Aghajanyan, A., Lin, J., Wang, S., Wallace, E., Shi, F., Zhong, R., Yih, W.-t., Zettlemoyer, L., Lewis, M.: InCoder: A Generative Model for Code Infilling and Synthesis, (2023). https://doi.org/10.48550/arXiv.2204.05999. arXiv: 2204.05999[cs]. http://arxiv.org/abs/2204.05999 (visited on 05/21/2025).

28. Gandomi, A., Haider, M.: Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management **35**(2), 137–144 (2015). https://doi.org/10.1016/j.ijinfomgt.2014.10.007. https://linkinghub.elsevier.com/retrieve/pii/S0268401214001066 (visited on 05/14/2025)

29. Gu, Y., Cao, J., Guo, Y., Qian, S., Guan, W.: Plan, Generate and Match: Scientific Workflow Recommendation with Large Language Models. In: Monti, F., Rinderle-Ma, S., Ruiz Cortés, A., Zheng, Z., Mecella, M. (eds.) Service-Oriented Computing, pp. 86–102. Springer Nature Switzerland, Cham (2023)

30. Hasselbring, W., Carr, L., Hettrick, S., Packer, H., Tiropanis, T.: From FAIR research data toward FAIR and open research software. it - Information Technology **62**(1), 39–47 (2020). https://doi.org/10.1515/itit-2019-0040. https://www.degruyter.com/document/doi/10.1515/itit-2019-0040/html (visited on 05/14/2025)

31. He, X., Zhao, K., Chu, X.: AutoML: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021). https://doi.org/10.1016/j.knosys.2020.106622. https://www.sciencedirect.com/science/article/pii/S0950705120307516

32. Hevner, March, Park, Ram, Design Science in Information Systems Research. MIS Quarterly **28**(1), 75 (2004). https://doi.org/10.2307/25148625. https://www.jstor.org/stable/10.2307/25148625 (visited on 06/06/2025)

33. Howard, A., inversion, Makridakis, S., vangelis, M5 Forecasting - Accuracy, https://kaggle.com/competitions/m5-forecasting-accuracy (2020). Kaggle.

34. Hu, Y., Liu, D., Wang, Q., Yu, C., Ji, H., Xiong, J.: Automating Knowledge Discovery from Scientific Literature via LLMs: A Dual-Agent Approach with Progressive Ontology Prompting, (2024). https://doi.org/10.48550/arXiv.2409.00054. arXiv: 2409.00054[cs]. http://arxiv.org/abs/2409.00054 (visited on 05/14/2025).

35. John, R.J.L., Bacon, D., Chen, J., Ramesh, U., Li, J., Das, D., Claus, R., Kendall, A., Patel, J.M.: DataChat: An Intuitive and Collaborative Data Analytics Platform. In: Companion of the 2023 International Conference on Management of Data, pp. 203–215. ACM, Seattle WA USA (2023). https://doi.org/10.1145/3555041.3589678. https://dl.acm.org/doi/10.1145/3555041.3589678 (visited on 05/14/2025)

36. Kaeser-Chen, C., Pathology, F., Maggie, Dane, S.: Plant Pathology 2020 - FGVC7, https://kaggle.com/competitions/plant-pathology-2020-fgvc7 (2020). Kaggle.

37. Kahn, A.B.: Topological sorting of large networks. Commun. ACM **5**(11), 558–562 (1962). https://doi.org/10.1145/368996.369025

38. Kandogan, E., Roth, M., Schwarz, P., Hui, J., Terrizzano, I., Christodoulakis, C., Miller, R.J.: LabBook: Metadata-driven social collaborative data analysis. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 431–440. IEEE, Santa Clara, CA, USA (2015). https://doi.org/10.1109/BigData.2015.7363784. http://ieeexplore.ieee.org/document/7363784/ (visited on 05/14/2025)

39. Kapitsaki, G.M.: Generative AI for Code Generation: Software Reuse Implications. In: Achilleos, A., Fuentes, L., Papadopoulos, G.A. (eds.) Reuse and Software Quality, pp. 37–47. Springer Nature Switzerland, Cham (2024)

40. Karacapilidis, N., Christodoulou, S., Tzagarakis, M., Tsiliki, G., Pappis, C.: Strengthening collaborative data analysis and decision making in web communities. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 1005–1010. ACM, Seoul Korea (2014). https://doi.org/10.1145/2567948.2578845. https://dl.acm.org/doi/10.1145/2567948.2578845 (visited on 05/14/2025)

41. Khalajzadeh, H., Abdelrazek, M., Grundy, J., Hosking, J., He, Q.: Survey and Analysis of Current End-User Data Analytics Tool Support. IEEE Trans. Big Data **8**(1), 152–165 (2022). https://doi.org/10.1109/TBDATA.2019.2921774. https://ieeexplore.ieee.org/document/8733046/ (visited on 05/14/2025)

42. Khalifa, S., Elshater, Y., Sundaravarathan, K., Bhat, A., Martin, P., Imam, F., Rope, D., Mcroberts, M., Statchuk, C.: The Six Pillars for Building Big Data Analytics Ecosystems. ACM Comput. Surv. **49**(2), 1–36 (2017). https://doi.org/10.1145/2963143. https://dl.acm.org/doi/10.1145/2963143 (visited on 05/14/2025)

43. Kienzle, J., Mussbacher, G., Alam, O., Schöttle, M., Belloir, N., Collet, P., Combemale, B., DeAntoni, J., Klein, J., Rumpe, B.: VCU: The Three Dimensions of Reuse. In: Kapitsaki, G.M., Santana de Almeida, E. (eds.) Software Reuse: Bridging with Social-Awareness, pp. 122–137. Springer International Publishing, Cham (2016)

44. Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M., Wiedenbeck, S.: The state of the art in end-user software engineering. ACM Comput. Surv. **43**(3), 1–44 (2011). https://doi.org/10.1145/1922649.1922658. https://dl.acm.org/doi/10.1145/1922649.1922658 (visited on 05/14/2025)

45. Kolpakov, F., Akberdin, I., Kashapov, T., Kiselev, L., Kolmykov, S., Kondrakhin, Y., Kutumova, E., Mandrik, N., Pintus, S., Ryabova, A., Sharipov, R., Yevshin, I., Kel, A.: BioUML: an integrated environment for systems biology and collaborative analysis of biomedical data. Nucleic Acids Research **47**, W225–W233 (2019). https://doi.org/10.1093/nar/gkz440. https://academic.oup.com/nar/article/47/W1/W225/5498754 (visited on 05/14/2025)

46. Kuang, E., Jin, X., Fan, M.: "Merging Results Is No Easy Task": An International Survey Study of Collaborative Data Analysis Practices Among UX Practitioners. In: CHI Conference on Human Factors in Computing Systems, pp. 1–16 (2022). https://doi.org/10.1145/3491102.3517647. arXiv: 2204.02823[cs]. http://arxiv.org/abs/2204.02823 (visited on 05/14/2025)

47. Langenkamp, M., Yue, D.N.: How Open Source Machine Learning Software Shapes AI. Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society (2022). https://api.semanticscholar.org/CorpusID:251107220

48. Lazarova-Molnar, S., Mohamed, N.: Collaborative data analytics for smart buildings: opportunities and models. Cluster Comput **22**, 1065–1077 (2019). https://doi.org/10.1007/s10586-017-1362-x. http://link.springer.com/10.1007/s10586-017-1362-x (visited on 05/14/2025)

49. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS '20. Curran Associates Inc., Vancouver, BC, Canada (2020)

50. Li, G., Zhou, X., Zhao, X.: LLM for Data Management. Proc. VLDB Endow. **17**(12), 4213–4216 (2024). https://doi.org/10.14778/3685800.3685838. https://dl.acm.org/doi/10.14778/3685800.3685838 (visited on 05/14/2025)

51. Lismont, J., Van Calster, T., Óskarsdóttir, M., Vanden Broucke, S., Baesens, B., Lemahieu, W., Vanthienen, J.: Closing the Gap Between Experts and Novices Using Analytics-as-a-Service: An Experimental Study. Bus Inf Syst Eng **61**(6), 679–693 (2019). https://doi.org/10.1007/s12599-018-0539-z. http://link.springer.com/10.1007/s12599-018-0539-z (visited on 05/21/2025)

52. Liu, C., Vitagliano, G., Rose, B., Prinz, M., Samson, D.A., Cafarella, M.: PalimpChat: Declarative and Interactive AI analytics, (2025). https://doi.org/10.48550/arXiv.2502.03368. arXiv: 2502.03368[cs]. http://arxiv.org/abs/2502.03368 (visited on 05/14/2025).

53. Liu, M.X., Kittur, A., Myers, B.A.: To Reuse or Not To Reuse?: A Framework and System for Evaluating Summarized Knowledge. Proc. ACM Hum.-Comput. Interact. **5**, 1–35 (2021). https://doi.org/10.1145/3449240. https://dl.acm.org/doi/10.1145/3449240 (visited on 05/21/2025)

54. Liu, S.-C., Wang, S., Lin, W., Hsiung, C.-W., Hsieh, Y.-C., Cheng, Y.-P., Luo, S.-H., Chang, T., Zhang, J.: JarviX: A LLM No code Platform for Tabular Data Analysis and Optimization, (2023). https://doi.org/10.48550/arXiv.2312.02213. arXiv: 2312.02213[cs]. http://arxiv.org/abs/2312.02213 (visited on 05/14/2025).

55. Liu, S., Biswal, A., Kamsetty, A., Cheng, A., Schroeder, L.G., Patel, L., Cao, S., Mo, X., Stoica, I., Gonzalez, J.E., Zaharia, M.: Optimizing LLM Queries in Relational Data Analytics Workloads, (2025). https://doi.org/10.48550/arXiv.2403.05821. arXiv: 2403.05821[cs]. http://arxiv.org/abs/2403.05821 (visited on 05/14/2025).

56. Liu, X., Wang, Z., Ni, S., Alsudais, S., Huang, Y., Kumar, A., Li, C.: Demonstration of collaborative and interactive workflow-based data analytics in texera. Proc. VLDB Endow. **15**(12), 3738–3741 (2022). https://doi.org/10.14778/3554821.3554888. https://dl.acm.org/doi/10.14778/3554821.3554888 (visited on 05/14/2025)

57. Mäkitalo, N., Taivalsaari, A., Kiviluoto, A., Mikkonen, T., Capilla, R.: On opportunistic software reuse. Computing **102**(11), 2385–2408 (2020). https://doi.org/10.1007/s00607-020-00833-6. https://link.springer.com/10.1007/s00607-020-00833-6 (visited on 05/21/2025)

58. Mao, Y., Wang, D., Muller, M., Varshney, K.R., Baldini, I., Dugan, C., Mojsilović, A.: How Data ScientistsWork Together With Domain Experts in Scientific Collaborations: To Find The Right Answer Or To Ask The Right Question? Proc. ACM Hum.-Comput. Interact. **3**, 1–23 (2019). https://doi.org/10.1145/3361118. https://dl.acm.org/doi/10.1145/3361118 (visited on 05/14/2025)

59. Martinez-Maldonado, R., Gašević, D., Echeverria, V., Fernandez Nieto, G., Swiecki, Z., Buckingham Shum, S.: What Do You Mean by Collaboration Analytics? A Conceptual Model. JLA **8**(1), 126–153 (2021). https://doi.org/10.18608/jla.2021.7227. https://www.learning-analytics.info/index.php/JLA/article/view/7227 (visited on 05/21/2025)

60. Marzukhi, S., Awang, N., Alsagoff, S.N., Mohamed, H.: RapidMiner and Machine Learning Techniques for Classifying Aircraft Data. J. Phys.: Conf. Ser. **1997**(1), 012012 (2021). https://doi.org/10.1088/1742-6596/1997/1/012012. https://iopscience.iop.org/article/10.1088/1742-6596/1997/1/012012 (visited on 05/14/2025)

61. Mathisen, A., Horak, T., Klokmose, C.N., Grønbæk, K., Elmqvist, N.: InsideInsights: Integrating Data-Driven Reporting in Collaborative Visual Analytics. Computer Graphics Forum **38**(3), 649–661 (2019). https://doi.org/10.1111/cgf.13717. https://onlinelibrary.wiley.com/doi/10.1111/cgf.13717 (visited on 05/14/2025)

62. Merkelbach, S., Von Enzberg, S., Kuhn, A., Dumitrescu, R.: Towards a Process Model to Enable Domain Experts to Become Citizen Data Scientists for Industrial Applications. In: 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS), pp. 1–6. IEEE, Coventry, United Kingdom (2022). https://doi.org/10.1109/ICPS51978.2022.9816871. https://ieeexplore.ieee.org/document/9816871/ (visited on 05/21/2025)

63. Michalczyk, S., Nadj, M., Azarfar, D., Mädche, A., Gröger, C.: A State-of-the-Art Overview and Future Research Avenues of Self-Service Business Intelligence and Analytics. In: Proceedings of the 28th European Conference on Information Systems (ECIS), An Online AIS Conference (Marrakesh, Marokko), June 15-17, 2020, Art.Nr. 46. AIS eLibrary (AISeL) (2020)

64. Mineau, J.-M., Lalande, J.-F.: Evaluating the Reusability of Android Static Analysis Tools. In: Achilleos, A., Fuentes, L., Papadopoulos, G.A. (eds.) Reuse and Software Quality, pp. 153–170. Springer Nature Switzerland, Cham (2024)

65. Montoya, A., DataCanary, House Prices - Advanced Regression Techniques, https://kaggle.com/competitions/house-prices-advanced-regression-techniques (2016). Kaggle.

66. Montoya, A., inversion, KirillOdintsov, Kotek, M.: Home Credit Default Risk, https://kaggle.com/competitions/home-credit-default-risk (2018). Kaggle.

67. Nejjar, M., Zacharias, L., Stiehle, F., Weber, I.: LLMs for Science: Usage for Code Generation and Data Analysis, (2024). https://doi.org/10.48550/arXiv.2311.16733. arXiv: 2311.16733[cs]. http://arxiv.org/abs/2311.16733 (visited on 05/14/2025).

68. Oinn, T.M., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, R.M., Carver, T.J., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20 17**, 3045–54 (2004). https://api.semanticscholar.org/CorpusID:1375038

69. Papamichail, M.D., Diamantopoulos, T., Symeonidis, A.L.: Measuring the reusability of software components using static analysis metrics and reuse rate information. Journal of Systems and Software **158**, 110423 (2019). https://doi.org/10.1016/j.jss.2019.110423. https://linkinghub.elsevier.com/retrieve/pii/S0164121219301979 (visited on 05/14/2025)

70. Passi, S., Jackson, S.J.: Trust in Data Science: Collaboration, Translation, and Accountability in Corporate Data Science Projects. Proc. ACM Hum.-Comput. Interact. **2**, 1–28 (2018). https://doi.org/10.1145/3274405. https://dl.acm.org/doi/10.1145/3274405 (visited on 05/14/2025)

71. Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. J. Manage. Inf. Syst. **24**(3), 45–77 (2007). https://doi.org/10.2753/MIS0742-1222240302

72. Piorkowski, D., Park, S., Wang, A.Y., Wang, D., Muller, M., Portnoy, F.: How AI Developers Overcome Communication Challenges in a Multidisciplinary Team: A Case Study. Proc. ACM Hum.-Comput. Interact. **5**, 1–25 (2021). https://doi.org/10.1145/3449205. arXiv: 2101.06098[cs]. http://arxiv.org/abs/2101.06098 (visited on 05/14/2025)

73. Project Jupyter, https://jupyter.org (visited on 06/07/2024).

74. Rahmatulloh, A., Nugraha, F., Gunawan, R., Darmawan, I., Haerani, E., Rizal, R.: Event-Driven Architecture (EDA) vs API-Driven Architecture (ADA): Which Performs Better in Microservices? In: 2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD), pp. 31–36. IEEE, Indonesia (2024). https://doi.org/10.1109/ICoABCD63526.2024.10704326. https://ieeexplore.ieee.org/document/10704326/ (visited on 05/14/2025)

75. Reade, W., Chow, A.: Regression with an Abalone Dataset, https://kaggle.com/competitions/playground-series-s4e4 (2024). Kaggle.

76. Saied, M.A., Ouni, A., Sahraoui, H., Kula, R.G., Inoue, K., Lo, D.: Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software **145**, 164–179 (2018). https://doi.org/10.1016/j.jss.2018.08.032. https://linkinghub.elsevier.com/retrieve/pii/S0164121218301699 (visited on 05/14/2025)

77. Sarkar, A.: Will Code Remain a Relevant User Interface for End-User Programming with Generative AI Models? In: Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, pp. 153–167 (2023). https://doi.org/10.1145/3622758.3622882. arXiv: 2311.00382[cs]. http://arxiv.org/abs/2311.00382 (visited on 05/21/2025)

78. Schmidt, P.J., Riley, J., Swanson Church, K.: Investigating Accountants' Resistance to Move beyond Excel and Adopt New Data Analytics Technology. Accounting Horizons **34**(4), 165–180 (2020). https://doi.org/10.2308/HORIZONS-19-154. https://publications.aaahq.org/accounting-horizons/article/34/4/165/2435/Investigating-Accountants-Resistance-to-Move (visited on 05/14/2025)

79. Starlinger, J., Brancotte, B., Cohen-Boulakia, S., Leser, U.: Similarity search for scientific workflows. Proc. VLDB Endow. **7**(12), 1143–1154 (2014). https://doi.org/10.14778/2732977.2732988

80. Tao, C., Rina, S., Yongjuan, Z., Xin, Y., Rui, Z.: Ontology Service Center: A Datahub for Ontology Application. IJWesT **12**(3), 01–14 (2021). https://doi.org/10.5121/ijwest.2021.12301. https://aircconline.com/ijwest/V12N3/12321ijwest01.pdf (visited on 05/14/2025)

81. Taques, F.H., Chasco, C., Taques, F.H.: Integrating big data with KNIME as an alternative without programming code: an application to the PATSTAT patent database. J Geogr Syst **27**(1), 31–61 (2025). https://doi.org/10.1007/s10109-024-00445-0. https://link.springer.com/10.1007/s10109-024-00445-0 (visited on 05/14/2025)

82. The Galaxy Community, Abueg, L.A.L., Afgan, E., Allart, O., Awan, A.H., Bacon, W.A., Baker, D., Bassetti, M., Batut, B., Bernt, M., Blankenberg, D., Bombarely, A., Bretaudeau, A., Bromhead, C.J., Burke, M.L., Capon, P.K., Čech, M., Chavero-Díez, M., Chilton, J.M., Collins, T.J., Coppens, F., Coraor, N., Cuccuru, G., Cumbo, F., Davis, J., De Geest, P.F., De Koning, W., Demko, M., DeSanto, A., Begines, J.M.D., Doyle, M.A., Droesbeke, B., Erxleben-Eggenhofer, A., Föll, M.C., Formenti, G., Fouilloux, A., Gangazhe, R., Genthon, T., Goecks, J., Beltran, A.N.G., Goonasekera, N.A., Goué, N., Griffin, T.J., Grüning, B.A., Guerler, A., Gundersen, S., Gustafsson, O.J.R., Hall, C., Harrop, T.W., Hecht, H., Heidari, A., Heisner, T., Heyl, F., Hiltemann, S., Hotz, H.-R., Hyde, C.J., Jagtap, P.D., Jakiela, J., Johnson, J.E., Joshi, J., Jossé, M., Jum'ah, K., Kalaš, M., Kamieniecka, K., Kayikcioglu, T., Konkol, M., Kostrykin, L., Kucher, N., Kumar, A., Kuntz, M., Lariviere, D., Lazarus, R., Bras, Y.L., Corguillé, G.L., Lee, J., Leo, S., Liborio, L., Libouban, R., Tabernero, D.L., Lopez-Delisle, L., Los, L.S., Mahmoud, A., Makunin, I., Marin, P., Mehta, S., Mok, W., Moreno, P.A., Morier-Genoud, F., Mosher, S., Müller, T., Nasr, E., Nekrutenko, A., Nelson, T.M., Oba, A.J., Ostrovsky, A., Polunina, P.V., Poterlowicz, K., Price, E.J., Price, G.R., Rasche, H., Raubenolt, B., Royaux, C., Sargent, L., Savage, M.T., Savchenko, V., Savchenko, D., Schatz, M.C., Seguineau, P., Serrano-Solano, B., Soranzo, N., Srikakulam, S.K., Suderman, K., Syme, A.E., Tangaro, M.A., Tedds, J.A., Tekman, M., Cheng (Mike) Thang, W., Thanki, A.S., Uhl, M., Van Den Beek, M., Varshney, D., Vessio, J., Videm, P., Von Kuster, G., Watson, G.R., Whitaker-Allen, N., Winter, U., Wolstencroft, M., Zambelli, F., Zierep, P., Zoabi, R.: The Galaxy platform for accessible, reproducible, and collaborative data analyses: 2024 update. Nucleic Acids Research **52**, W83–W94 (2024). https://doi.org/10.1093/nar/gkae410. https://academic.oup.com/nar/article/52/W1/W83/7676834 (visited on 05/14/2025)

83. Tucker, I., Gil-Garcia, J.R., Sayogo, D.S.: Collaborative Data Analytics for Emergency Response: Identifying Key Factors and Proposing a Preliminary Framework. In: Proceedings of the 10th International Conference on Theory and Practice of Electronic Governance, pp. 508–515. ACM, New Delhi AA India (2017). https://doi.org/10.1145/3047273.3047379. https://dl.acm.org/doi/10.1145/3047273.3047379 (visited on 05/14/2025)

84. Velepucha, V., Flores, P.: A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access **11**, 88339–88358 (2023). https://doi.org/10.1109/ACCESS.2023.3305687. https://ieeexplore.ieee.org/document/10220070/ (visited on 05/14/2025)

85. Vogt, L., Strömert, P., Matentzoglu, N., Karam, N., Konrad, M., Prinz, M., Baum, R.: Suggestions for extending the FAIR Principles based on a linguistic perspective on semantic interoperability. Sci

Data **12**(1), 688 (2025). https://doi.org/10.1038/s41597-025-05011-x. https://www.nature.com/articles/s41597-025-05011-x (visited on 05/14/2025)

86. Wang, D.Y.-B., Shen, Z., Mishra, S.S., Xu, Z., Teng, Y., Ding, H.: SLOT: Structuring the Output of Large Language Models, (2025). arXiv: 2505.04016 [cs.CL]. https://arxiv.org/abs/2505.04016.

87. Wang, Z., Huang, Y., Ni, S., Kumar, A., Alsudais, S., Liu, X., Lin, X., Ding, Y., Li, C.: Texera: A System for Collaborative and Interactive Data Analytics Using Workflows. Proc. VLDB Endow. **17**(11), 3580–3588 (2024). https://doi.org/10.14778/3681954.3682022. https://dl.acm.org/doi/10.14778/3681954.3682022 (visited on 05/14/2025)

88. Wang, Z., Li, C.: Building a Collaborative Data Analytics System: Opportunities and Challenges. Proc. VLDB Endow. **16**(12), 3898–3901 (2023). https://doi.org/10.14778/3611540.3611580. https://dl.acm.org/doi/10.14778/3611540.3611580 (visited on 05/14/2025)

89. Weng, L., Tang, Y., Feng, Y., Chang, Z., Chen, R., Feng, H., Hou, C., Huang, D., Li, Y., Rao, H., Wang, H., Wei, C., Yang, X., Zhang, Y., Zheng, Y., Huang, X., Zhu, M., Ma, Y., Cui, B., Chen, P., Chen, W.: DataLab: A Unified Platform for LLM-Powered Business Intelligence, (2025). arXiv: 2412.02205 [cs.DB]. https://arxiv.org/abs/2412.02205.

90. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Da Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J., Groth, P., Goble, C., Grethe, J.S., Heringa, J., 'T Hoen, P.A., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., Van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., Van Der Lei, J., Van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: The FAIR Guiding Principles for scientific data management and stewardship. Sci Data **3**(1), 160018 (2016). https://doi.org/10.1038/sdata.2016.18. https://www.nature.com/articles/sdata201618 (visited on 05/14/2025)

91. Wilkinson, S.R., Aloqalaa, M., Belhajjame, K., Crusoe, M.R., Kinoshita, B.d.P., Gadelha, L., Garijo, D., Gustafsson, O.J.R., Juty, N., Kanwal, S., Khan, F.Z., Köster, J., Gehlen, K.P.-v., Pouchard, L., Rannow, R.K., Soiland-Reyes, S., Soranzo, N., Sufi, S., Sun, Z., Vilne, B., Wouters, M.A., Yuen, D., Goble, C.: Applying the FAIR Principles to computational workflows. Sci Data **12**(1), 328 (2025). https://doi.org/10.1038/s41597-025-04451-9. arXiv: 2410.03490[cs]. http://arxiv.org/abs/2410.03490 (visited on 05/14/2025)

92. Xu, W., Luo, G., Meng, W., Zhai, X., Zheng, K., Wu, J., Li, Y., Xing, A., Li, J., Li, Z., Zheng, K., Li, K.: MRAgent: an LLM-based automated agent for causal knowledge discovery in disease via Mendelian randomization. Briefings in Bioinformatics **26**(2), bbaf140 (2025). https://doi.org/10.1093/bib/bbaf140. https://academic.oup.com/bib/article/doi/10.1093/bib/bbaf140/8107848 (visited on 05/14/2025)

93. Yamakami, T.: A Handover Challenge of Data Analytics: Multi-user Issues in Sustainable Data Analytics. In: Barolli, L., Okada, Y., Amato, F. (eds.) Advances in Internet, Data and Web Technologies, pp. 373–383. Springer International Publishing, Cham (2020)

94. Yao, L., Rabhi, F.A.: Building architectures for data-intensive science using the ADAGE framework. Concurrency and Computation **27**(5), 1188–1206 (2015). https://doi.org/10.1002/cpe.3280. https://onlinelibrary.wiley.com/doi/10.1002/cpe.3280 (visited on 05/14/2025)

95. Ye, Y., Fischer, G.: Reuse-Conducive Development Environments. Autom Software Eng **12**(2), 199–235 (2005). https://doi.org/10.1007/s10515-005-6206-x. http://link.springer.com/10.1007/s10515-005-6206-x (visited on 05/14/2025)

96. Zhang, A.X., Muller, M., Wang, D.: How do Data Science Workers Collaborate? Roles, Workflows, and Tools, (2020). https://doi.org/10.48550/arXiv.2001.06684. arXiv: 2001.06684[cs]. http://arxiv.org/abs/2001.06684 (visited on 05/14/2025).

97. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., Wen, J.-R.: A Survey of Large Language Models, (2025). arXiv: 2303.18223 [cs.CL]. https://arxiv.org/abs/2303.18223.

98. Zhong, C., Oruongo, J., Kim, J.B.J.B.: LLM-Powered Low-Code/No-Code Data Analytics in Education and Workforce Development. Computer **58**(3), 49–59 (2025). https://doi.org/10.1109/MC.2024.3516614. https://ieeexplore.ieee.org/document/10897937/ (visited on 05/21/2025)

99. Zhou, Z., Jin, J., Phadnis, V., Yuan, X., Jiang, J., Qian, X., Wright, K., Sherwood, M., Mayes, J., Zhou, J., Huang, Y., Xu, Z., Zhang, Y., Lee, J., Olwal, A., Kim, D., Iyengar, R., Li, N., Du, R.: InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs. In: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, pp. 1–22 (2025). https://doi.org/10.1145/3706598.3713905. arXiv: 2312.09672[cs]. http://arxiv.org/abs/2312.09672 (visited on 05/14/2025)

100. Zozas, I., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Avgeriou, P., Stamelos, I.: REI: An integrated measure for software reusability. J Software Evolu Process **31**(8), e2216 (2019). https://doi.org/10.1002/smr.2216. https://onlinelibrary.wiley.com/doi/10.1002/smr.2216 (visited on 05/14/2025)