DIPLOMA THESIS

Building Bridges in Research: Automated Capture and Semantic Mapping of Provenance in Virtual Research Environments.

submitted in partial fulfillment of the requirements for the degree of
Diplom-Ingenieur

in
Data Science

by
Reema George Dass
Registration Number: 12144026

to the
Faculty of Informatics
at the
Vienna University of Technology

Advisor: Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Vienna, August 31, 2025

Reema George Dass	Andreas Rauber

Technische Universität Wien
A-1040 Wien, Karlsplatz 13 Tel. +43-1-58801-0 www.tuwien.ac.at

Declaration of Authorship

I, **Reema George Dass**, hereby declare that I have independently written this thesis, that I have fully disclosed all sources and aids used, and that all passages of this work, including tables, maps, and figures taken either verbatim or in essence from other works or from the internet have been clearly marked as such with appropriate references.

The use of AI-based tools, particularly OpenAI's GPT-4, which enhanced early versions and improved sections of the introduction and literature review, is something I would want to acknowledge. Every text was meticulously reviewed and modified to match my own research, analysis, and academic style, even though the AI offered a structured starting point. With caution to maintain academic integrity, this tool was used as an additional resource to enhance the writing process.

Vienna, August 31, 2025

Reema George Dass

Abstract

Machine learning workflows in modern scientific research are often fragmented across multiple tools and environments, resulting in scattered or incomplete metadata. This fragmentation makes it impracticable to determine how models were trained, which datasets were utilized, and what decisions impacted experimental results. As a result, transparency, reproducibility, and auditability deteriorate, limiting the long-term reliability and reusable nature of machine learning research.

This thesis examines how semantic provenance techniques can be included into Virtual Research Environments (VREs) in order to address these problems. By defining key metadata needs across the machine learning lifecycle, we create a formal mapping to community standards like as FAIR, FAIR4ML, PROV-O, Croissant, and MLSEA. Based on this basis, we suggest a strategy for integrating diverse metadata sources into a semantically rich representation.

This approach is implemented within a modular framework that combines JupyterHub, GitHub, DBRepo, and Invenio, with MLflow support for semi-automated metadata gathering. Key aspects including dataset provenance, training setups, runtime environment, and justification information are exported in machine-readable forms (JSON-LD and RDF/XML). A Streamlit-based dashboard makes it easier to visualize, compare experiment runs, and do SPARQL-based provenance queries.

The framework is assessed using targeted experiments and a user study with two machine learning practitioners. The results show that over 70% of field-level metadata was captured using automation or semi-automation, confirming the framework's usability, traceability, and conformity with semantic standards. Overall, this work provides a repeatable and standards-compliant solution to manage provenance in machine learning pipelines within VREs.

Kurzfassung

Machine-Learning-Workflows in der modernen wissenschaftlichen Forschung sind oft über mehrere Tools und Umgebungen fragmentiert, was zu verstreuten oder unvollständigen Metadaten führt. Diese Fragmentierung macht es unmöglich, nachzuvollziehen, wie Modelle trainiert, welche Datensätze verwendet und welche Entscheidungen die experimentellen Ergebnisse beeinflusst haben. Dadurch verschlechtern sich Transparenz, Reproduzierbarkeit und Überprüfbarkeit, was die langfristige Zuverlässigkeit und Wiederverwendbarkeit der Machine-Learning-Forschung einschränkt.

Diese Arbeit untersucht, wie semantische Provenienztechniken in virtuelle Forschungsumgebungen (VREs) integriert werden können, um diese Probleme zu lösen. Durch die Definition der wichtigsten Metadatenanforderungen über den gesamten Lebenszyklus des maschinellen Lernens erstellen wir eine formale Zuordnung zu Community-Standards wie FAIR, FAIR4ML, PROV-O, Croissant und MLSEA. Auf dieser Grundlage schlagen wir eine Strategie zur Integration verschiedener Metadatenquellen in eine semantisch reichhaltige Repräsentation vor.

Dieser Ansatz wird in einem modularen Framework implementiert, das JupyterHub, GitHub, DBRepo und Invenio mit MLflow-Unterstützung für die halbautomatische Metadatenerfassung kombiniert. Wichtige Aspekte wie Datensatzherkunft, Trainings-Setups, Laufzeitumgebung und Begründungsinformationen werden in maschinenlesbaren Formaten (JSON-LD und RDF/XML) exportiert. Ein Streamlit-basiertes Dashboard erleichtert die Visualisierung, den Vergleich von Experimentläufen und SPARQL-basierte Herkunftsabfragen.

Das Framework wurde anhand gezielter Experimente und einer Nutzerstudie mit zwei Machine-Learning-Experten bewertet. Die Ergebnisse zeigen, dass über 70%

der Metadaten auf Feldebene automatisiert oder halbautomatisiert erfasst wurden. Dies bestätigt die Benutzerfreundlichkeit, Rückverfolgbarkeit und Konformität des Frameworks mit semantischen Standards. Insgesamt bietet diese Arbeit eine wiederholbare und standardkonforme Lösung für das Herkunftsmanagement in Machine-Learning-Pipelines innerhalb von VREs.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Andreas Rauber, Associate Professor at TU Wien, for his insightful advice, supportive encouragement, and constructive feedback during the writing of my thesis. His extensive expertise in research data management, digital preservation, reproducibility, and FAIR-aligned infrastructures has been instrumental in shaping both the technical and conceptual direction. I sincerely appreciate the time and guidance he dedicated throughout this work.

Special gratitude to my family and friends for their unwavering support, patience, and faith in me throughout times of stress and uncertainty, and to my mother, Jenet G Dass, whose constant belief in me has been my anchor and greatest source of strength.

I'd also want to thank the developers and maintainers of open-source tools like MLflow, Streamlit, and scikit-learn for their contributions to the prototype system that forms the basis of this work.

Finally, I would like to thank the academic community for their past research on metadata standards, data management, and Virtual Research Environments (VREs), which served as the foundation for our study.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Machine Learning (ML) procedures have become essential in current scientific research, enabling everything from exploratory data analysis to predictive modeling. As these workflows develop in size and complexity, ensuring transparency, reproducibility, auditability, and traceability becomes more important-but remains an everyday challenge.

Today's ML experimental processes are frequently scattered among tools, computational environments, and storage systems. Datasets may be reused in several studies without effectively documenting their origin or revisions. Models may be trained and retrained without noting the reasoning for crucial decisions. This absence of organized metadata makes it harder to replicate past studies, troubleshoot procedures, and assess the trustworthiness of results, lowering the research quality and hindering progress.

To address these issues, Virtual Research Environments (VREs) have arisen as platforms for combining tools, databases, and procedures. However, many VREs currently lack built-in mechanisms for collecting and linking metadata information consistently across the ML pipeline, from data import to model evaluation.

This thesis attempts to fill this gap by developing a provenance-aware VRE framework that allows for structured, verifiable ML experimentation. It integrates JupyterHub, GitHub, and MLflow with a research data repository ecosystem (DBRepo and Invenio). This framework enables end-to-end documentation and examination of machine learning operations, with a primary focus on classification

workflows, in a transparent and standards-compliant manner.

1.2 Research Questions

The primary purpose of this thesis is to investigate how provenance mechanisms might be practically integrated into VREs to support data-driven, provenance-aware ML operations. This is led by the following research questions:

RQ1: Model and Data Provenance

Which methods for tracing and documenting the provenance of machine learning experiments capture metadata from data sources and code repositories with the highest degree of (a) automation, (b) alignment with provenance standards (e.g., W3C PROV-O, FAIR), and (c) completeness of metadata, based on evaluation within a reproducible benchmarking environment?

Sub-questions:

- **RQ1.1 — Data Provenance:** What percentage of required data provenance metadata (e.g., dataset origin, versioning, preprocessing steps) can be automatically or semi-automatically captured and structured using standards such as Croissant, FAIR, and PROV-O, by embedding workflows in Jupyter Notebooks and linking to DBRepo?
- **RQ1.2 — ML Model Provenance:** What percentage of machine learning model provenance metadata (e.g., training configuration, model version, evaluation metrics) can be automatically or semi-automatically recorded and aligned with standards like PROV-O, MLSEA, and FAIR4ML by monitoring training activities in Jupyter Notebooks and integrating with MLflow?

RQ2: Use of Metadata for Reproducibility and Analysis

What role can the collected metadata play in supporting provenance-related tasks such as auditing, debugging, reproducibility checks, and inspection of end-to-end

workflows?

Use-case questions include:

- **Provenance and Reproducibility:** How was a specific result (e.g., a metric or plot) produced?

  Which versions of data, code, parameters, and preprocessing steps were used?

  Can the exact experiment be reproduced solely based on the captured metadata?

- **Experiment Versioning and Error Tracing:** Which experiments were run on outdated or faulty data/code?

  Who was affected by these experiments, and how can they be notified?

- **Configuration and Evaluation Tracking:** What data splits, tuning parameters, or evaluation strategies were used in an experiment?

  How did these choices affect model performance e.g. precision, recall, F1-score?

- **Model–Data Relationship Mapping:** Which models were trained on which (versioned) datasets?

  How did each model perform across different datasets or configurations?

- **Repository Fork Awareness and Collaboration:** Are there outdated or diverging forks of code or datasets?

  Can collaborators affected by such forks be identified and alerted?

RQ3: Visualization and Interaction

How effectively can interactive visualization and analysis of provenance metadata support researchers in understanding, interpreting, and improving their machine learning workflows within a virtual research environment (VRE)?

Sub-questions:

- Which types of visual and analytical elements (e.g., traceability flows, model plots, segmented and structured visualization of metadata information) improve users ability to interpret and trace ML workflow metadata?

- Can provenance-based visualizations enable researchers to identify inefficiencies, trace decision points, and enhance reproducibility, as measured through task completion accuracy and user feedback?

RQ4: Standards and Metadata Mapping

What methods can be employed to convert collected provenance metadata into standardized schemas (JSON-LD, RDF/XML), and how can these improve interoperability and long-term usability across platforms?

1.3 Thesis Structure

This thesis is organized into six chapters and an appendix:

- **Chapter 1: Introduction** explains the rationale, research challenge, and important research questions for this thesis. It discusses the constraints of current ML workflow techniques and suggests a provenance-aware Virtual Research Environment.

- **Chapter 2: Related Work** investigates previous research on metadata standards (e.g., FAIR, PROV-O), Virtual Research Environments (VREs), and tools for ML provenance tracking.

- **Chapter 3: Methodology** describes the study design, which includes the Design Science study Methodology (DSRM), a user-centered assessment methodology, and a systematic approach to mapping metadata fields. It also emphasizes the purpose of each research question.

- **Chapter 4: System Architecture and Implementation** describes the proposed architecture, implementation method, and provenance capture tech-

nologies. It contains system diagrams, metadata pipelines, and framework-specific semantic mapping approaches.

- **Chapter 5: Evaluation and Visualization** describes the user research and empirical evaluation. It evaluates usability, metadata completeness, and repeatability utilizing important metrics, qualitative comments from test users, and visualizations relevant to these evaluations.

- **Chapter 6: Discussion and Conclusion** reflects on the findings, assesses strengths and limits, and proposes future options for increasing automation, visualization, and standardization in ML provenance systems.

- **Appendix** provides additional artifacts, such as system screenshots, metadata schema mappings, and utility scripts used during implementation and evaluation.

2 Related Work

2.1 Virtual Research Environments (VREs)

Virtual Research Environments (VREs) offer the necessary infrastructure to enable collaborative, repeatable, and data-intensive scientific operations. These platforms integrate processing, storage, and data analysis capabilities to make it easier to run and share experiments. However, its integration with machine learning (ML) processes is restricted, notably in terms of metadata, modularity, and provenance.

Galaxy Originally built for biomedical research, Galaxy [1] provides a browser-based interface for performing workflows. While it enables reproducibility through workflow versioning, it is largely focused on genomics and does not support general-purpose ML features like as model registries and feature provenances.

REANA The REANA platform [2] emphasizes reproducibility and reusability via containerization and workflow language support. It works seamlessly with CERN's architecture, tracking data via containers and storage levels. However, it lacks semantic-level information and integrations for ML-based experiment tracking.

JupyterHub [3] is widely used for notebook-based experimentation and supports shared computing environments. It offers freedom but does not enforce provenance or structured metadata. External tools such as MLflow are required for accurate experiment tracking.

iRODS The iRODS platform [4] provides rule-based data governance and metadata annotation, however it focuses more on data preservation than dynamic ML experimentation. Its value rests on backend assistance rather than pipeline execution.

Taverna and myExperiment These tools [5] [6] helped pioneer process sharing in early e-Science activities. Despite their historical importance, they lack current capabilities like as container support and deep machine learning integration.

COMET , Configurable, Observable, Modular Evaluation Tool [7] is a provenance-aware framework that interacts with Jupyter to automatically collect information using standards like PROV-O, FAIR, and Croissant. It attempts to decrease user burden while still enabling comprehensive metadata gathering and ML-native design. COMET is a standalone tool that aims to enable repeatable experimentation in machine learning contexts.

While our system has comparable aims, such as modularity and adherence with FAIR principles, it was created independently and focuses on lightweight setup and organized information output. Unlike COMET, it provides for more detailed customisation across several ML pipelines, notably in adapting provenance capture and export logic to individual research requirements and allowing flexibility to extend the functionality.

Limitations in VREs

- **Provenance granularity** is limited; many platforms don't capture transformation details.
- **Metadata standardization** is lacking, with formal formats such as W3C PROV-O, Croissant, and FAIR4ML being seldom used.
- **ML-centric features** like versioning and hyperparameter tracking are missing.

- **User adaptability** suffers due to high configuration and deployment overhead.

2.2 Metadata Management and Ontologies

Effective metadata improves repeatability and transparency in ML workflows. Despite the availability of established standards, most ML tools do not completely implement them.

W3C PROV-O and PROV-IO+: The W3C PROV [8] model captures provenance using entities, activities, and agents. PROV-IO+ [9] applies this to describing scientific data and parameters.

Croissant Croissant [10] is a lightweight JSON-LD schema aligned with Schema.org, aimed at FAIR datasets. It excels at static information but lacks pipeline dynamism, such as training configuration.

FAIR, FAIR4ML The FAIR rules [11] encourage open and reuse research metadata. FAIR4ML [12] applies this to ML settings, focusing on repeatability and auditability.

Limitations

- Poor integration with ML libraries
- Weak support for entire pipeline semantics
- Ontologies need a significant amount of learning
- Gaps in the description of modular pipelines

Our approach attempts to reduce these gaps by automatically or semi-automatically capturing metadata during execution.

2.3 Experiment Tracking and Provenance

MLflow MLflow [13] supports logging, artifact storage, and model versioning. However, it uses an internal format and lacks native support for semantic standards like PROV-O, FAIR, or others.

DVC Data Version Control [14] adds Git-style lineage tracking to data and models. It captures pipeline I/O but does not log training-specific metadata or semantic relations, which is important for data lineage.

Limitations

- Proprietary metadata formats
- High-level logging with minimal traceability
- Burdensome post-hoc metadata transformations

Our Contribution enhances MLflow by including semantic metadata collection across the ML pipeline using standards like as PROV-O, FAIR4ML, MLSEA, and Croissant. It adds structured semantic metadata to MLflow logs, such as justification logs, Git commit linkage, and dataset DOI mappings, to increase traceability and repeatability. Metadata is automatically exported in JSON-LD and RDF/XML, making it compatible with semantic web technologies. This is accomplished with minimum setup, minimizing cognitive strain on practitioners while enabling transparent, FAIR-compliant experimentation.

2.4 Usability and User Adaptability

Cognitive Load Many provenance technologies need sophisticated setup and ontology configuration, which increases the cognitive strain on researchers, particularly those who lack semantic web competence. This typically hinders adoption in actual machine learning environments [15].

Reuse Barriers Tools like DVC and MLflow promote modular pipelines, but still require users to navigate intricate directory structures and tracking server configurations. Our framework addresses this by offering:

- Lightweight selection interfaces.
- Modular, preconfigured templates.
- Minimal code rewrites for new datasets or models for comparable machine learning challenges, such as classification.

2.5 Summary

This chapter examined the environment of Virtual Research Environments (VREs), metadata standards, experiment tracking tools, and usability issues related to repeatable machine learning workflows.

Most existing VREs, such as Galaxy, REANA, and iRODS, provide generic infrastructure for data-intensive science but lack significant integration with machine learning processes, particularly in terms of modularity, semantic metadata collection, and provenance tracking. While solutions like MLflow and DVC provide for fundamental experiment tracking and version control, they do not natively support semantic standards like PROV-O, FAIR4ML, or Croissant, making downstream reuse and auditability challenging.

Ontologies such as W3C PROV-O, Croissant, and FAIR4ML lay a solid foundation for structured information, but they are frequently neglected because of their integration complexity and high learning curves. Previous attempts, such as COMET, have demonstrated that semantic logging within Jupyter-based systems is possible, but they may impose predefined processes or limit flexibility.

Our system bridges these gaps by integrating semantic metadata collection, lightweight setup, and automatic export to compatible formats. It supports fine-grained provenance, just-in-time user input, and modular flexibility, distinguishing itself as a versatile solution for FAIR-compliant ML exploration.

3 Methodology

This research project uses a multi-method approach that includes a Design Science Research Methodology (DSRM), a Systematic Literature Review (SLR), and structured metadata harmonization. The methodological basis provides both theoretical soundness and practical feasibility, assisting in the construction of a provenance-aware ML infrastructure that runs on notebook-based VREs. Jupyter, Streamlit, and MLflow were used alongside with GitHub data versioning and database repositories such as Invenio and DBRepo.

3.1 Design Science Research Methodology (DSRM)

DSRM, as stated by Hevner [16], facilitates iterative design and assessment of artifacts in information systems research. In this thesis, the artifact is a modular Virtual Research Environment that allows for metadata collection, version tracking, and semantic export inside ML workflow.

Problem Identification

Existing ML tools and VREs lack fine-grained, standards-compliant provenance capture. Metadata is spread among systems like JupyterHub, GitHub, DBRepo, and Invenio, which reduces transparency, reproducibility, and metadata compatibility.

Objectives of the Solution

The system must enable semi-automatic and uniform metadata collecting throughout the ML lifespan. This includes session metadata, dataset metadata through DOI resolution, Git-based version control, and alignment with semantic metadata standards such as FAIR [11], PROV-O [8], Croissant [10], MLSEA [17], and FAIR4ML [12].

Design and Development

The core system integrates:

- Metadata collection from JupyterHub, MLflow, GitHub, Invenio and DBRepo
- Global access to machine learning experiment metadata via a structured, queryable database on DBRepo.
- Ontology-based classification using FAIR, PROV-O, FAIR4ML, MLSEA, and Croissant and structuring one Json file.
- Streamlined dashboard interface designed to support intuitive user interaction and accelerate framework adaptability.
- Structured export to machine-readable forms (JSON, JSON-LD, and RDF/XML), the latter two of which are compatible with open-source tools and offer interactive querying via SPARQL.

Demonstration

Prototype execution occurs in Jupyter notebooks, where the system logs metadata during live ML experiments. Outputs are visualized via Streamlit in components such as:

- `Dataset Metadata` – Metadata fields extracted from DBRepo and DOI endpoints, including license, schema, and citation metadata.

- `Model Metadata` – Hyperparameters, metrics, and training details structured and grouped by ontology schemas such as PROV-O , FAIR4ML, and MLSEA.
- `Provenance Trace` – Visual links between dataset, model, and environment versions, including Git commit hashes and session metadata. Additionally, selected two runs are compared side by side, with differences highlighted between the two.
- `Error and Version Impact` – Highlights version drift and outdated runs by comparing metadata snapshots across executions. Includes an option to programmatically notify relevant collaborators by creating GitHub issues, with the notification details also captured in the metadata file.
- `Model-Dataset Mapping` – Visual metadata showing model reuse across datasets, aiding traceability.
- `Export Provenance` – Metadata export in machine-readable formats such as JSON-LD and RDF/XML, aligned to PROV-O standards.

Evaluation

Based on the four research questions presented in Chapter 1.2, the assessment of the suggested system is organized. Metadata quality, reproducibility, usability, and interoperability are the four main components of system performance that are reflected in the particular assessment criteria that are translated from each research question. Implementation fidelity and practical value are evaluated using these criteria.

Table 3.1 covers each research question in brief, along with the relevant criterion, evaluation technique, and section discussing the findings.

RQ	Evaluation Criterion	Method	Section
RQ1	Metadata completeness, alignment, and automation	Evaluate metadata from multiple runs against a reference schema; measure field presence, default usage, and degree of automation (manual vs. auto-captured fields).	5.2
RQ2	Reproducibility, traceability, debugging	Reproduce experiments using metadata alone; verify Git commit and dataset links; assess debugging via version comparison and dashboard tools.	5.3
RQ3	UI visualization effectiveness	Conduct structured user study with task execution and feedback collection; analyze responses from Likert ratings and open comments on visual clarity and usefulness.	5.4
RQ4	Metadata export compatibility and SPARQL access.	Test exports to JSON-LD and RDF/XML using open-source tools and run SPARQL queries to confirm accessibility.	5.5

Table 3.1: Summary of evaluation criteria and methods per research question

Communication

All findings, metadata structures, and tool usage are documented in this thesis. The Streamlit dashboard provides an interactive entry point for the users.

3.2 Systematic Literature Review (SLR)

A systematic literature review (SLR) was carried out to guide the metadata architecture, standards alignment, and system design of the proposed provenance-aware Virtual Research Environment (VRE). The review follows Kitchenham's approach [18], concentrating on metadata schemas, VRE architectures, and provenance tracking tools in machine learning repeatability.

Review Scope

- **Databases and Sources:** Sources included IEEE Xplore, SpringerLink, ACM Digital Library, arXiv, Zenodo, DataCite, myExperiment, and institutional repositories (e.g., TU Wien's *reposiTUm*, CERN Document Server). We prioritized both general frameworks (e.g., DBRepo [19]) and tools like ProTracker [20], PROV-IO+ [9], and Trrack [21].

- **Keywords and Search Terms:** “provenance logging”, “provenance tracking”, “ML reproducibility”, “semantic logging”, “XAI provenance”, “XAI machine learning pipeline” , “provenance data management”, “ provenance in machine learning”, “ontology-based logging”, “PROV-O”, “MLSEA”, “metadata mapping”, “data lineage”, “VRE provenance”.

- **Inclusion/Exclusion Criteria and Review Strategy:** Papers that merely deal with static information (e.g., for datasets) or focus entirely on domain-specific schemas (e.g., genomics, astronomy) without extending to machine learning or reproducibility infrastructure. We focused on literature from 2010-2024 to capture both foundational and emerging developments

Thematic Analysis and Literature Synthesis

Provenance Models: Foundational and Emerging Ontologies The underlying standard for structured and interoperable provenance modeling is the W3C PROV-O ontology [8]. Its abstraction frequently restricts its expressiveness in ML-specific circumstances, despite the fact that its entity-activity-agent pattern has been adopted in many fields. Hyperparameters, evaluation metrics, model explanations, and feature selection are examples of domain-specific elements used in machine learning workflows that PROV-O does not natively capture.

In order to overcome these constraints, recent publications have suggested alternatives and expansions. A systematic vocabulary for recording ML pipeline metadata, including feature selection and training parameters, is provided by MLSEA [17]. Croissant [10] and FAIR4ML [12] concentrate on the reuse and interoperability of dataset metadata. Despite the encouraging nature of these initiatives, acceptance is still dispersed because of their complexity, uneven tooling, and lack of native connection with ML lifecycle platforms.

Provenance Logging in ML Platforms Popular tools like MLflow are now essential for putting ML experiments into practice. Articles like "**Practical Deep Learning at Scale with MLflow**" [22] and "**Machine Learning Engineering with MLflow**" [23] explain how MLflow can serve models via registry and tracking

APIs and log parameters, metrics, models, and artifacts.

The metadata format used by MLflow, however, is proprietary and does not adhere to semantic standards. RDF, JSON-LD, and SPARQL are not supported; export is restricted to JSON. Metadata is not platform-interoperable, stays flat, and lacks global identifiers like ORCID and DOI. Because of this, MLflow has minimal usefulness for long-term reproducibility, cross-platform reuse, or interaction with metadata repositories (such as Invenio or DBRepo), despite its exceptional usability. Practical evaluations consistently found these gaps [9, 24].

Scientific and Semantic Provenance Frameworks The goal of a number of research-grade systems is to semantically coherently capture provenance. PROV-IO+ [9], for instance, expands PROV-O by providing HPC-specific support for RDF export and structured metadata collecting across platforms. By adding domain-specific semantics for workflow-level provenance and fine-grained execution tracing, tools like ProTracker [20], ProvONE [25], and PAV [26] expand upon PROV-O. PAV is especially pertinent to machine learning systems that incorporate code, model, and dataset development since it places an emphasis on authorship, versioning, and citation metadata. ProvONE focuses on data dependencies and workflow organization in scientific computing.

However, real-time, iterative machine learning development is not directly supported by these tools, which are primarily made for static, batch-oriented scientific operations. In particular, not many provide connection with notebook-based environments (like Jupyter) or capture technologies that are often used in modern machine learning pipelines, such as GitHub, MLflow, or Streamlit.

Virtual Research Environments (VREs): Provenance-Aware but ML-Limited VRE platforms such as Galaxy [1], REANA [2], and iRODS [4] promote reproducibility through containerization, execution encapsulation, and data lineage tracking. However, a number of machine learning-specific needs, such as hyper-parameter monitoring, model or dataset versioning, architecture-level logging, and semantic export of results in line with ML ontologies, are usually not covered by these platforms.

The provenance-aware framework COMET [7] is used in combination with Jupyter to gather metadata in accordance with PROV-O, FAIR, and Croissant standards. Nevertheless, automatic export to compatible semantic forms such as RDF/XML or JSON-LD utilizing FAIR4ML or PROV-O vocabularies is not supported by COMET. Additionally, it provides little flexibility for adjusting export logic across several ML pipelines or changing provenance capture. Users are unable to modify the granularity of collected provenance or define custom information fields, for instance.

The suggested solution, on the other hand, allows for customizable information capture at many pipeline phases, such as justification logging, model design, dataset import, session startup, and Git snapshotting. Custom metadata schemas and mappings may be defined by users using editable JSON structures, and results can be exported in RDF/XML and JSON-LD formats that are compatible with FAIR4ML, MLSEA, PROV-O, and Croissant. For enhanced provenance tracking, it also facilitates direct interaction with collaborative platforms like GitHub and metadata repositories like DBRepo [19].

Visualization and Debugging: Gaps in Traceability Many research investigations that highlight the significance of provenance-based explainability may be found in the literature that is currently available. Lineage tracking is used to assist model interpretability in **"Provenance-based Explanations for ML Models"** [27]. **PipelineProfiler** [28] and similar tools, however, concentrate on debugging at the pipeline level using interactive visualizations.

These systems, however, hardly ever offer automatic metadata export or conform to semantic standards. Therefore, even while they enhance interpretability and usability, they don't help create standard-compliant metadata pipelines, which are crucial for open science and reproducibility checks.

Conclusion

Across the surveyed literature, clear limitations persist:

- The majority of machine learning technologies only record metadata in tool-specific, ad hoc forms; they don't offer ontology-based export or provenance standard integration.
- Semantic frameworks offer powerful expressiveness but lack tooling support or usability for everyday ML tasks.
- Although VRE systems make replication easier, they are not tailored to the unique requirements of notebook-centric machine learning experiments.
- Despite growing calls for linked, FAIR-aligned data exchange, exporting to machine-readable formats (RDF, JSON-LD) is still uncommon.

To address these enduring gaps and allow ML practitioners to gather, view, and export provenance-rich metadata without sacrificing usability, an integrated framework is required. By integrating standards-compliant export, ontology-based organization, and live metadata collection inside a modular VRE, this study offers one such option. The framework attempts to bridge the gap between the needs of semantic interoperability and reproducibility and real-world machine learning tools.

3.3 Metadata Mapping and Harmonization

Systematic Mapping Studies in Software Engineering [29] served as the inspiration for our implementation of a systematic mapping process to integrate heterogeneous metadata across technologies such as DBRepo, MLflow, GitHub, and Jupyter environments. This procedure ensures semantic enrichment and interoperability, both of which are essential for repeatability in machine learning workflows [30, 31].

Field Extraction and Categorization

Code execution logs, Git commits, dataset schemas, model configuration files, and session environments were among the several sources from which metadata was

automatically retrieved. These were categorized into five semantic groups that corresponded to the various phases of the machine learning lifecycle:

- **Descriptive Dataset Metadata (FAIR):** Persistent IDs and rich, standards-based fields (DCTERMS/DCAT) for Findability (F1–F2), Accessibility (A1 via landing/download URLs), Interoperability (I1 via shared vocabularies), and Reusability (R1.1 license, R1.2 provenance); refined for this project's scope.
- **Experiment Metadata (FAIR4ML):** Records information about the script environment, target variables, training duration, model type, and run IDs.
- **Model Architecture and Hyperparameters (Croissant):** Explains input-output mappings, serialization formats, architectures, and learning techniques.
- **Performance Metrics and Justification (MLSEA):** Encompasses training/test splits, ROC-AUC, accuracy, F1, and reasoning areas that explain model behavior. For this work, elements from MLS and MLSO are treated as integral parts of MLSEA, as both share a common conceptual foundation within the same overarching schema.
- **Workflow Provenance (PROV-O):** Keeps track of training agents, interactions, and entities, including connections to code, datasets, and system environments.

Ontology Mapping Approach

Metadata from six pipeline sources (dataset, model, justification, experiment, session, Git) was consolidated into five semantic standards: **FAIR**, **FAIR4ML**, **MLSEA** (umbrella for MLS/MLSO), **Croissant**, and **PROV-O**. Each block uses canonical prefixes (`dcterms:`, `fair4ml:`, `mlso:`, `mls:`, `croissant:`, `prov:`) to retain semantic clarity.

Representative mappings include:

- **FAIR**: Dataset identifiers, titles, licenses, landing pages (`dcterms:identifier`, `dcat:landingPage`).
- **FAIR4ML**: Run/session IDs, training times, script commit, environment (`fair4ml:modelCategory`, `fair4ml:trainedOn`).
- **MLSEA (MLS/MLSO)**: Metrics and model/data relations (`mlso:Dataset`, `mls:Feature`, `mlso:HyperParameter`).
- **Croissant**: Schema artefacts, example records, file paths (`croissant:examples`, `croissant:isLiveDataset`).
- **PROV-O**: Entity-activity-agent links (`prov:wasGeneratedBy`, `prov:End`).

Missing values are replaced with explicit placeholders (e.g., `"info not available"`) to keep schema completeness.

Conflict Resolution Strategy

In this context, conflict resolution refers to handling situations where the same conceptual field appears in multiple metadata sources, with differences in value, format, or placement across ontologies. These conflicts arise naturally from combining diverse capture points in the pipeline-dataset metadata, MLflow logs, justification inputs, model configurations, session descriptors, and Git commit records.

Conflicts are resolved according to the following principles:

1. **Source precedence:** For each field, the most authoritative source is selected. For example, `mlso:hasAlgorithmType` is taken from the model block (reflecting the actual algorithm used) and only falls back to experiment logs if missing; `dcterms:identifier` is taken from dataset metadata, never inferred from file names.
2. **Temporal consistency:** Where timestamps differ across sources, the value consistent with the run's recorded start/end times and commit timestamp is retained. For instance, `prov:Start` and `prov:End` are aligned with

`prov:generated AtTime` to ensure the recorded metrics belong to the same code state.

3. **Semantic de-duplication across ontologies:** The most expressive ontology term is considered canonical, but important fields may be duplicated across *different* ontologies to maintain completeness of each ontology's view. For example, the dataset reference appears in both `mls:Dataset` (model/data relationship) and `fair4ml:evaluatedOn` (experimental context).

4. **Safe defaults:** When values are missing or ambiguous, explicit placeholders (e.g., `"info not available"`) or stable constants are used. For instance, `dcterms:license` defaults to this placeholder if not provided in the dataset DOI metadata.

5. **Integration of extra metadata:** Additional fields beyond strict ontology requirements are mapped to improve reproducibility and interpretability, consistent with prior work advocating richer documentation for responsible ML use [32, 33, 34]. Such *reporting extras* (e.g., `prov:usedNotebook`, `fair4ml:intended Use`) support real-world use cases demonstrated in RQ2, even when they are not mandatory for coverage scoring.

6. **Trade-off in coverage metrics:** Duplication across ontologies is permitted when it contributes to the completeness of a given ontology, but redundant mappings are generally avoided within the ontology. This selective approach may slightly lower the overall "unique field" coverage percentage compared to forcing full duplication, but it also reduces unnecessary redundancy while keeping each ontology self-contained where it matters. For example, `mls:Software` is not included in the mapping because `mlso:hasRelatedSoftware` already captures the necessary information for our thesis scope.

This approach balances strict ontology alignment with practical considerations of reproducibility, interpretability, and downstream usability. By embedding these rules in the export function, every run produces a consistent, ontology-aligned JSON file that is both semantically valid and operationally complete. The complete list can be found in Section **??**

Semantic Export and Interoperability

The harmonized metadata is stored in a structured JSON file and exported to two semantic formats:

- `JSON-LD`: Used for lightweight semantic linking with schema.org and W3C ontologies, supporting visual inspection and integration with metadata dashboards [35].
- `RDF/XML`: Designed for SPARQL query engines and institutional repositories; enables deep graph-based provenance tracking [36].

`rdflib` is used to construct export logic, which converts metadata fields into RDF triples. In accordance with W3C PROV-O, every run attempts to create a fully connected knowledge graph that links datasets, agents, actions, and model outputs.

3.4 Summary

This chapter outlined the multi-method approach underpinning this thesis, combining Design Science Research Methodology (DSRM), a Systematic Literature Review (SLR), and structured metadata mapping and harmonization. DSRM guided the iterative design and evaluation of a notebook-native, provenance-aware Virtual Research Environment (VRE), while the SLR identified persistent gaps in ML tooling and standards-informing ontology choices (FAIR, FAIR4ML, MLSEA, Croissant, PROV-O) and the need for selective integration strategies.

The metadata harmonization process unified heterogeneous sources (DBRepo, MLflow, GitHub, Jupyter) into a single, ontology-aligned JSON structure with conflict resolution rules, explicit placeholders, and targeted "extra metadata" to enhance the quality of the metadata. Duplication across ontologies was applied selectively to ensure completeness of individual ontologies without unnecessary redundancy, a trade-off that preserves coverage where it matters most.

By embedding these mappings into an export pipeline for RDF/XML and JSON-LD, the system enables standards-compliant interoperability, reproducibility, and

interpretability-demonstrated in real-world use cases in RQ2–RQ4. This methodological synthesis bridges the usability of popular ML tools with the semantic rigor of provenance standards, producing a framework that is both practical for practitioners and aligned with open science principles.

4 System Architecture and Implementation

4.1 System Design Principles

Key flaws found in the Systematic Literature Review (see Section 3.2) are addressed in the design of this provenance-aware Virtual Research Environment (VRE). *automated metadata capture*, *semantic interoperability*, and *support for reproducible ML workflows* are its three guiding concepts.

The solution integrates information collection directly into standard machine learning operations,including dataset selection, model training, and assessment, with a focus on notebook-centric workflows. This ensures that provenance is gathered with the least amount of user work [37].

Captured metadata is produced in machine-readable forms and mapped to common ontologies (such as PROV-O, FAIR4ML, MLSEA, and Croissant) to ensure compatibility. A lightweight, modular system that enhances repeatability and traceability.

Design Goals

In order to include provenance capture, justification recording [27], and semantic export into their current processes with the least amount of manual effort. The system architecture was shaped by the following objectives:

- **Metadata Automation:** It is preferable to collect metadata passively or with simple user cues. Only ambiguous fields or justifications should require

manual entry. The integration of metadata logging through MLflow can be found as the downloadable file in Zenodo

- **Semantic Interoperability:** All metadata, such as PROV-O [8], FAIR4ML [12], MLSEA [17], FAIR [11], and Croissant [10], should be in line with semantic web standards and community ontologies. Machine readability, export compatibility, and smooth interaction across repositories, dashboards, and provenance systems.

- **Minimal Intrusion:** The system must not interfere with the Jupyter user's workflow. Python scripts and ML lifecycle tools (like MLflow and Git) ought to be seamlessly integrated with it.

- **Reproducibility Support:** Every experiment has to have a unique identity, be preserved in organized metadata snapshots, and be tied to dataset DOIs, Git versions, and hyperparameter settings.

- **Visual Transparency:** A dashboard should allow users to examine and verify information. It should be possible to view justifications, version drifts, and traceability pathways.

4.1.1 Rationale for VRE Component Selection

The virtual research environment was deliberately composed of DBRepo, Invenio, Jupyter Notebook, Git, and MLflow to balance practical usability with provenance, reproducibility, and interoperability requirements. DBRepo provides a structured, queryable source for dataset storage and retrieval, enabling metadata enrichment through DOI integration. Invenio serves as the archival and dissemination layer, assigning persistent identifiers and ensuring long-term accessibility. Jupyter Notebook offers an interactive, notebook-native execution environment that aligns with common ML workflows while enabling fine-grained, cell-level logging [38]. Git captures version-controlled source code and experiment lineage, while MLflow orchestrates automated logging of parameters, metrics, and artifacts across the pipeline. Together, these components form a lightweight yet extensible VRE that integrates seamlessly with semantic export workflows while minimising user friction. Any of

these components could be replaced with equivalent technologies, as the framework aims to simulate the structure and interoperability principles of a real-world VRE rather than enforce specific tool dependencies.

High-Level System Design Overview

Fundamentally, the system architecture is modular in nature and covers every stage of the machine learning lifecycle, from dataset selection to metadata ingestion and archiving. Figure 4.1 shows this multi-layered configuration:

- **Metadata Ingestion:** Automatically fetches structured metadata (title, DOI, license, schema version) from DBRepo APIs.
- **Execution Logger:** Uses the MLflow library in Jupyter notebooks to record runtime information, including model type, hyperparameters, total execution time, preprocessing time, and evaluation metrics. Without needing user input, the preprocessing duration is automatically determined by recording timestamps at the start and end of the `preprocess_and_log()` function. In a similar manner, the whole time utilized for the experiment is recorded from the beginning to the end.

  Our system uses internal MLflow tags to differentiate between the total experiment execution and the training execution, even though PROV-O abstracts all phases as `prov:Activity` nodes. We scope these durations semantically: `prov:Start`, `prov:End`, and `prov:generatedAtTime` capture the overall experiment execution window, while `prov:startedAtTime` and `prov:endedAtTime` specifically record the training phase. This separation enables more precise provenance queries and performance analysis by isolating training time from the broader experimental process.

  The system is flexible to accommodate more complex workflows; users may add more MLflow tags to arrange stages as custom-defined phases or annotate enhanced processing steps. These can subsequently be incorporated into the export pipeline after being mapped to the appropriate ontology terms. This design enables future development toward flexible, user-defined provenance

blocks that mirror the structure of real-world analytics pipelines, even if the current system records preset steps.

- **Version Tracker:** Anchoring provenance in replicable code snapshots, the experiment is bound to Git commit hashes and author IDs.
- **Semantic Mapping Engine:** Maps raw metadata fields to ontological terms, resolving conflicts and enforcing schema consistency.
- **Visualization and Export:** Renders interactive metadata views in a Streamlit dashboard and supports export in `JSON-LD` and `RDF/XML` for integration with linked data ecosystems. Model evaluation plots (e.g., ROC curves, feature importance charts, confusion matrices) are loaded into the UI from a local storage generated at runtime. These files remain available for inspection during the current session; long-term preservation is handled by the archival layer.
- **Archival Layer:** Ensures long-term persistence of all provenance artifacts in Invenio, including structured metadata snapshots, semantic exports (`.jsonld`, `.rdf.xml`), trained model files (`.pkl`), and an archived copy of the generated plots. This ensures that all visual and non-visual artifacts displayed in the dashboard remain accessible for future reproducibility and auditing.

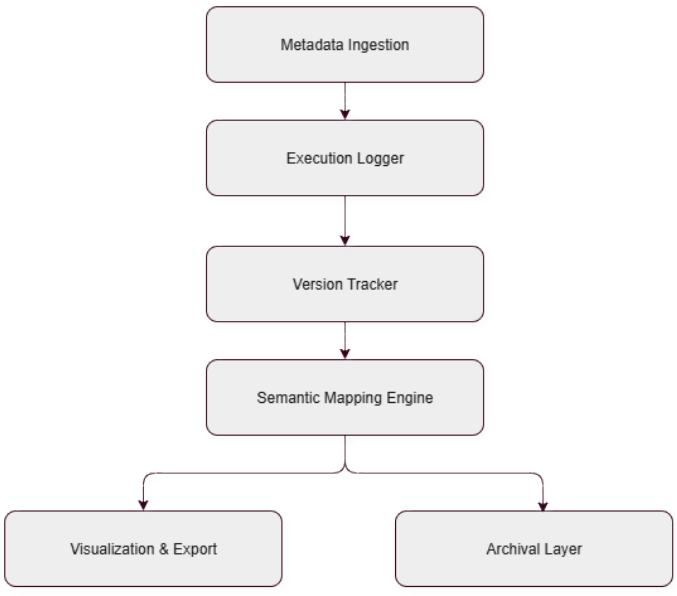


Figure 4.1: High Level System Design Overview

Alignment with semantic metadata standards, maintainability, and expansion are made easier by this modular architecture. Later on in Section 4.5, a user-facing interface for working with the acquired metadata is shown. The organized dashboard enables users to examine, filter, and analyze experiment metadata without having direct access to the underlying logs.

4.2 Architecture and Workflow

System-Level Component Roles and Flow

Interoperating modules including data access, experiment execution, provenance logging, semantic mapping, visualization, and archive data make up the system architecture Figure 4.2. When combined, these elements are intended to facilitate semantic conformity and reproducibility across the notebook-native machine learning process. The Evaluation chapter (see Section 5) shows how the system can reproduce execution traces from the collected metadata.

- **DBRepo:** A FAIR-aligned metadata repository that serves two roles: (1) providing dataset-level metadata that aligns with DataCite, and (2) acting as the final archival backend for structured metadata. Upon user input of a DOI, the system fetches metadata fields such as `title`, `creator`, `license`, and dataset statistics through DBRepo's API. This metadata is injected into the active MLflow context for the experiment and forms the foundation for further semantic alignment. After the run, the MLflow metadata (including execution and model parameters) is parsed into relational tables, and inserted into a dedicated DBRepo database named Provenance Database. This allows both persistent archival and global access.

- **JupyterHub:** Hosts the core computational environment where the entire machine learning workflow is executed as modular, interactive notebooks [3]. In addition to providing user-specific sessions, it acts as the runtime environment for starting experiment runs, prepping datasets, training models, and recording session information. The framework incorporates user prompts, ex-

ecution logging, and environment introspection (such as operating system, Python version, and user role) right into Jupyter notebooks. Every run gathers provenance-aware metadata from the ML code and environment, which is then sent to downstream modules for archival and semantic mapping.

- **MLflow Logger:** Serves as the main execution tracker, continuously logging run-specific information, environment variables, training metrics, and model parameters. Reproducibility without external dependencies is made possible by the framework's establishment of a local SQLite-based MLflow tracking backend. Every experiment logs FAIR-aligned metadata (e.g., dataset DOI, hyperparameters, accuracy scores) and execution-specific tags (e.g., model architecture, test size, random state). The metadata for snapshots is saved as an MLflow artifact and exported as structured JSON. To ensure semantic completeness for downstream export and visualization, manual tagging is paired with autologging using `mlflow.sklearn`.

- **GitHub:** Git metadata, such as commit hash, author, email, date, message, and branch, are extracted from the working repository to enable fine-grained code-level provenance. To provide robustness across environments, this metadata is programmatically collected using the `subprocess` fallbacks and the GitPython API. In order to link experiment metadata to the precise code snapshot and facilitate debugging and semantic repeatability, Git commit IDs are stored as MLflow tags. Every MLflow run may be linked to a particular codebase state thanks to this version-binding step.

- **Semantic Mapper:** Acts as the harmonization engine that synchronizes gathered metadata with recognized semantic web standards. A fully qualified ontology URI (e.g., `prov:startedAtTime`, `mlso:hasAlgorithmType`, `fair4ml: trainedOn`) is mapped to each metadata entry, maintaining semantic precision. These standardized vocabularies are a systematic translation of internal MLflow tags like `trainingStartTime`, `learningAlgorithm`, and `input`. This mapping step resolves name conflicts or structural discrepancies.

- **Invenio Repository:** Serves as the last archive destination for metadata ex-

ports with semantic enrichment. The system serializes RDF/XML and JSON-LD files after aligning metadata with ontologies, and then uses a structured HTTP-based commit mechanism to submit the files to Invenio. Although Invenio metadata, including publishing details and persistent IDs, is retrieved through its API and added to the structured JSON for completeness, it is neither shown in the dashboard nor queried in real time. This clean decoupling between provenance capture and long-term metadata registration reflects a design choice for the current implementation, and can be adapted if tighter integration is desired.

- **Streamlit Dashboard:** Acts as the layer for interactive experiment metadata viewing and examination. The `MODEL_PROVENANCE/` directory(local storage) contains locally saved JSON files from which it retrieves structured metadata (onthology aligned). Users may examine preprocessing stages, provenance trails, model setups, performance metrics, dataset metadata, and user justifications through the dashboard. Transparency, repeatability, and post-hoc analysis are supported by visual components such as accuracy graphs, and git-based version impact notifications.

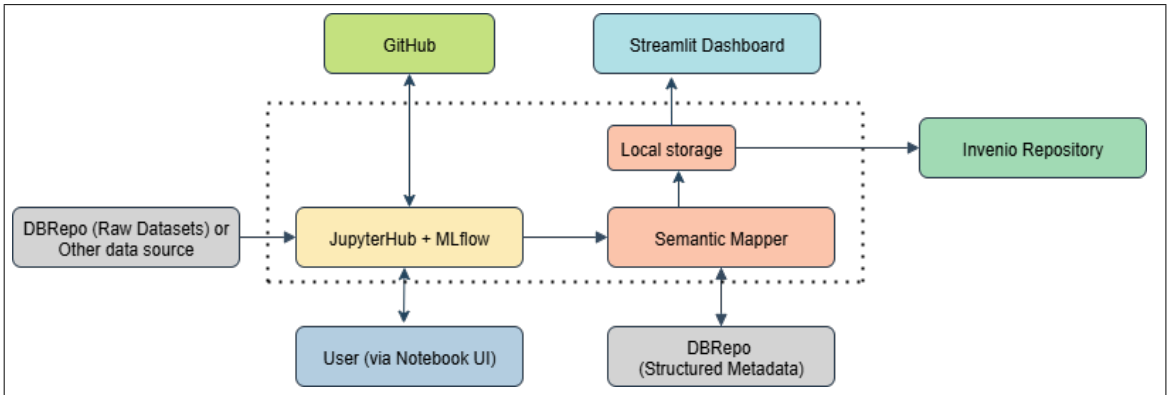


Figure 4.2: System Architecture

Figure 4.2 shows that the range of flexible input sources could be used as a data source, including DBRepo or other structured datasets. Every activity is started using the JupyterHub notebook interface, and runtime metadata is captured using MLflow. Initially, this metadata is saved in DBRepo after being processed into structured tables. The Semantic Mapper then retrieves it and harmonizes it into a

single structured JSON file that is aligned with the ontology, which we refer in this thesis as **structured metadata JSON** file. The Streamlit Dashboard retrieves this file from local storage as its unified source, and it also generates machine-readable outputs in `JSON-LD` and `RDF/XML`. The Invenio repository contains the archived versions of these export files, structured JSON file, plots, .pkl of the model. Additional metadata (such as the ID, Title, and the timestamp of ingestion) is returned by Invenio and appended to the structured JSON for completeness. Nevertheless, neither the produced semantic files nor the Streamlit dashboard display this Invenio-sourced metadata.

Data and Metadata Flow

The metadata flow in Figure 4.3 is explained in more detail in this section. A Jupyter notebook is instantiated from a Git-tracked repository to start the experiment. The session-level metadata is automatically logged such as **session id**, **session start time**, with optional semi-automatic metadata such as **user role** and **project name** during the initial setup.

The user then chooses a dataset from a selection of choices that have already been loaded and retrieved from DBRepo via API calls. The dataset-level related metadata is imported in the JupyterHub and MLflow incorporates the obtained metadata as tags for consistent tracking, including fields like `title`, `creator`, `license`, and basic statistics.

MLflow automatically logs metadata, including as model settings, evaluation metrics, and execution timestamps, as the notebook runs. A subset of hyperparameters are given as part of the user-defined experiment configuration block.

The Semantic Mapper maps internal fields to ontology-aligned terms from `PROV-O`, `FAIR4ML`, `FAIRMLSEA`, and `Croissant`, harmonizing all collected metadata, including Git, DBRepo, user input, and MLflow logs. The structured JSON that is produced is exported in RDF/XML and JSON-LD forms, and it powers the interactive local visualization feature of the Streamlit Dashboard.

Figure 4.3: End-to-end data and metadata flow through the system.

4.3 Metadata Field Capture and Semantic Mapping

The acquisition, harmonization, and semantic alignment of metadata across the machine learning experimentation process are described in this section. MLflow logging and user prompts are used to gather metadata at first, after which it is normalized and saved in DBRepo utilizing structured tables Appendix B.1. A mapping engine that aligns each field to ideas from well-known ontologies like PROV-O, FAIR4ML, FAIR, MLSEA, and Croissant is then used to unify these entries. To facilitate replication and reuse later on, the final structured metadata is exported in JSON-LD and RDF-XML forms.

There are three components involved in metadata capture:

- **Automated fields** are populated during pipeline execution via MLflow, Git hooks, and system logging (e.g., timestamps, model scores).
- **Semi-automated fields** are injected through user prompts (e.g., target variable, train-test split, hyperparameters) and chosen from the options provided, but not limited to just that.

- **Manual fields** are selectively defined by the user (e.g., experiment descriptions, intended use, ethical considerations).

To comply with ontology terms, these fields undergo a renaming and alignment procedure after they are gathered. Depending on its conceptual purpose, each field is divided into one or more semantic domains.

Field Renaming and Ontology Mapping Logic

Before serialization, internal keys are rewritten to CURIEs[1] `PROV-O`, `FAIR`/`dcterms`/`dcat`, `FAIR4ML`, `MLSEA` via `mls`/`mlso`, `Croissant`) using a domain-specific mapping dictionary. Matching is case-sensitive.

- **PROV-O** - Chosen for its broad adoption in representing provenance, enabling detailed tracking of activities, agents, and entities.
- **FAIR / DCTERMS / DCAT** - FAIR is principle-driven; DCTERMS/DCAT terms were used to express these principles and ensure repository interoperability.
- **FAIR4ML** - Selected for its ML-specific extension of FAIR principles, aligning experiment metadata with machine-readable reproducibility standards.
- **MLSEA (MLS/MLSO)** - Adopted for its domain-rich ML lifecycle vocabulary, allowing fine-grained capture of metrics, hyperparameters, and evaluation details.
- **Croissant** - Included to demonstrate compatibility with MLCommons' emerging dataset packaging standard for discoverability and reuse.

Purpose and Benefits:

- Ensures consistent naming conventions across all semantic export formats.

[1] CURIEs (Compact URIs) are shortened identifiers that use a prefix (e.g., `dcterms`, `prov`, `mls`) mapped to a full namespace URI. For example, instead of writing `http://purl.org/dc/terms/license`, the compact form `dcterms:license` is used. This improves readability while ensuring unambiguous links to ontology terms.

- Valid RDF serialization is made possible by ontology-compliant vocabulary.
- Enhances the interpretability of exported metadata for both humans and machines.
- Enables interaction with LOD systems and downstream querying using SPARQL.

The script excerpt of renaming logic is shown in Listing 1, and the full mapping dictionary is detailed in Zenodo.

Listing 1: Excerpt of Python script for ontology alignment

```python
def generate_structured_metadata_subset(data: dict) -> dict:
    ds      = data.get("dataset_metadata", {}) or {}
    model   = data.get("model_metadata", {}) or {}
    just    = data.get("justification_metadata", {}) or {}
    exp     = data.get("experiment_metadata", {}) or {}
    session = data.get("session_metadata", {}) or {}
    git     = data.get("git_metadata", {}) or {}

    FAIR = {
        "dcterms:identifier":  ds.get("dataset_id"),
        "dcterms:title":       ds.get("title"),
        "dcterms:license":     ds.get("license"),
        "dcterms:language":    ds.get("language"),
        "dcat:landingPage":    ds.get("source_url"),
    }

    FAIR4ML = {
        "fair4ml:experimentID": exp.get("experiment_id"),
        "fair4ml:runID":        exp.get("runid"),
        "fair4ml:modelID":      exp.get("modelid"),
        "fair4ml:trainingStartTime": exp.get("training_start_time"),
        "fair4ml:targetVariable":    model.get("target_var"),
    }

    MLSEA = {
        "mlso:f1Score":      just.get("f1_macro"),
        "mlso:trainSplit":   model.get("train_split"),
        "mlso:testSplit":    model.get("test_split"),
        "mlso:hasAlgorithmType": model.get("algo"),
    }

    Croissant = {
        "croissant:citeAs":   ds.get("citations_over_time"),
        "croissant:source":   ds.get("doi") or ds.get("source"),
        "croissant:field":    model.get("features"),
        "croissant:fileObject": model.get("model_path"),
    }

    PROVO = {
        "prov:startedAtTime": exp.get("training_start_time"),
        "prov:endedAtTime":   exp.get("training_end_time"),
        "prov:commit":        git.get("commit_hash"),
        "prov:usedNotebook":  exp.get("source_notebook"),
        "prov:wasGeneratedBy": exp.get("runid"),
    }
```

```
return {
    "FAIR": FAIR,
    "FAIR4ML": FAIR4ML,
    "MLSEA": MLSEA,
    "Croissant": Croissant,
    "PROV-O": PROVO
}
```

Selection of Mapped Fields

The metadata fields included in the ontology mapping logic were chosen based on three criteria: (1) their applicability to the main assessment goals of the framework (automation, repeatability, and metadata completeness within scope of ML classification tasks); (2) the infrastructure's ability to capture them during real experiment runs; and (3) the presence of semantically meaningful counterparts in one or more supported ontologies. The system concentrates on fields that are both **practically collectible** and **semantically interoperable**, as opposed to striving for comprehensive field coverage.

Excluded Metadata Fields and Rationale

The metadata fields excluded in the ontology mapping logic were chosen based on three criteria: Fields that were (a) outside the thesis scope-given the exclusive focus on simple ML classification tasks, making metadata related to other workflows such as regression or deep learning unnecessary (b) lacking substantial alignment with the targeted ontology concepts (c) if the metadata is already part of the ontology, mapped to a similar term. Representative fields removed due to their low relevance, limited utility, or semantic mismatch are shown in Figure 4.1.

Table 4.1: Subset of ontology metadata fields excluded.

Metadata Standard	Field Not Included	Justification
FAIR	dcterms:conformsTo	Standard compliance reference; not relevant to classification experiments.
FAIR	dcterms:coverage	Geographic/temporal coverage not applicable to current datasets.
FAIR4ML	fair4ml:modelHyperparamsPath	Hyperparameters logged inline, not as an external file.
FAIR4ML	fair4ml:inferenceLatency	No real-time inference measured in experiments.
Croissant	croissant:regex	Low-level data parsing detail; not part of classification task metadata.
Croissant	croissant:jsonPath	Field path extraction detail; unnecessary for current dataset structure.
Croissant	croissant:parentField	Hierarchical field reference not used in the datasets under study.
MLsea	mls:Quality	General quality descriptor; not assessed in current classification experiments.
MLsea	mls:Software	Generic software descriptor; covered by similar term.
MLsea	mlso:hasVariant	No variants used in current scope.
MLsea	mlso:hasDataLoaderLocation	Data loader location not tracked in current framework.
MLsea	mlso:relatedToField	Domain relationship descriptor not used in experiments.
MLsea	mlso:hasPredictionsLocation	Prediction storage location not applicable to present setup.
PROV-O	prov:Delegation	No delegation of activities recorded in current workflow.
PROV-O	prov:alternateOf	No alternative entities generated during experiments.
PROV-O	prov:qualifiedForm	Qualified form description not used in the workflow representation.

Complete Field Table

A comprehensive list of all metadata fields that the system has recorded and exported is provided in the following tables, arranged according to how they were populated. Fields can be captured manually, semi-automatically, or automatically, depending on how much system involvement and human input are needed throughout the experiment process.

The field name, its mapped ontology term or terms, the target ontology or ontologies, and the precise source from which the value was derived are all included in each entry. Git repositories, MLflow logs, system-level environment variables, DOI metadata records, DBRepo entries, and user-provided explanations are some examples of these sources.

Some fields appear in more than one ontology, such `runID`, `targetVariable`, or `modelID`. To maintain consistent cross-ontology alignment and prevent redundancy in these situations, the export logic employs scoped naming and prioritizing rules (see Section 4.3).

Table 4.2: Automatically captured metadata fields

Field	Ontology Terms	Ontologies	Capture Sources
Task*	mls:Task, mlso:hasTaskType	MLSEA	MLFlow
EvaluationSpecification*	mls:EvaluationSpecification, mls:realizes	MLSEA	MLFlow
experimentID*	fair4ml:experimentID, mls:Experiment, mlso:hasRelatedImplementation, mlso:justificationClassWeight, mlso:justificationCriterion, mlso:justificationMaxDepth, mlso:justificationMaxFeatures, mlso:justificationMinSamplesLeaf, mlso:justificationMinSamplesSplit, mlso:justificationNEstimators, mlso:justificationNJobs	FAIR4ML, MLSEA	MLFlow
field*	croissant:field, mls:featureList, mls:hasPart	Croissant, MLSEA	MLFlow
fileObject*	croissant:fileObject, mls:modelPath, prov:generated	Croissant, MLSEA, PROV-O	MLFlow
targetVariable*	fair4ml:targetVariable, mls:labelEncoding, mls:targetVariable, mlso:hasDefaultTargetFeature	FAIR4ML, MLSEA	MLFlow
fineTunedFrom*	fair4ml:fineTunedFrom, mls:HyperParameter, mls:HyperParameterSetting, mls:Implementation, mls:hasHyperParameter, mls:specifiedBy, prov:EntityInfluence, prov:Influence	FAIR4ML, MLSEA, PROV-O	MLFlow
source*	croissant:source, dcat:keyword, dcterms:contributor, dcterms:license, dcterms:source, dcterms:subject	Croissant, FAIR	DOI
sharedBy*	fair4ml:sharedBy, prov:Association, prov:Role, prov:hadRole	FAIR4ML, PROV-O	MLFlow
sessionID*	fair4ml:sessionID, prov:sessionID	FAIR4ML, PROV-O	MLFlow
serializationFormat*	mls:serializationFormat, mlso:hasFormat	MLSEA	MLFlow
EvaluationMeasure*	mls:EvaluationMeasure, mls:ModelEvaluation	MLSEA	MLFlow

Continued on next page

Table 4.2: Automatically captured metadata fields

Field	Ontology Terms	Ontologies	Capture Sources
runID*	fair4ml:runID, mls:Run, prov:Activity, prov:Generation, prov:activity, prov:hadActivity, prov:wasGeneratedBy, rdfs:label	FAIR4ML, MLSEA, PROV-O	MLFlow
hasRelatedSoftware*	mlso:hasRelatedSoftware, prov:pythonVersion	MLSEA, PROV-O	MLFlow
hasTrainTestSplitIndices*	mlso:hasTrainTestSplitIndices, mlso:trainSplit	MLSEA	MLFlow
isLiveDataset*	croissant:isLiveDataset, croissant:repeated	Croissant	MLFlow
rowCountEnd*	mlso:rowCountEnd, mlso:rowCountStart	MLSEA	MLFlow
registered*	dcat:registered, dcterms:available	FAIR	DOI
references*	croissant:references, dcat:relatedResource, dcterms:relation	Croissant, FAIR	DOI
Location*	prov:Location, prov:atLocation	PROV-O	MLFlow
MLModel*	fair4ml:MLModel, fair4ml:mlTask, fair4ml:modelCategory	FAIR4ML	MLFlow
modelID*	fair4ml:modelID, mls:Model	FAIR4ML, MLSEA	MLFlow
numDeletedRows*	mlso:numDeletedRows, mlso:numInsertedRows	MLSEA	MLFlow
hasQuality*	mls:hasQuality, mlso:accuracy	MLSEA	MLFlow
ethicalLegalSocial*	fair4ml:ethicalLegalSocial, fair4ml:intendedUse, fair4ml:modelRisks, mlso:justificationDatasetVersion, mlso:justificationDropColumnX, mlso:justificationEthicalConsiderations, mlso:justificationExperimentName, mlso:justificationIntendedUse, mlso:justificationMetricChoice, mlso:justificationModelChoice, mlso:justificationModelLimitations, mlso:justificationNotIntendedFor, mlso:justificationTargetVariable, mlso:justificationTestSplit, mlso:justificationThresholdAccuracy, prov:Organization	FAIR4ML, MLSEA, PROV-O	MLFlow
Plan*	prov:Plan, prov:scriptName, prov:usedNotebook	PROV-O	MLFlow
title*	dcterms:title, fair4ml:trainedOn, mls:hasInput, mlso:trainedOn	FAIR, FAIR4ML, MLSEA	DBRepo
Algorithm*	mls:Algorithm, mls:hasAlgorithmType, mls:implements, mlso:hasAlgorithmType, prov:SoftwareAgent	MLSEA, PROV-O	MLFlow
Catalog*	dcat:Catalog, dcat:Distribution, dcat:distribution, dcterms:publisher	FAIR	DOI
Agent*	prov:Agent, prov:Person, prov:agent, prov:wasAttributedTo, prov:wasEndedBy, prov:wasStartedBy	PROV-O	MLFlow
transform*	croissant:transform, mls:preprocessingSteps, mlso:ignoresFeature	Croissant, MLSEA	MLFlow
citeAs*	croissant:citeAs, dcterms:bibliographicCitation, dcterms:isReferencedBy	Croissant, FAIR	DOI
End*	prov:End, prov:generatedAtTime	PROV-O	MLFlow
trainingStartTime*	fair4ml:trainingStartTime, prov:startedAtTime	FAIR4ML, PROV-O	MLFlow
Attribution*	prov:Attribution, prov:commitAuthor	PROV-O	MLFlow
trainingScriptVersion*	fair4ml:trainingScriptVersion, prov:commit	FAIR4ML, PROV-O	MLFlow
containedIn*	croissant:containedIn, dcterms:hasPart, dcterms:hasVersion, dcterms:partOf	Croissant, FAIR	DBRepo
data*	croissant:data, croissant:includes, mls:Feature, mls:FeatureCharacteristic, mlso:featureSelection	Croissant, MLSEA	DBRepo
Data*	mls:Data, mlso:hasType	MLSEA	DBRepo
dataset*	dcat:dataset, dcat:servesDataset, dcterms:identifier, fair4ml:evaluatedOn, fair4ml:testedOn, fair4ml:validatedOn, mls:Dataset, prov:Entity, prov:hadUsage, prov:wasDerivedFrom	FAIR, FAIR4ML, MLSEA, PROV-O	DBRepo
trainAccuracy*	mlso:trainAccuracy, mlso:trainF1, mlso:trainPrecision, mlso:trainRecall, mlso:trainScore	MLSEA	MLFlow
dateSubmitted*	dcterms:dateSubmitted, dcterms:issued	FAIR	DOI
trainingEndTime*	fair4ml:trainingEndTime, prov:endedAtTime	FAIR4ML, PROV-O	MLFlow
author*	dcterms:author, dcterms:creator	FAIR	DOI

Continued on next page

Table 4.2: Automatically captured metadata fields

Field	Ontology Terms	Ontologies	Capture Sources
qualifiedDerivation	prov:qualifiedDerivation	PROV-O	MLFlow
qualifiedUsage	prov:qualifiedUsage	PROV-O	MLFlow
usageInstructions	fair4ml:usageInstructions	FAIR4ML	MLFlow
recall	mlso:recall	MLSEA	MLFlow
Process	mls:Process	MLSEA	MLFlow
used	prov:used	PROV-O	MLFlow
trainRocAuc	mlso:trainRocAuc	MLSEA	MLFlow
Usage	prov:Usage	PROV-O	MLFlow
runEnvironment	fair4ml:runEnvironment	FAIR4ML	System
trainLogLoss	mlso:trainLogLoss	MLSEA	MLFlow
schemaVersion	dcat:schemaVersion	FAIR	DOI
scientificReferenceOf	mlso:scientificReferenceOf	MLSEA	DOI
Start	prov:Start	PROV-O	MLFlow
precision	mlso:precision	MLSEA	MLFlow
testSplit	mlso:testSplit	MLSEA	MLFlow
rocAuc	mlso:rocAuc	MLSEA	MLFlow
accessRights	dcterms:accessRights	FAIR	DOI
justificationBootstrap	mlso:justificationBootstrap	MLSEA	MLFlow
osPlatform	prov:osPlatform	PROV-O	MLFlow
achieves	mls:achieves	MLSEA	MLFlow
actedOnBehalfOf	prov:actedOnBehalfOf	PROV-O	MLFlow
branch	prov:branch	PROV-O	MLFlow
category	prov:category	PROV-O	MLFlow
column	croissant:column	Croissant	DBRepo
commitEmail	prov:commitEmail	PROV-O	MLFlow
commitTime	prov:commitTime	PROV-O	MLFlow
content	croissant:content	Croissant	MLFlow
created	dcterms:created	FAIR	DOI
dataType	croissant:dataType	Croissant	MLFlow
delimiter	croissant:delimiter	Croissant	Derived
EvaluationProcedure	mls:EvaluationProcedure	MLSEA	MLFlow
f1Score	mlso:f1Score	MLSEA	MLFlow
fileProperty	croissant:fileProperty	Croissant	MLFlow
fileSet	croissant:fileSet	Croissant	DBRepo
format	croissant:format	Croissant	MLFlow
hasCacheFormat	mlso:hasCacheFormat	MLSEA	MLFlow
modified	dcterms:modified	FAIR	DBRepo
modelVersion	mls:modelVersion	MLSEA	MLFlow
modelName	mls:modelName	MLSEA	MLFlow
ModelCharacteristic	mls:ModelCharacteristic	MLSEA	MLFlow
language	dcterms:language	FAIR	DOI
labelMap	mlso:labelMap	MLSEA	MLFlow
platform	prov:platform	PROV-O	MLFlow
key	croissant:key	Croissant	MLFlow
imbalanceRatio	mlso:imbalanceRatio	MLSEA	MLFlow
hasOutput	mls:hasOutput	MLSEA	MLFlow
hasLearningMethodType	mlso:hasLearningMethodType	MLSEA	MLFlow
hasEvaluationProcedureType	mlso:hasEvaluationProcedureType	MLSEA	MLFlow
hasEvaluationMeasureType	mlso:hasEvaluationMeasureType	MLSEA	MLFlow
hasCO2eEmissions	fair4ml:hasCO2eEmissions	FAIR4ML	MLFlow
wasAssociatedWith	prov:wasAssociatedWith	PROV-O	MLFlow
wasInformedBy	prov:wasInformedBy	PROV-O	MLFlow

Note: Fields marked with * indicate that the same recorded value is mapped to more than one ontology term, reflecting cross-ontology reuse for that metadata element.

Without the need for user participation, automatically acquired fields Tables 4.2 are obtained only through system instrumentation, logs, or API requests. MLflow,

Git, the operating system, or third-party services like DBRepo, and Invenio are the main sources of these fields.

Table 4.3: Semi-Automatically Captured Metadata Fields

Field	Ontology Terms	Ontology
MLModel*	fair4ml:MLModel, fair4ml:mlTask, fair4ml:modelCategory	FAIR4ML
HyperParameter*	fair4ml:fineTunedFrom, mls:HyperParameter, mls:HyperParameterSetting, mls:Implementation, mls:hasHyperParameter, mls:specifiedBy, prov:EntityInfluence, prov:Influence	MLSEA,PROVO-O,FAIR4ML
Model*	fair4ml:modelID, mls:Model	MLSEA,FAIR4ML
ModelCharacteristic	mls:ModelCharacteristic	MLSEA
modelName	mls:modelName	MLSEA
modelVersion	mls:modelVersion	MLSEA
hasTrainTestSplitIndices*	mlso:hasTrainTestSplitIndices, mlso:trainSplit	MLSEA
testSplit	mlso:testSplit	MLSEA
HyperParameterSetting*	fair4ml:fineTunedFrom, mls:HyperParameter, mls:HyperParameterSetting, mls:Implementation, mls:hasHyperParameter, mls:specifiedBy, prov:EntityInfluence	MLSEA, FAIR4ML,PROVO-O
trainSplit*	mlso:hasTrainTestSplitIndices, mlso:trainSplit	MLSEA
dataType	croissant:dataType	Croissant
delimiter	croissant:delimiter	Croissant
examples	croissant:examples	Croissant
modelCategory*	fair4ml:MLModel, fair4ml:mlTask, fair4ml:modelCategory	FAIR4ML

Note: Fields marked with * indicate that the same recorded value is mapped to more than one ontology term (cross-ontology reuse).

The semi-automatically collected fields are populated using a hybrid approach (see Table 4.3). Users typically select from predefined options presented in the interface, such as choosing a model type from the supported algorithms or a dataset from DBRepo. In other cases, users may provide values directly, for example specifying test/train split ratios, random seed values, or numerical hyperparameters. The system applies rule-based logic and internal mappings to normalize, validate, and, where applicable, enrich these inputs. This approach ensures repeatable configuration while preserving flexibility for custom experimentation, achieving a balance between user control and system-level consistency. The fields listed in Table 4.3 are provided as representative examples; they do not imply that only these parameters can be populated in a semi-automated manner.

Table 4.4: Manually Entered Metadata Fields

Field	Ontology Terms	Ontology
BootstrapChoice	mlso:justificationBootstrap	MLSEA
ClassWeightChoice	mlso:justificationClassWeight	MLSEA
ContainedIn	croissant:containedIn	Croissant
CriterionChoice	mlso:justificationCriterion	MLSEA
Data	croissant:data	Croissant
DatasetVersionRationale	mlso:justificationDatasetVersion	MLSEA
DropColumnXChoice	mlso:justificationDropColumnX	MLSEA
EthicalConsiderations*	fair4ml:ethicalLegalSocial, mlso:justificationEthicalConsiderations	FAIR4ML, MLSEA
ExperimentName	mlso:justificationExperimentName	MLSEA
FileProperty	croissant:fileProperty	Croissant
IntendedUse*	fair4ml:intendedUse, mlso:justificationIntendedUse	FAIR4ML, MLSEA
IsLiveDataset	croissant:isLiveDataset	Croissant
LearningRelatedSoftware	mlso:hasRelatedSoftware	MLSEA
MaxDepthChoice	mlso:justificationMaxDepth	MLSEA
MaxFeaturesChoice	mlso:justificationMaxFeatures	MLSEA
MetricChoiceRationale	mlso:justificationMetricChoice	MLSEA
MinSamplesLeafChoice	mlso:justificationMinSamplesLeaf	MLSEA
MinSamplesSplitChoice	mlso:justificationMinSamplesSplit	MLSEA
ModelChoiceRationale	mlso:justificationModelChoice	MLSEA
ModelLimitations*	fair4ml:modelRisks, mlso:justificationModelLimitations	FAIR4ML, MLSEA
NEstimatorsChoice	mlso:justificationNEstimators	MLSEA
NJobsChoice	mlso:justificationNJobs	MLSEA
NotIntendedFor	mlso:justificationNotIntendedFor	MLSEA
RecordSet	croissant:recordSet	Croissant
TargetVariableRationale	mlso:justificationTargetVariable	MLSEA
TestSplitRationale	mlso:justificationTestSplit	MLSEA
ThresholdAccuracy	mlso:justificationThresholdAccuracy	MLSEA

Note: Fields marked with * indicate that the same recorded value is mapped to more than one ontology term (cross-ontology reuse).

Through a structured justification form, users directly enter manually recorded fields as shown in Table 4.4. In order to facilitate interpretability and traceability, these variables are mostly mapped to MLSEA concepts and are used to record experiment justification, ethical considerations, design intent, and model assumptions.

Conflict Resolution and Scoped Naming

Numerous metadata categories use conceptually overlapping language or originate from different ontologies as seen in Section 1.

- **Temporal Resolution:** Fields such as `prov:startedAtTime` and `prov:endedAtTime` capture temporal aspects at the ontology level. In PROV-O, `prov:startedAtTime` and `prov:endedAtTime` can be associated with any *prov:Activity* - for example, the overall MLflow run, a preprocessing stage(as in this case), or a model evaluation job. These are retained separately because they describe different points in time within the provenance graph and are mapped to formal PROV-O concepts.

- **Version Field Disambiguation:** Across datasets, models, scripts, and runtime environments, several version-related attributes are recorded. Each is scoped with an ontology term to preserve its semantic intent and retrieval context:
  - `dcterms:hasVersion` - dataset version identifier (semi-automated),
  - `mls:modelVersion` - trained model artifact version (automatic),
  - `prov:pythonVersion` - Python runtime version in the execution environment (automatic).

  This scoped separation supports fine-grained repeatability and improves interpretability when reproducing or auditing runs.

- **Fallback and Uncategorized Fields:** Attributes not present in the matched ontology term list, or without a stable semantic equivalent, are placed in an `Uncategorized` group.

Justification Block Design

The **Justification Block** captures human-centred metadata-information about *why* specific modelling choices were made, intended use and limitations, and ethical considerations - which cannot be inferred from technical parameters alone. This supports accountability, auditability, and informed reuse decisions. The block is intentionally extensible: the framework code can be adapted to project-specific requirements for documenting modelling rationale as in Table 4.5.

Ontology-aligned fields and purpose:

Prompting logic: At the end of a run, immediately before the commit/export step, the framework blocks execution to display a structured free-text form for all fields. If left blank, a placeholder (e.g., "No justification provided") is stored, ensuring omissions are explicit. While completion is encouraged, it is not enforced to avoid blocking iterative experimentation.

Table 4.5: Justification Block Fields and Their Purpose

Ontology Terms	Purpose
`fair4ml:intendedUse`, `mlso:justificationIntendedUse`	Scope of valid application.
`mlso:justificationNotIntendedFor`	Prohibited or high-risk uses.
`fair4ml:ethicalLegalSocial`, `mlso:justificationEthicalConsiderations`	Privacy, fairness, or data-sensitivity considerations.
`mlso:justificationModelChoice`	Rationale for algorithm selection.
`mlso:justificationMetricChoice`	Justification for metric selection.
`mlso:justificationTargetVariable`	Reasoning for label choice.
`mlso:justificationDatasetVersion`	Dataset version used and motivation for selection.
`mlso:justificationDropColumnX`	Reasoning for removing specific features.
`mlso:justificationTestSplit`	Rationale for chosen train/test split ratio.
`mlso:justificationThresholdAccuracy`	Minimum acceptable model performance threshold.
`mlso:justificationMaxDepth`, `mlso:justificationNEstimators`, `mlso:justificationCriterion`, `mlso:justificationClassWeight`, `mlso:justificationMinSamplesLeaf`, `mlso:justificationMinSamplesSplit`, `mlso:justificationNJobs`, `mlso:justificationBootstrap`	Intent behind selected hyperparameter values.
`fair4ml:modelRisks`, `mlso:justificationModelLimitations`	Known weaknesses and limitations of the model.
`mlso:hasRelatedSoftware`	Relevant external software dependencies that influence interpretation or reuse.
`croissant:containedIn`, `croissant:data`, `croissant:fileProperty`, `croissant:isLiveDataset`, `croissant:recordSet`	Dataset structural or packaging metadata provided manually when required for contextual clarity.

Export and Interoperability Layer

The finalized metadata is exported in two standard linked data formats - JSON-LD and RDF/XML-to enable semantic interoperability (see Figure 4.4). These exports are derived from a unified RDF graph constructed using `rdflib`, based on ontology-aligned mappings to five specific standards: FAIR, PROV-O, Croissant, FAIR4ML, and MLSEA.

- **JSON-LD Export:** The complete RDF graph is serialized in JSON-LD format and saved as `prov_JSONLD_export.jsonld`. This supports compatibility with SPARQL endpoints, RDF triple stores, and reasoning engines. See Zenodo for an example.
- **RDF/XML Export:** The same RDF graph is also serialized in RDF/XML format as `prov_RDFXML_export.rdf`. This supports compatibility with SPARQL endpoints, RDF triple stores, and reasoning engines. See Zenodo for an example.
- **Graph Construction:** The export process constructs a typed RDF graph in which all resources are assigned RDF classes: `prov:Entity` (e.g., datasets,

models), `prov:Activity` (e.g., training run), and `prov:Agent` (e.g., user). All instances are scoped with run-specific URIs (e.g., `/run_20250718_131502/`) to support traceability and disambiguation.

- **Export Triggering:** Once the structured metadata JSON is processed and all mappings are applied, the RDF export process is triggered automatically. The resulting JSON-LD and RDF/XML files. The complete export logic can be found in Zenodo

Export Graph Generation (Excerpt):

```
# Define core URIs
model_uri = safe_uri(EX, f"model/{structured_metadata['Croissant'].get('mls:modelID')}")
dataset_uri = safe_uri(EX, f"dataset/{structured_metadata['FAIR'].get('dcterms:identifier')}")
code_uri = safe_uri(EX, f"code/{structured_metadata['PROV-O'].get('prov:commit')}")
activity_uri = safe_uri(EX, f"run/{run_id}")
agent_uri = safe_uri(EX, f"agent/{structured_metadata['PROV-O'].get('prov:Agent')}")

# Assign RDF types
g.add((model_uri, RDF.type, PROV.Entity))
g.add((dataset_uri, RDF.type, PROV.Entity))
g.add((code_uri, RDF.type, PROV.Entity))
g.add((activity_uri, RDF.type, PROV.Activity))
g.add((agent_uri, RDF.type, PROV.Agent))

# Establish relationships
g.add((activity_uri, PROV.used, dataset_uri))
g.add((activity_uri, PROV.used, code_uri))
g.add((model_uri, PROV.wasGeneratedBy, activity_uri))
g.add((activity_uri, PROV.wasAssociatedWith, agent_uri))

# Inject literals by category
for category, subject in [
    ("Croissant", model_uri),
    ("FAIR", dataset_uri),
    ("PROV-O", code_uri),
    ("FAIR4ML", activity_uri),
    ("MLSEA", activity_uri)
]:
    add_literals(structured_metadata.get(category, {}), subject)
```

Export Pipeline Overview:

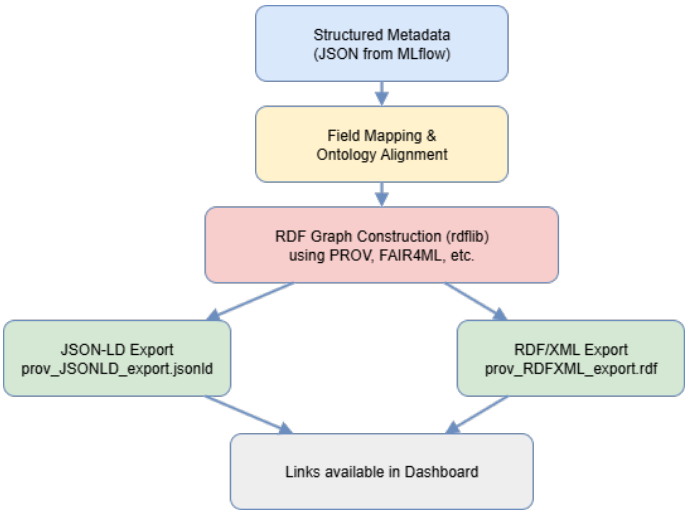


Figure 4.4: Export process for structured semantic metadata

SPARQL Support

Because all RDF-based exports work with SPARQL 1.1, the resulting provenance graph may be queried in a variety of ways. Users have the option to write their own queries or choose from the pre-written ones.

Here is an example SPARQL query that returns the experiment start time, dataset ID, and model ID for a certain execution:

```
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX ex: <https://github.com/reema-dass26/ml-provenance/provenance/run_20250718_131502/>

SELECT ?model ?dataset ?startTime
WHERE {
    ?run a prov:Activity ;
         prov:used ?dataset ;
         prov:startedAtTime ?startTime .
    ?model prov:wasGeneratedBy ?run .
}
```

Support for standards-based querying and structured exporting ensures long-term archiving, downstream reusability, and compliance with provenance-aware machine learning best practices and FAIR.

4.4 Implementation in the VRE

The full implementation of the provenance-aware machine learning process within the Virtual Research Environment (VRE) is described in this section. Written from the user's point of view, it covers every stage of execution, from metadata injection to semantic export, while defining the bounds of automation and points of connection with MLflow, Jupyter, Git, DBRepo, and Invenio Figure 4.5.

1. Session Initialization

Session-level metadata, including as the timestamp, environment details, and user ID, are recorded by the first code block run in the Jupyter notebook. To provide consistent traceability throughout all ensuing activities, this metadata is put into the MLflow context.

2. Dataset Selection, Retrieval, and Notebook Provenance Capture

The system connects to a metadata-enabled repository endpoint for dataset access. In the current setup, this is an active DBRepo instance with pre-uploaded datasets, but the architecture is endpoint-agnostic and can work with any API-based service (e.g., Hugging Face Datasets Hub, Zenodo) by adapting the retrieval and mapping logic.

- **Provenance capture:** Dataset retrieval automatically records notebook-level provenance (session metadata, dataset ID, API endpoint, dataset columns) to link the chosen dataset to the exact code state for reproducibility.
- The user selects a dataset from a multiple-choice interface connected to the current endpoint.
- Metadata is fetched from the endpoint via key alignment and the dataset is loaded into the notebook.
- Optionally, entering a DOI triggers an external metadata lookup (e.g., DataCite, Crossref) for enrichment.

- **DBRepo-specific fields:** internal ID, title, description, creators, license, upload timestamp, version tag, file checksum, and size.

3. Experiment Configuration and Execution

After dataset ingestion, the user is prompted to provide key configuration details such as model type, test/train split ratio, experiment name, hyperparameters, and random seed.

- **Preprocessing:** Triggered either (i) using the framework's default preprocessing pipeline (e.g., automatic type coercion, dropping ID columns, splitting into train/test) or (ii) using user-supplied parameters for steps such as scaling, encoding, feature selection, or sampling. The exact preprocessing sequence is stored in the metadata.

- **Model execution:** The chosen model is instantiated, trained, and evaluated according to the configuration. For the purposes of this implementation, the ML tasks are intentionally kept as relatively **simple classification problems** so that the evaluation can focus on provenance tracking and metadata completeness rather than on complex model architectures or domain-specific optimisation.

- **Notebook adaptability:** While the current notebook implements a fixed sequence (metrics, confusion matrix, ROC/PR curves), the architecture allows *any* notebook to be adapted by inserting minimal annotation cells or function calls at key stages (e.g., after preprocessing, after training, after evaluation). This annotation enables the framework to map the notebook's variables and outputs to the corresponding metadata fields.

- **Logging:** All parameters, tags, runtime environment, and model artifacts are automatically logged to MLflow during execution.

- **Git integration:** Activated once all metadata (including Justification Block inputs) is collected. In the background, a commit is triggered, capturing Git hash, branch, status, and author details, which are injected into the metadata. The user is then prompted to enter commit version tag.

4. Justification Metadata Entry

Following execution, users are prompted to fill out the justification block explained in Section 1. This data is also logged in MLflow.

5. MLflow Metadata Dump and Parsing

After completion, MLflow logs are dumped into a raw JSON file that aggregates all tracked experiment data. This file includes:

- Session metadata (timestamps, run name, environment)
- Dataset details (merged from DBRepo and optional DOI enrichment)
- Model parameters and performance metrics
- Git metadata (commit hash, commit author, branch, script name, repository URL)
- Runtime environment details (Python version, OS, notebook used)
- Experiment metadata and user-supplied justification inputs

This metadata is parsed and inserted into structured tables within DBRepo (Appendix B.1). These names also reflect the table names in the DBRepo database.

6. Semantic Harmonization and Export

Following parsing explained in Section 1:

- A renaming function maps internal metadata keys directly to ontology terms.
- The mapped metadata is written to a structured JSON file (Zenodo), which serves as the source for the dashboard and two export formats:
  - `JSON-LD` (Linked Data)
  - `RDF/XML` (for LOD compatibility)

7. Invenio Integration and Background Archival

In the background, an automated archival process is triggered via Invenio:

- A draft is created in Invenio, which includes the structured JSON, semantic exports (JSON-LD, RDF/XML), generated plots, and the trained model (.pkl), and is then published to the repository.
- Metadata returned by Invenio (e.g., record ID, submission timestamps, archive link) is appended to the structured JSON for completeness.

8. Full-Stack Integration

Throughout this process Figure 4.5., all metadata from DBRepo, Jupyter session, Git, and justification inputs is tracked using MLflow, ensuring complete traceability and semantic auditability.

Figure 4.5 illustrates the full execution flow, beginning from notebook initiation to final archival. Metadata is incrementally collected from DBRepo, the notebook environment, Git, and user inputs at various stages of the ML pipeline. After logging via MLflow, it is parsed and inserted into structured DBRepo tables. Post this, the renaming function maps internal fields to ontology terms, producing a harmonized JSON file used for export and visualization. An automated archival step then uploads these artifacts to Invenio.

4.5 Visualization and Interaction Layer

The dashboard makes it possible to examine experiment metadata interactively by using a modular Streamlit interface. The `streamlit-option-menu` library is used to organize the interface into tabs, with each metadata category shown in its own panel. The dashboard's only data source is a structured JSON file that is produced after metadata harmonization.

Users can utilize the interface to filter, review, and export pertinent metadata subsets for each experiment that has been finished, and can access these runs by selecting the run_id from the drop-down at each panel.

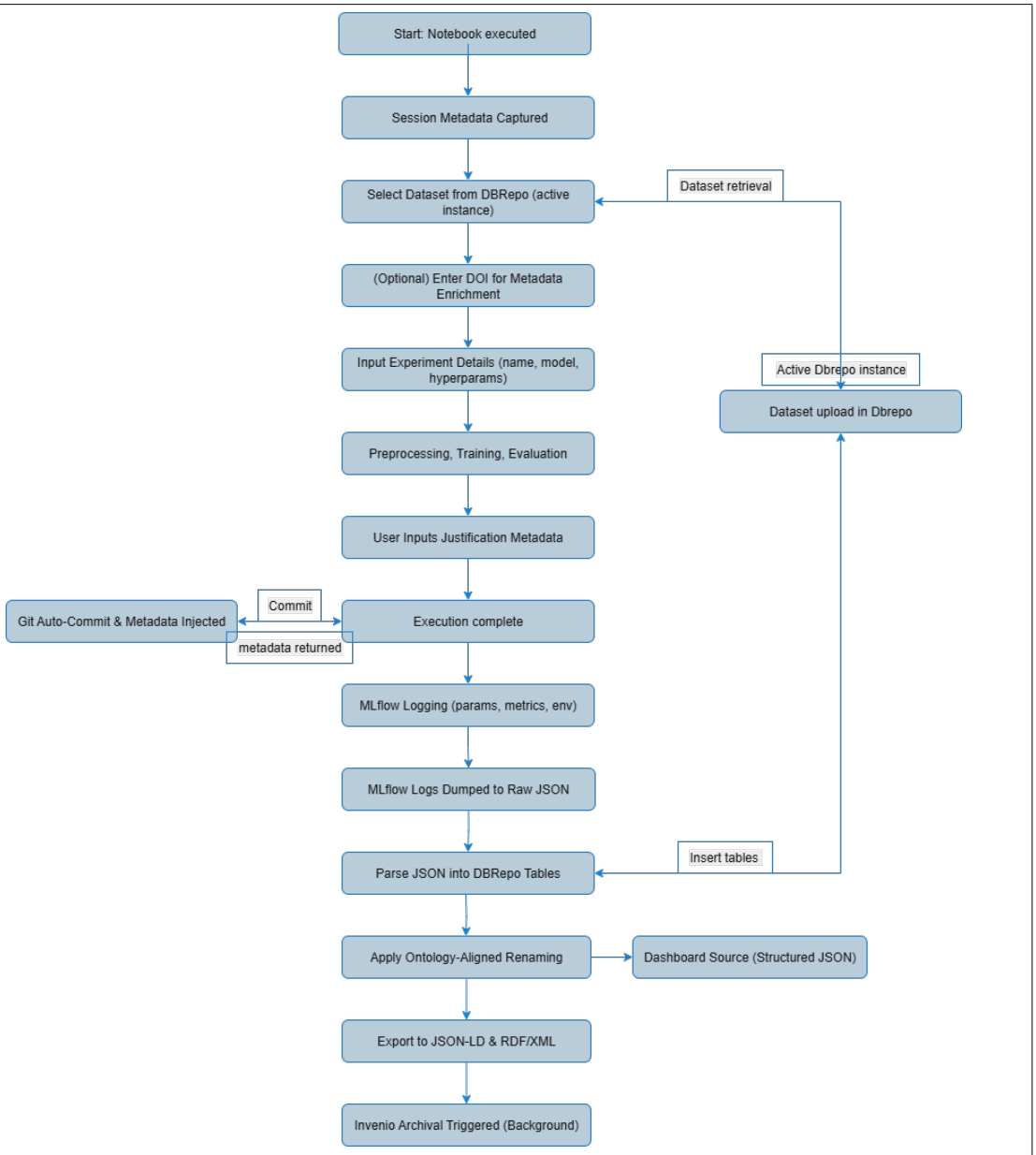


Figure 4.5: Execution flow diagram

UI Tabs and Functionality

- **Dashboard:** Gives a summary of every metadata section in a graphically organized. An interactive narrative walkthrough of the complete infrastructure flow(BOX# 2) is included, along with render-styled cards that provide an overview of the features(BOX# 1), and on the left, we see tabs such as the Dataset, Model, Justifications, and Export(BOX# 3).

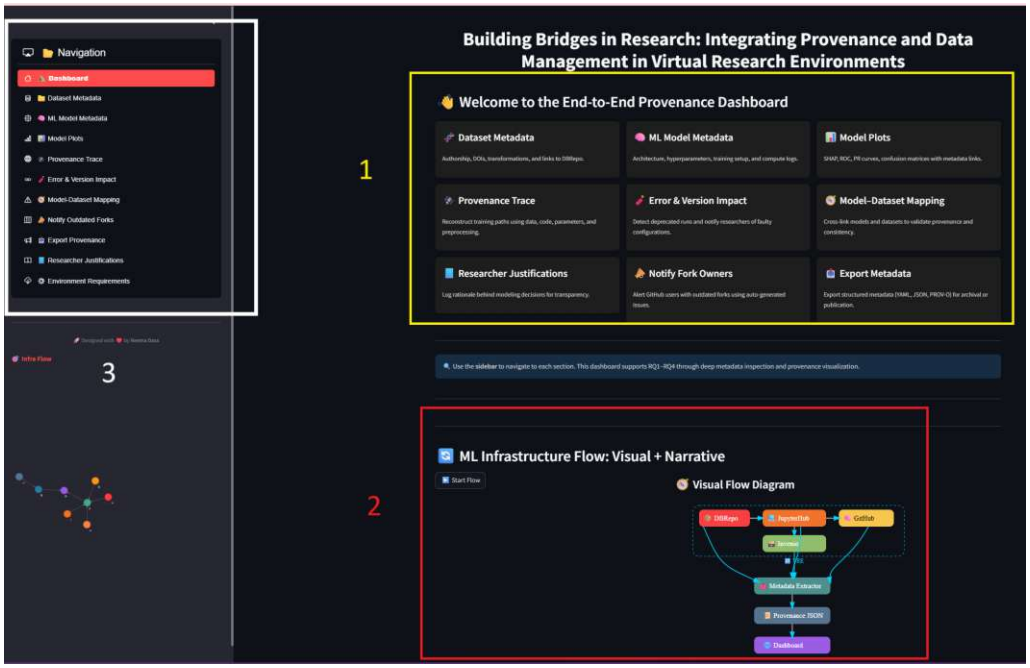


Figure 4.6: Overview dashboard with infrastructure walkthrough and metadata summary cards

- **Dataset Metadata:** Shows metadata at the dataset level in three different tables, including dataset title, version, DOI, source, and preparation procedures.

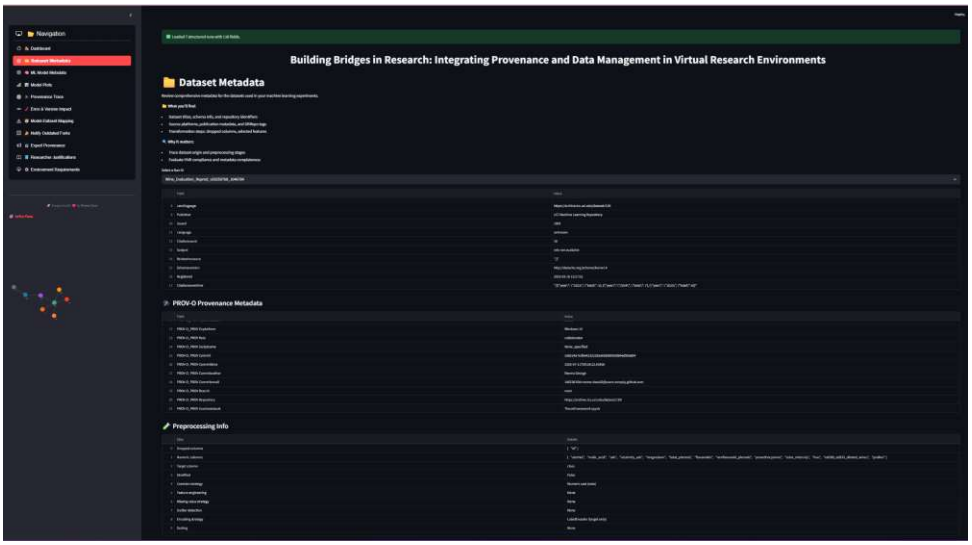


Figure 4.7: Dataset metadata view showing title, DOI, preprocessing summary, and DBRepo tags

- **ML Model Metadata:** Enables users to examine associated model details, such as type, hyperparameters, train and test metrics (accuracy, F1), training

timestamps, and MLflow run ID, by selecting a specific run ID. Comparative analysis is supported through juxtaposed display of multiple runs.

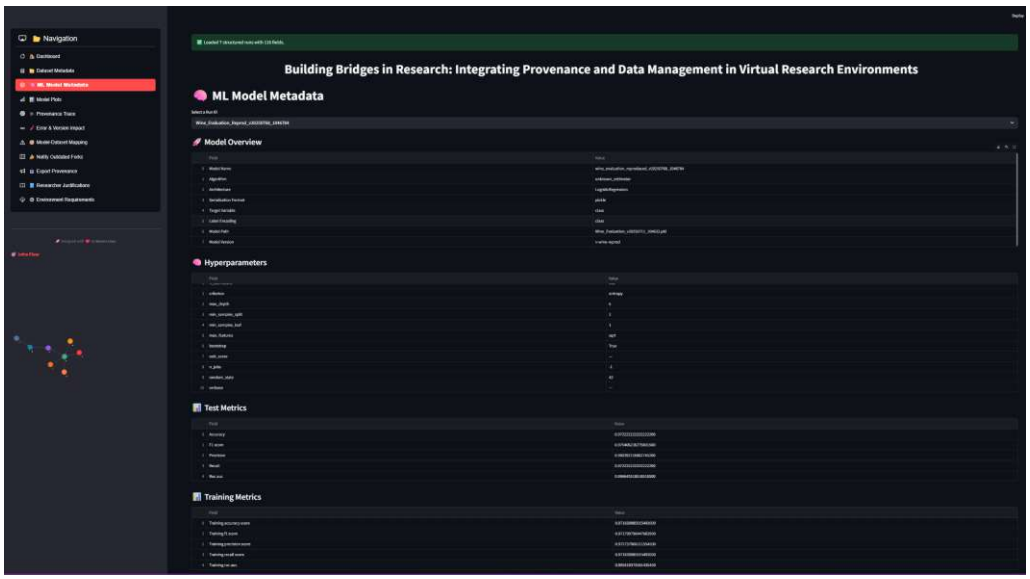


Figure 4.8: Model metadata viewer with hyperparameters, training metrics, and version information

- **Model Plots:** Displays evaluation visuals such as ROC/AUC curves, feature importance plots, and confusion matrices, selectable from the dropdown in Figure 4.9. A subset of the metadata of the chosen run is shown for context (Figure 4.10, Box #1).

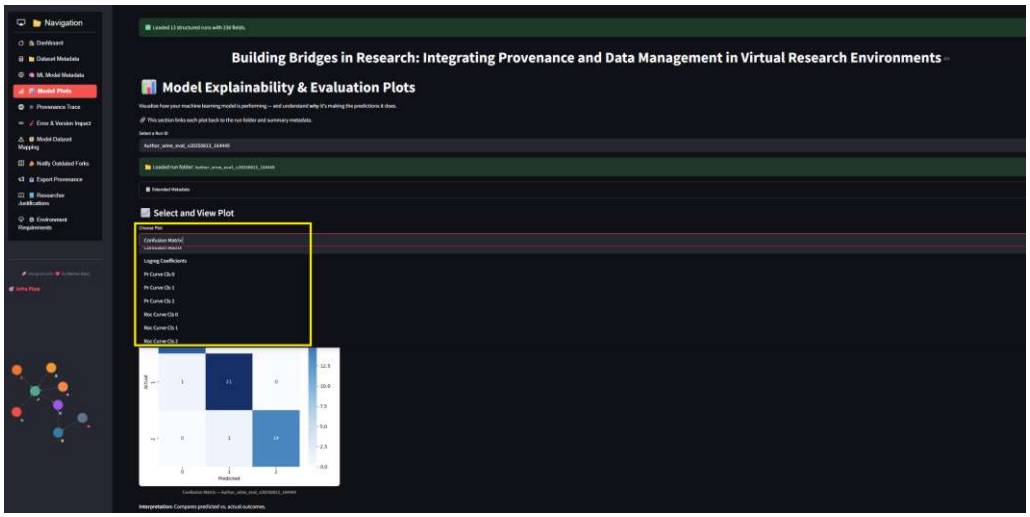


Figure 4.9: Model Plots options.

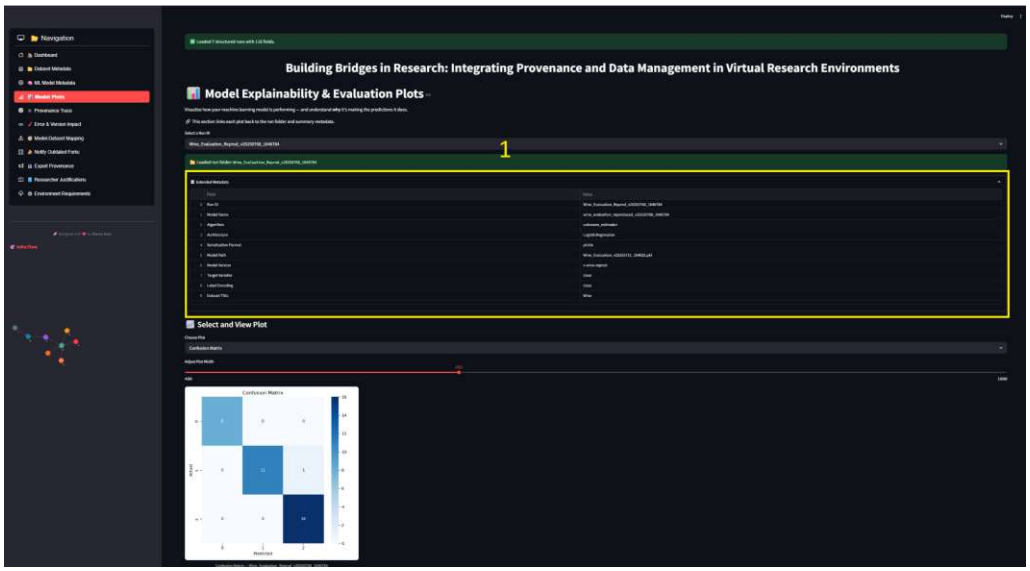


Figure 4.10: Model Plots tab showing the selected experiment's metadata for completeness.

- **Provenance Trace:** Provides an interactive comparison interface across training runs to show how configuration, parameters, and results have changed, enabling reproducibility analysis by including full metadata lineage.

  In Figure 4.11, the annotated areas correspond to:

BOX#1 Run selection and reproducibility guide: Allows the user to select the primary run for inspection, download the `reproducibility_guide` file (which contains hyperparameters, dataset version, preprocessing steps, metrics, and Git commit hash), and optionally check the "Compare with another run" box.

BOX#2 Comparison view: Displays the selected runs side-by-side. Differences in key fields such as Git commit hash, dataset version, or hyperparameters are highlighted to indicate *provenance mismatches* between the two executions.

BOX#3 Run dropdown: When comparison mode is enabled, a dropdown lists all available past runs; selecting one populates the comparison table in area #2.

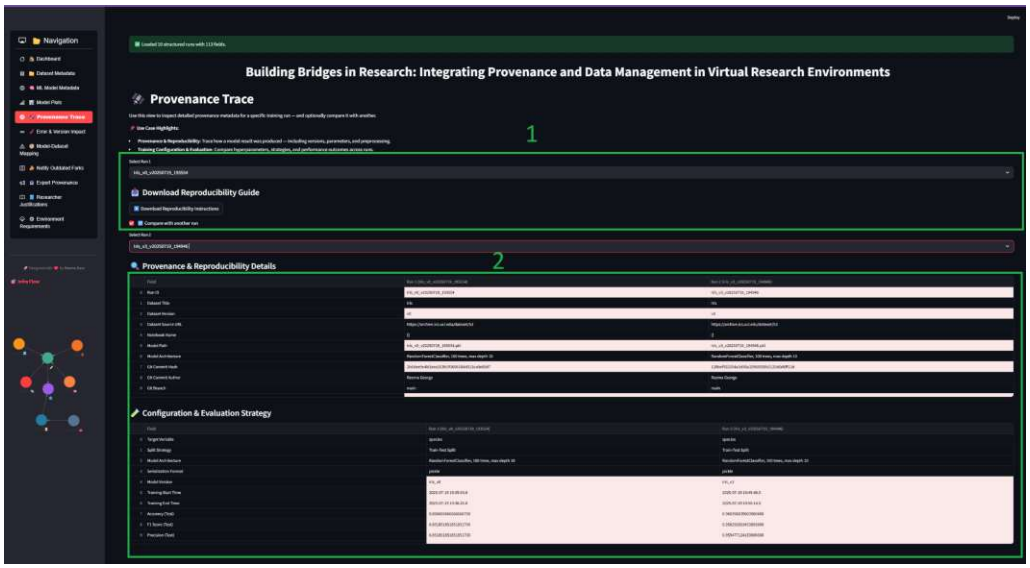


Figure 4.11: Provenance trace tab with annotated areas: #1 run selection ; #2 comparison table highlighting provenance mismatches (differences in Git commit, dataset version, or parameters) between runs.

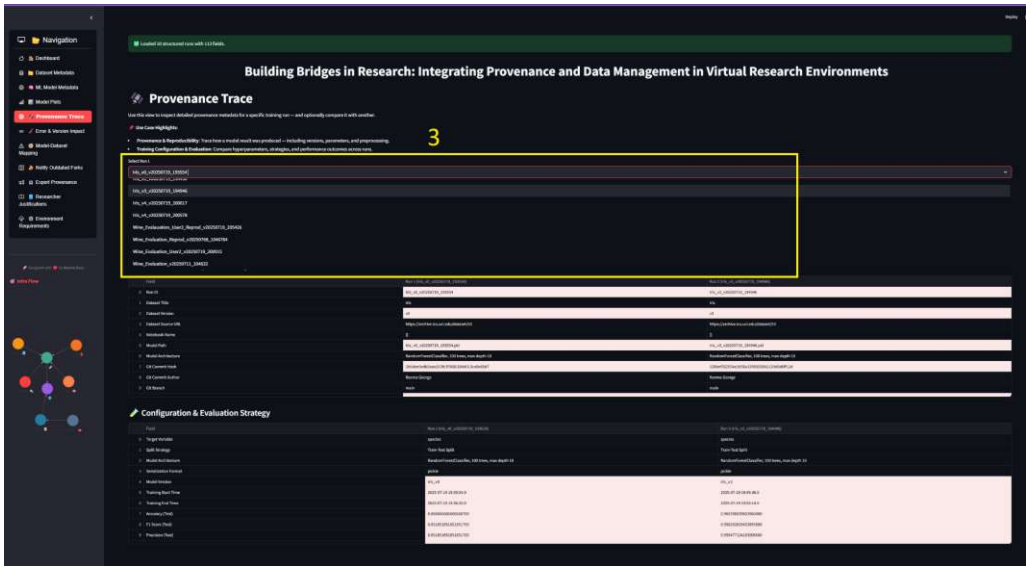


Figure 4.12: Provenance trace tab with annotated areas: #3 run selection from the dropdown list.

For every run selected, the tab also offers a downloadable **reproducibility guideline** (Figure 4.13), which is a structured text file containing:

- Model hyperparameters and evaluation metrics
- Preprocessing pipeline details
- Dataset version

– Git commit hash of the code used

The sample shown in Figure 4.13 is from the run `Wine_Evaluation_v20250711_104632`-the filename encodes the dataset name (`Wine`), model type (`Evaluation`), and timestamp. This file can be used as a step-by-step reference for recreating the exact experiment. The annotated areas correspond to:

#1 **Model Details:** Includes model name, dataset name, dataset version, run ID, and timestamp of execution.

#2 **Hyperparameters:** Lists the full training configuration used for the model (e.g., number of estimators, maximum depth, criterion, feature settings).

#3 **Metrics:** Displays the evaluation results for the run, including accuracy, macro F1-score, macro precision, macro recall, and ROC AUC.

#4 **Git Info and Reproduction Guide:** Provides the Git commit hash linked to the experiment, the exact `git checkout` command to retrieve it, and sequential instructions for reproducing the training and evaluation steps.

Figure 4.13: Annotated reproducibility guideline (`Wine_Evaluation_v20250711_104632_reproducibility.txt`) showing: #1 model details; #2 hyperparameters; #3 metrics; and #4 Git commit information with reproduction instructions.

- **Error & Version Impact**

  This use case demonstrates how the framework detects ML experiments affected by outdated or faulty code or dataset versions. In collaborative environments, such issues can propagate errors if not caught early. The tab works by comparing each run's Git commit hash and version tag against the current main branch and any user-specified deprecated tags. If a match is found, the corresponding runs are flagged as impacted. The user can then directly notify collaborators through GitHub.

  Figures 4.14 and 4.15 annotate the main components of this tab:

  #1 **Current Git Commit:** Shows the commit hash of the current working version for comparison.

  #2 **Experiment List:** Displays all tracked experiments with their commit hash, version tag, model type, accuracy, and dataset version.

  #3 **Deprecated Tag Search:** Field to enter a known faulty or outdated version tag; on execution, the table of impacted experiments is generated.

  #4 **GitHub Notification:** Allows the user to authenticate and send automated notifications to collaborators linked to the impacted runs.

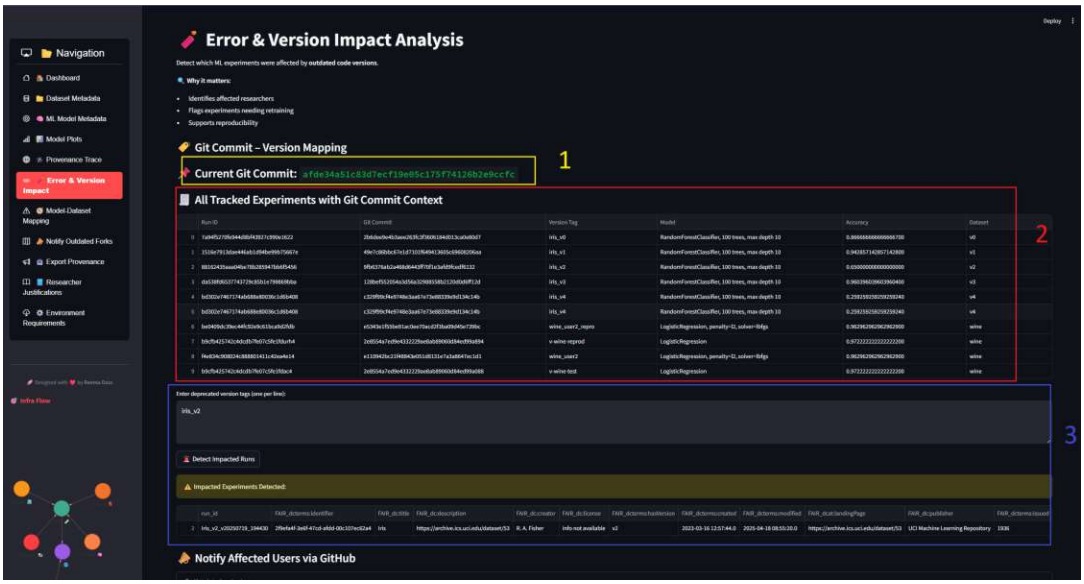


Figure 4.14: Annotated Error & Version Impact tab: #1 current Git commit; #2 experiment list; #3 deprecated version tag search and detection results.

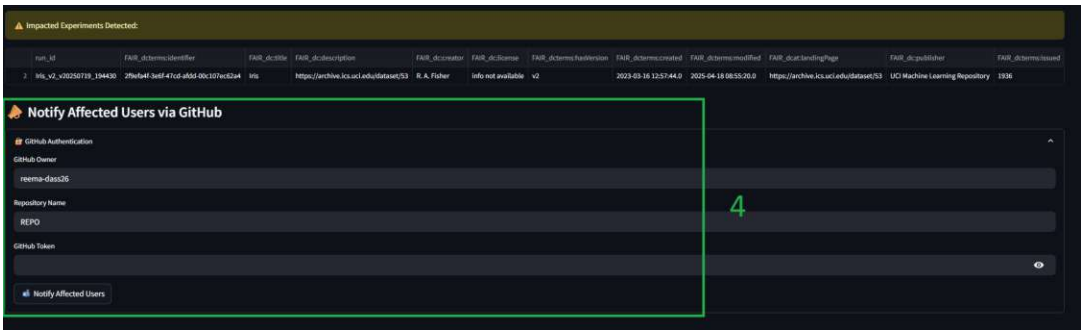


Figure 4.15: Annotated GitHub notification section #4 for contacting collaborators about impacted experiments.

- **Model-Dataset Mapping:** In collaborative and long-running projects, it is often necessary to verify that a given model can be reproduced from its original dataset and training conditions[39] - a process sometimes referred to as a *reproducibility audit*. Such audits aim to confirm that the training outcomes reported for a model can be regenerated using the documented dataset, preprocessing steps, and hyperparameters. This tab addresses that need by explicitly enumerating the connections between datasets and the machine learning models trained on them. It also exposes the associated metadata, enabling reviewers to confirm dataset-model consistency, check provenance records, and ensure reproducibility.

  In Figure 4.16, the annotated areas highlight:

  #1 **Run ID:** Unique identifier for each experiment execution.

  #2 **Model Details:** Name and architecture of the trained ML model.

  #3 **Dataset Details:** Dataset title, version, and persistent access URL.

  #4 **Performance Metrics:** Accuracy, recall, precision, and ROC AUC (test set).

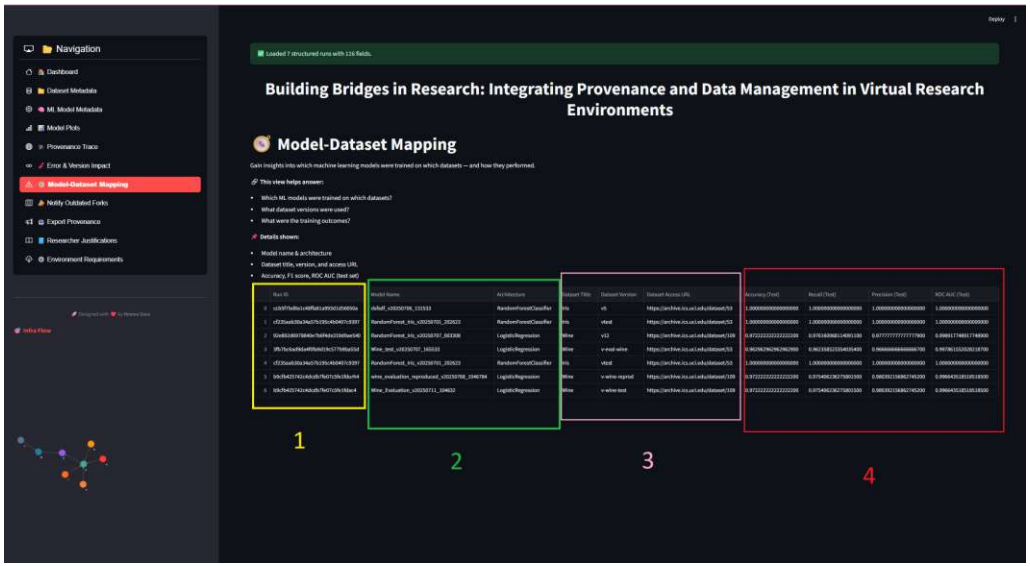


Figure 4.16: Annotated Model-Dataset Mapping tab showing dataset-model links, dataset metadata, and evaluation results.

- **Notify Outdated Forks:** In collaborative code development, forks that lag behind the main branch can introduce integration conflicts, propagate bugs, or result in experiments being run on outdated code. If experiments are executed on such forks without synchronized dataset or code versions, results may deviate from current baselines, thereby compromising reproducibility. The *Notify Outdated Forks* tab addresses this by integrating with GitHub to automatically compare commit histories, identify forks that are behind, and highlight version gaps. Users can initiate alerts or create GitHub issues directly from the interface to prompt timely updates.

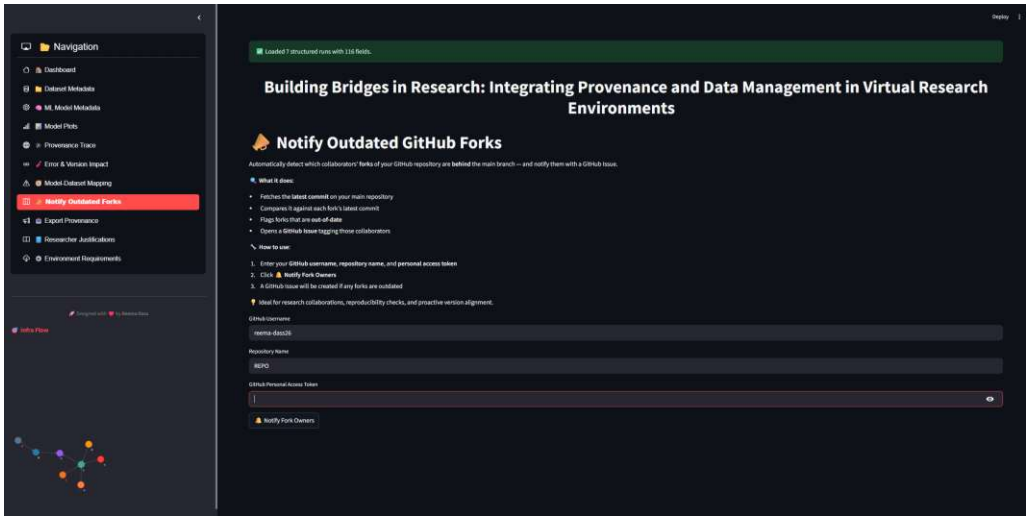


Figure 4.17: GitHub fork tracker highlighting outdated forks and version gaps.

- **Export Provenance:** Shows the '.jsonld' and '.rdf.xml' files and offers download links as well as support for SPARQL querying.

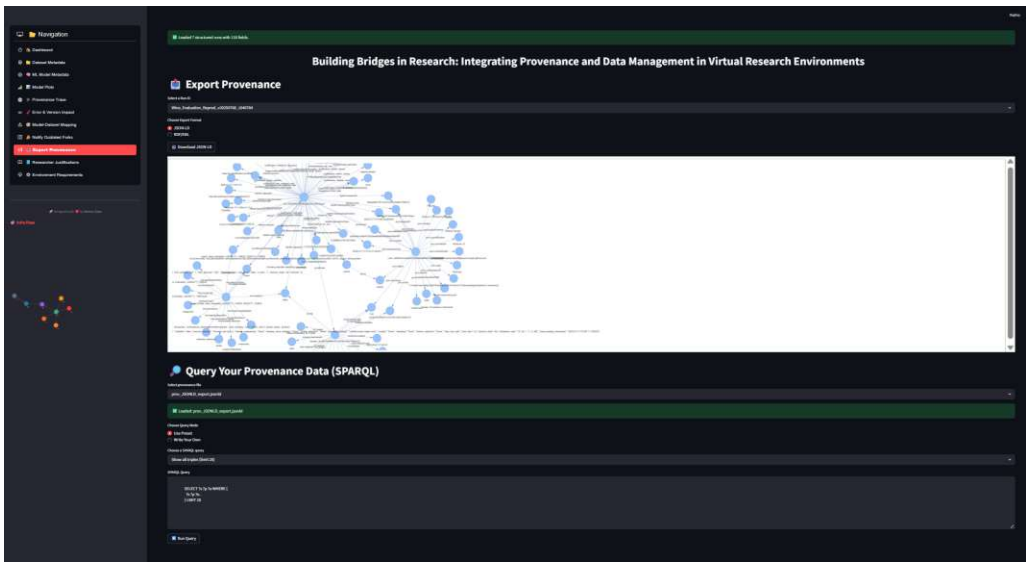


Figure 4.18: Export tab showing available RDF/XML and JSON-LD downloads per run

- **Researcher Justifications:** Shows every reasoning field that has been manually filled in, including the model's justification, the metric selection, the intended usage, and any ethical issues.

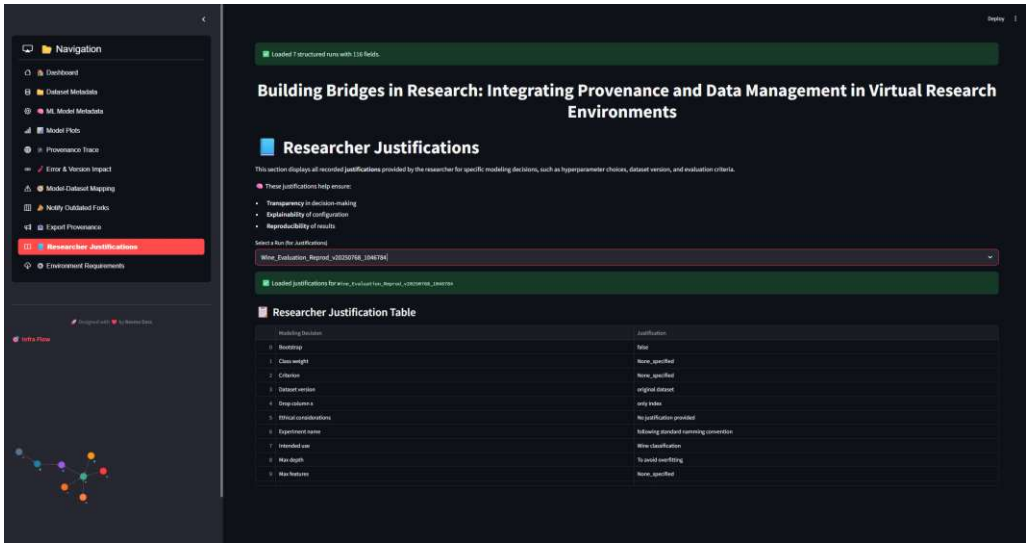


Figure 4.19: Researcher justification tab showing manual metadata.

- **Environment Requirements:** Lists the exact Python packages, versions, and installation paths that were active in the runtime environment. This

information is extracted from MLflow logs at execution time and displayed as a downloadable text file.

```
[
  "datashape==0.5.4",
  "GitPython==3.1.43",
  "matplotlib @ file:///C:/b/abs_085jhivdha/croot/matplotlib-suite_1693812524572/work",
  "matplotlib-inline==0.1.6",
  "mlflow==2.21.2",
  "mlflow-skinny==2.21.2",
  "numpy==1.24.4",
  "numpydoc @ file:///C:/ci_311/numpydoc_1676453412027/work",
  "pandas==2.2.3",
  "psutil==5.9.5",
  "PyLD==2.0.4",
  "python-dotenv @ file:///C:/ci_311/python-dotenv_1676455170580/work",
  "PyYAML @ file:///C:/ci_311/pyyaml_1676432488822/work",
  "pyyaml_env_tag==0.1",
  "rdflib==6.3.2",
  "requests @ file:///C:/b/abs_316c2inijk/croot/requests_1690400295842/work",
  "requests-file @ file:///Users/ktietz/demo/mc3/conda-bld/requests-file_1629455781986/work",
  "requests-toolbelt @ file:///C:/b/abs_2fsmts66wp/croot/requests-toolbelt_1690874051210/work",
  "scikit-learn @ file:///C:/b/abs_55olq_4gzc/croot/scikit-learn_1690978955123/work",
  "scikit-learn-intelex==20230426.121932",
  "seaborn @ file:///C:/ci_311/seaborn_1676446547861/work",
  "shap==0.47.1"
]
```


Figure 4.20: `requirements.json` file showing package names, versions, and installation paths from the recorded experiment environment.

Execution Selection Mechanism: At the top of each functional dashboard tab (e.g., *Provenance Trace*, *Model-Dataset Mapping*, *Error & Version Impact*), a dropdown menu allows the user to select from previously recorded experiment executions. Once selected, the experiment's context, model, environment, and full metadata trace are automatically reflected in all panels of that tab.

Summary

This chapter presented the design and implementation of a provenance-aware, ontology-aligned machine learning evaluation framework. The architecture was guided by requirements identified in the literature review, addressing gaps in reproducibility, metadata completeness, and standards-based interoperability.

The system adopts a modular, notebook-native design with clear separation of data retrieval, experiment execution, provenance capture, and semantic export. Integration with MLflow enables continuous, automated tracking of dataset, model, environment, and justification metadata, while the DBRepo backend supports structured storage and queryable access. Ontology mappings to PROV-O, FAIR, FAIR4ML, MLSEA, and Croissant ensure that captured metadata is semantically consistent and reusable across platforms.

By combining automated metadata capture with targeted manual inputs, the frame-

work supports both repeatable experimentation and human-centered documentation. The design is endpoint-agnostic, allowing substitution of data repositories or metadata APIs with minimal code changes. This architecture establishes the foundation for the evaluation presented in the next chapter, where its effectiveness is assessed against the thesis research questions.

5 Evaluation

5.1 Overview

This chapter evaluates the proposed provenance-aware ML framework along four research dimensions corresponding to the thesis research questions:

- **RQ1:** Metadata coverage, ontology alignment, and degree of automation.
- **RQ2:** Reproducibility traceability, debugging, version tracking, and configuration auditing.
- **RQ3:** Visualization effectiveness and usability.
- **RQ4:** Export into semantic formats for interoperability.

5.2 RQ1: Metadata Coverage, Metadata Alignment, and Degree of Automation

5.2.1 Use Case: Data and Model Provenance capture, alignment

Provenance-aware ML systems enable two things: (i) reproducible, auditable experiments and (ii) model governance over time - continuous, auditable oversight of models across their lifecycle (versioning, provenance and lineage, performance/-drift monitoring, controlled changes, and eventual deprecation). Both require the following:

1. **Coverage** — whether the framework records the *right kinds* of fields expected by established ontologies for ML experiments and datasets.

2. **Alignment** — whether recorded keys are *canonically mapped* to ontology terms (not merely lookalikes), so that downstream tools can parse and reason over them in a standards-based way.

3. **Automation** — whether those fields are captured *without burdening* the user, reducing omissions and human error at scale.

RQ1 tests these three aspects in a structured way. We use a reference built from official ontology releases and limit the analysis to tabular classification, so the baseline is clear and not padded with out-of-scope fields.

5.2.2 Scope, Assumptions, and Artifacts

Task scope. Supervised *classification* on tabular data, executed in a *notebook-native* workflow (Jupyter), including preprocessing, training, evaluation, artifact logging, and archival. Inference services, deployment-time governance, and non-tabular modalities are out of scope for RQ1.

Ontologies under test. We assess coverage against five property vocabularies: PROV-O, FAIR, Croissant, FAIR4ML, and MLSEA.

Definition of a run. A *run* is a single, end-to-end execution of the pipeline (session start → archival), yielding one structured metadata JSON plus linked artifacts (plots, model file, semantic exports) under local folder `MODEL_PROVENANCE/`, which is then archived in Invenio.

Runs evaluated. Nine executed notebooks:

- **Iris** variants (v0–v4): controlled perturbations for stress-testing provenance capture.
- **Wine** (User study): two users, original+reproduction each.

A full list of run IDs, timestamps, and dataset links is provided in Appendix A.2.

Table 5.1: Datasets and run purposes used in RQ1

Dataset/Run	Purpose
Iris v0 (Original)	Framework evaluation (baseline)
Iris v1 (Duplicated)	Robustness to duplicates
Iris v2 (First 100)	Subsampled training set
Iris v3 (Normalized)	Preprocessing variation
Iris v4 (Distorted)	Feature perturbation
Wine (User 1, Original)	User study: framework test
Wine (User 1, Reproduction)	Repeatability check
Wine (User 2, Original)	User study: framework test
Wine (User 2, Reproduction)	Repeatability check

5.2.3 Building the Reference: Ontology Union U_{all}

Objective. Create a single, reproducible *union catalog of ontology terms* expected in notebook-native ML experiments and dataset descriptions, sourced from official releases. Identifiers are normalized to stable prefixes and de-duplicated by IRI so the same term is not counted twice.

Sources. We ingest PROV-O, FAIR, DCTERMS, Croissant, FAIR4ML, and MLSEA from their official RDF serializations (Turtle/JSON-LD).

Table 5.2: Snapshots used to construct the ontology union U_{all}

Ontology	Source URL	Snapshot (ver/tag/commit)	Retrieved (UTC)	File / Format
PROV-O	https://www.w3.org/ns/prov.ttl	W3C Recommendation 2013-04-30	2025-08-13	prov.ttl / Turtle
FAIR Vocabulary	https://w3id.org/fair/principles/terms/FAIR-Vocabulary	5.1.11	2025-08-13	FAIR-Vocabulary.ttl / Turtle
Croissant	https://raw.githubusercontent.com/mlcommons/croissant/v1.0.21/docs/croissant.ttl	v1.0.21	2025-08-13	croissant.ttl / Turtle
FAIR4ML	https://rda-fair4ml.github.io/FAIR4ML-schema/release/0.1.0/fair4ml.jsonld	0.1.0	2025-08-13	fair4ml.jsonld / JSON-LD
MLSEA	http://w3id.org/mlsea	1.0.0	2025-08-13	mlsea.ttl / Turtle

Pipeline.

- **Parse:** Load each ontology file with `rdflib`.

- **Collect terms:** Extract *both* properties (`rdf:Property`, `owl:ObjectProperty`, `owl:DatatypeProperty`, `owl:AnnotationProperty`) *and* classes (`rdfs:Class`, `owl:Class`).
- **Normalize names:** Obtain qnames (qualified names, a.k.a. CURIEs: `prefix: localName`, e.g., `prov:startedAtTime` ↦ `http://www.w3.org/ns/prov#startedAtTime`) via the graph's namespace manager; strip angle brackets; fix default-namespace PROV forms (e.g.,`:startedAtTime`→`prov:startedAtTime`); if still a full IRI, map via a fixed base→prefix table (`prov`, `dcterms`, `dcat`, `croissant`, `schema`, `fair4ml`, `mls`/`mlso`, `foaf`).
- **De-duplicate:** Within each ontology, drop duplicate IRIs[1]; concatenate per-ontology frames (preserving overlaps for reporting).

Deliverables.

- **Evaluation catalog (terms = properties + classes).** `ontologies/Results/unified_keys_with_overlaps.csv` (see in Zenodo) - normalized keys per source with overlaps preserved; used for the coverage/alignment metrics reported in this section.
- **Auxiliary transparency artifacts.**
  - `ontologies/Results/ontology_catalog_with_overlaps.csv` (full catalog with properties+classes and overlaps)(see in Zenodo).
  - `ontologies/Results/overlapping_uris_across_ontologies.csv` (terms observed in multiple ontologies)(see in Zenodo).

The code used to generate the above files can be found in Zenodo.

[1] An IRI (Internationalized Resource Identifier) is the canonical, globally unique identifier for a term, e.g., `http://www.w3.org/ns/prov#startedAtTime`. Prefixed forms like `prov:startedAtTime` are abbreviations that expand to this IRI.

5.2.4 Evaluation Protocol: Full-Union Coverage and Alignment

Goal. Quantify ontology alignment between our `structured_metadata.json` (archived in Zenodo) and the complete union reference U_{all}. We use exact, case-sensitive equality of key strings (no normalization). In the same pass, we compute (i) baseline completeness, and (ii) Ontology alignment.

Inputs.

- **Baseline files:**
  - Complete Union:ontologies/Results/unified_keys_with_overlaps.csv (columns: `ontology`, `key`)(archived in Zenodo).
  - *Strict subset (task-scoped):* `ontologies/Results/relevant_subset/relevant_union_generic_plus_classification.csv` (see next section).
- **Framework JSON:** keys taken from sections {`FAIR`, `FAIR4ML`, `MLSEA`, `Croissant`, `PROV-O`}; the `Uncategorized` bucket is excluded.

Method (occurrence-based; exact string equality). A key matches only if the strings are exactly equal (case-sensitive).

Notation. K_{json} is the *multiset* of JSON keys with counts $j(k)$. U_{all} and U_{cls} are *baseline multisets* with counts $u_{\text{all}}(k)$ and $u_{\text{cls}}(k)$. We write $A \cap_{\text{occ}} B$ for the *occurrence-based* (multiset) intersection.

$$\text{Matches}_{\text{all}} = \left| K_{\text{json}} \cap_{\text{occ}} U_{\text{all}} \right|, \qquad \text{Completeness}_{\text{all}}(\%) = 100 \cdot \frac{\text{Matches}_{\text{all}}}{|U_{\text{all}}|}$$

$$\text{Matches}_{\text{cls}} = \left| K_{\text{json}} \cap_{\text{occ}} U_{\text{cls}} \right|, \qquad \text{Completeness}_{\text{cls}}(\%) = 100 \cdot \frac{\text{Matches}_{\text{cls}}}{|U_{\text{cls}}|}$$

Per-ontology completeness is computed by restricting U_{all} or U_{cls} to rows for that ontology before applying the same formulas.

5.2.5 Measurement 1: Metadata Coverage (U_{all})

How we measure it. We compute coverage using U_{all} as a baseline and compare it against case-sensitive ontology terms that appear among the JSON keys K_{json}.

Table 5.3 can be interpreted as follows, *Total fields* is the number of comparison keys we kept for each ontology in the full-union baseline (U_{all}). *Matched* counts exact, case-sensitive key matches found in `structured_metadata.json`. *Missing* = *Total* − *Matched*.

Completeness (%) = *Matched* / *Total*×100. The subset of the fields that matched and unmatched can be seen in Table 5.4, the complete set of these fields is archived in Zenodo. The evaluation script are in Zenodo.

Table 5.3: Coverage of K_{json} - structured_metadata.json on U_{all} - Full-Union

Ontology	Total fields	Matched	Missing	Completeness (%)
FAIR4ML	14	14	0	100.00
Croissant	30	21	9	70.00
MLSEA	110	74	36	67.27
FAIR	14	6	8	42.86
PROV-O	101	41	60	40.59

Overall (full union): total rows = 269; matched rows = 150; completeness = 55.76%.

Table 5.4: Subset of matched vs. unmatched ontology terms on full-union baseline.

Matched key	Unmatched key
dcterms:title	mls:Study
fair4ml:trainedOn	croissant:jsonQuery
fair4ml:validatedOn	mls:definedOn
mls:Dataset	mlso:hasMD5
mls:Experiment	mlso:hasScientificReference
mls:Model	schema:codeRepository
mls:hasHyperParameter	prov:influenced
mls:specifiedBy	prov:qualifiedAssociation
mlso:hasAlgorithmType	prov:qualifiedAttribution
mlso:hasFormat	prov:qualifiedGeneration
mlso:hasTrainTestSplitIndices	prov:qualifiedStart
prov:Activity	prov:value
prov:startedAtTime	—
prov:endedAtTime	—
prov:used	—
prov:wasGeneratedBy	—
—	—

"—" indicates that this table is a subset of the complete table.
The complete set of these fields is archived in Zenodo.

Conclusion (full union). The full-union check yields an overall completeness of **55.76%** (269 rows, 150 matches), with strong coverage for FAIR4ML (100%) and partial coverage for Croissant (70%) and MLSEA (67.27%), while PROV-O and

FAIR are lower. Many unmatched terms in these latter vocabularies are out of scope for notebook-native, tabular *classification* runs (e.g., deployment/audit-time or narrative annotation fields). To assess fit-for-purpose capture rather than breadth across all lifecycle stages, we now evaluate against a *strict, task-scoped subset* of U_{all}; all subsequent metrics (including value validity and automation) use that subset as the baseline.

5.2.6 Task-Scoped Subset for Classification U_{cls}

Rationale. A union of all ontology properties includes many that are irrelevant to notebook-native ML classification tasks (e.g., deployment, non-tabular modalities, regression, clustering fields). To avoid misleadingly low coverage, we define $U_{\text{cls}} \subset U_{\text{all}}$ via *explicit inclusion rules* aligned to this thesis scope.

Construction of Baseline subset U_{cls}:

Starting from $U_{\texttt{all}}$, full-union file we apply three inclusion passes:

1. *Prefix (generic) pass:* keep keys whose CURIE prefix is in {`prov`, `dcterms`, `croissant`, `fair`, `fair4ml`}and is generic metadata.

2. *Classification regex pass:* keep keys whose `key` or `label` matches a classification-oriented regex (e.g., `accuracy`, `precision`, `recall`, `f1`, `roc/auc`, `log_loss`, `confusion`, TP/FP/TN/FN, `multiclass/binary`, `threshold/probab`, `target/label`, `one-vs-rest`, `train_test/split`, `stratified`, `cross_val`).[2]

3. *Hard-include pass:* force-include core **MLS/MLSO/MLSEA** objects, relations, and classification metrics needed for pipelines/evaluation-even if they do not match the regex (e.g., `mls:Model`, `mls:Run`, `mls:Task`, `mls:EvaluationMeasure`, `mlso:hasDefaultTargetFeature`, `mlso:trainedOn`, `mlso:hasTrainTestSplitIndices`, `mlso:accuracy`, `mlso:precision`, `mlso:recall`, `mlso:f1Score`, `mlso:rocAuc`).

Deliverables. The script writes:

- `/relevant_union_generic_plus, archived in Zenodo _classification.csv` (**used as U_{cls} throughout RQ1**)

[2]Regex is applied case-insensitively to the concatenation of `key` and `label`.

Size. The total count, $|U_{\text{cls}}| = 214$ terms across five vocabularies.

Table 5.5: Per-ontology term counts within U_{cls}

Ontology	$\lvert U_{\mathbf{cls},o} \rvert$
PROV-O	94
FAIR	6
Croissant	29
FAIR4ML	14
MLSEA	71

Subset rules The table below samples concrete filter conditions implemented in the script (prefix/regex/hard-include).

Table 5.6: Selected filtering conditions used to build U_{cls}

Rule type	**Condition / Examples**
Prefix whitelist	{`prov`, `dcterms`, `dcat`, `croissant`, `fair`, `fair4ml`}
Regex (classification)	`classif`, `classification`
Regex (metrics)	`accuracy`, `precision`, `recall`, `f1`
Regex (curves)	`roc`, `auc`
Regex (loss)	`logloss`, `log_loss`
Regex (confusion)	`confusion` (matrix)
Regex (counts)	`true_positive`, `false_positive`, `true_negative`, `false_negative`
Regex (task type)	`multiclass`, `binary`
Regex (thresholds)	`threshold`, `probab`
Regex (targets)	`target`, `label`
Regex (strategy)	`one-vs-rest`
Regex (splits)	`train_test`, `split`, `stratified`
Regex (validation)	`cross_val`
Whitelist key	`mlso:hasDefaultTargetFeature`
Whitelist key	`mlso:hasTaskType`, `mlso:hasEvaluationMeasureType`
Whitelist key	`mlso:hasEvaluationProcedureType`
Whitelist key	`mlso:hasAlgorithmType`
Whitelist key	`mlso:hasRelatedImplementation`, `mlso:hasRelatedSoftware`
Whitelist key	`mlso:hasFormat`, `mlso:hasCacheFormat`
Whitelist key	`mlso:ignoresFeature`, `mlso:trainedOn`
Whitelist key	`mlso:hasTrainTestSplitIndices`
Hard-include (core MLS)	`mls:Model`, `mls:Run`, `mls:Task`
Hard-include (eval)	`mls:EvaluationMeasure`, `mls:EvaluationSpecification`, `mls:ModelEval`
Hard-include (structure)	`mls:Feature`, `mls:HyperParameter`, `mls:HyperParameterSetting`
Hard-include (pipeline)	`mls:Algorithm`, `mls:Implementation`
Hard-include (relations)	`mls:hasInput`, `mls:hasOutput`, `mls:hasPart`, `mls:specifiedBy`

Filtering code for the above is archived in Zenodo.

5.2.7 Measurement 2: Metadata Coverage (U_{cls})

How we measure it. We compute coverage: exactly the same way we did with U_{all}, but we use our strict-subset U_{cls} as a baseline and compare it against case-sensitive ontology terms that appear among the JSON keys K_{json}.

Table 5.7: Coverage on U_{all} and U_{cls} (per ontology counting)

	U_{all}			U_{cls}		
Ontology	$\lvert U_{\text{all},o}\rvert$	$\lvert M_o\rvert$	**Overlap (%)**	$\lvert U_{\text{cls},o}\rvert$	$\lvert M_o\rvert$	**Overlap (%)**
PROV-O	101	41	40.59	94	39	41.49
FAIR	14	6	42.86	6	6	100.00
Croissant	30	21	70.00	29	21	72.41
FAIR4ML	14	14	100.00	14	14	100.00
MLSEA	110	74	67.27	71	70	98.59

Overall (bag view): for U_{all}, total rows = 269, matched = 150, completeness = 55.76%; for U_{cls}, total rows = 214, matched = 150, completeness = 70.09%.

Table 5.8: Subset of matched vs. unmatched terms on U_{cls}.

Matched key	Unmatched key	Why unmatched
`croissant:citeAs`	`croissant:equivalentProperty`	Not emitted by pipeline
`croissant:column`	`croissant:excludes`	Not used in this dataset
`croissant:containedIn`	`croissant:extract`	Feature not implemented
`croissant:content`	`croissant:jsonPath`	Not used
`croissant:data`	`croissant:jsonQuery`	Not used
`croissant:dataType`	`croissant:parentField`	Structural; not exported
`croissant:delimiter`	`croissant:regex`	Validation not captured
`croissant:examples`	`croissant:subField`	Nested field not exported
`dcterms:license`	`dcterms:licence`	Spelling variant not used
`mls:Dataset`	`mls:DataCharacteristic`	Out of scope for run logs
`mls:EvaluationMeasure`	`mls:defines`	Modeling relation not used
`mls:Feature`	`mls:ImplementationCharacteristic`	Impl. details logged elsewhere
`mlso:hasAlgorithmType`	`mlso:hasDataLoaderLocation`	Data-loader not tracked
`mlso:hasDefaultTargetFeature`	`mlso:hasIdFeature`	ID feature not modeled
`mlso:hasEvaluationMeasureType`	`mlso:hasNumberOfReferences`	Citations recorded elsewhere
`mlso:hasTrainTestSplitIndices`	`mlso:hasPredictionsLocation`	Prediction files not archived
`fair4ml:evaluatedOn`	`prov:qualifiedInfluence`	PROV-O qualified pattern unused
`fair4ml:trainedOn`	`prov:qualifiedPrimarySource`	PROV-O qualified pattern unused
`dcat:Distribution`	`prov:atTime`	Timestamp stored elsewhere
`prov:Location`	`prov:Bundle`	PROV-O bundle not generated

Full, per-occurrence lists (matched and unmatched) for both U_{all} and U_{cls} are provided in : Zenodo.

Conclusion & Interpretation

Coverage under strict, case-sensitive, occurrence-based matching is 55.76% on the full union U_{all} (269 rows, 150 matches) versus 70.09% on the task-scoped subset U_{cls} (214 rows, same 150 matches). The lift is expected because U_{all} includes out-of-scope terms (deployment, non-tabular, etc.), while U_{cls} reflects the notebook-native

tabular classification setting.

Most unmatched items are (i) Croissant ETL fields we did not emit (e.g., `jsonPath`, `jsonQuery`, `regex`), (ii) PROV-O qualified patterns unused in our runs (e.g., `prov:qualifiedInfluence`, `prov:Bundle`), and (iii) benign alias/spelling variants (e.g., `dcterms:licence` vs `dcterms:license`) among which one matched. These gaps mainly reflect scope and exporter choices; adding optional PROV-O qualifiers and a small alias map would raise coverage.

5.2.8 Measurement 3: Ontology-aligned vs. Additional metadata)

What we report. Besides coverage (how many *ontology* terms we hit exactly), we also log extra metadata that is useful in practice (e.g., experiment notes, environment/version details) but not always named in the ontologies. Prior work argues for such richer documentation to support reproducibility and responsible use (e.g., Model Cards, Datasheets for Datasets, and the NeurIPS Reproducibility Checklist) [32, 33, 34]. Consistent with metadata alignment research such as the book *Proceedings of the 10th International Conference on Biomedical Ontology*[40], which treats non-exact or contextually related terms separately from exact ontology matches, our pipeline classifies any key not found in the reference ontology term lists as a *reporting extra*. These metadata are not considered in the coverage calculation.

How we count. We split all JSON keys into two buckets: (i) *Ontology-aligned*—exact, case-sensitive matches to terms in the baseline; and (ii) *Reporting extras*—present in our JSON, but not defined in the baseline ontologies. We report both, so readers see how much is standards-based versus additional, but still important or relevant to the context.

Table 5.9: Ontology alignment and Extra metadata (JSON-side counts)

Category	**Count**	**Share (%)**
Ontology-aligned (exact matches)	*150*	*65.79*
Reporting extras (not in ontology)	*78*	*34.21*
Total JSON keys	*228*	100.00

Table 5.10 lists a representative subset of "framework extras" that are not part of the

baseline ontologies but are retained because they materially improve reproducibility, traceability, and interpretability. Some values are intentionally repeated under simplified key names to improve clarity and usability.

Table 5.10: Representative extra (non-ontology) metadata fields and their practical value.

Extra key	Practical value
`prov:commit`,`prov:commitAuthor`, `prov:commitTime`	Capture the exact code state and authorship at execution time, enabling precise version tracing.
`prov:pythonVersion`, `prov:osPlatform`	Record runtime environment details to guard against platform-dependent differences.
`prov:usedNotebook`	Links results to the specific interactive environment in which they were produced.
`mlso:justificationModelChoice`, `mlso:justificationTestSplit`	Record the rationale for model and parameter choices, supporting transparency and responsible use.
`mlso:modelPath`	Points to the stored model artefact for verification or reuse.
`croissant:examples.*`	Provide concrete sample data values for interpretability and quality checking.

The full set of extras is archived in Zenodo.

5.2.9 Measurement 4: Degree of Automation

Operational definitions.

- **Automatic** - emitted without user action (environment, timestamps, VCS/git, metrics, file hashes, dataset IDs).
- **Semi-automatic** - system prompts with bounded user input (e.g., dropdowns, choices).
- **Manual** - explicit user input (rationales, free-text descriptors).

Each property in U_{cls} is tagged with one of the three, as implemented in Section 4.3. We report overall distributions.

Table 5.11: Automation distribution over U_{cls}

Type	Count	Percent
Automatic	183	80.26
Semi-automatic	15	6.58
Manual	30	13.16

Table 5.12: Representative fields with capture mode and input source.

Section	Key	Input Type	Input Source
PROV-O	`prov:commit`	Automatic	git
PROV-O	`prov:commitAuthor`	Automatic	git
PROV-O	`prov:commitTime`	Automatic	git
PROV-O	`prov:pythonVersion`	Automatic	mlflow
PROV-O	`prov:osPlatform`	Automatic	mlflow
PROV-O	`prov:startedAtTime`	Automatic	mlflow
PROV-O	`prov:endedAtTime`	Automatic	mlflow
PROV-O	`prov:usedNotebook`	Semi-Automatic	jupyter
MLSEA	`mls:HyperParameter`	Semi-Automatic	user
MLSEA	`mlso:hasTrainTestSplitIndices`	Semi-Automatic	user
MLSEA	`mlso:trainAccuracy`	Automatic	mlflow
FAIR4ML	`fair4ml:intendedUse`	Manual	user
FAIR4ML	`fair4ml:modelCategory`	Semi-Automatic	mlflow
FAIR4ML	`fair4ml:evaluatedOn`	Automatic	dbrepo
FAIR4ML	`fair4ml:runEnvironment`	Automatic	mlflow
Croissant	`croissant:format`	Automatic	mlflow
FAIR	`dcterms:license`	Semi-Automatic	doi
FAIR	`dcterms:creator`	Semi-Automatic	dbrepo

Table 5.12 lists a representative subset of metadata fields, showing their capture mode and primary input source. This selection mixes Automatic, Semi-Automatic, and Manual fields from diverse origins such as *git*, *mlflow*, *user* input, *dbrepo*, *doi*, and *jupyter* notebooksis archived in Zenodo.

Conclusion. The RQ1 evaluation establishes, with quantitative evidence, that the framework achieves comprehensive provenance capture in alignment with established semantic vocabularies while remaining precisely scoped to the requirements of notebook-native, tabular classification workflows. By constructing a reproducible, ontology union from official releases, and then refining it to a task-scoped subset U_{cls}, the analysis isolates the fields that are truly relevant to this experimental context. The resulting 70.09,% completeness on U_{cls}, compared to 55.76,% on the full union U_{all}, demonstrates that the observed coverage gaps are largely due to out-of-scope lifecycle terms rather than deficiencies in capture. In theory, the few more

fields could be mapped if redundancy isnt a concern. The clear separation between ontology-aligned fields and "reporting extras" further ensures that the framework delivers both formal standards compliance and the additional operational metadata required for reproducibility, interpretability, and responsible model governance.

The automation analysis underscores the operational maturity of the approach: over 80% of all U_{cls} fields are captured automatically from trusted sources such as *git*, *mlflow*, *dbrepo*, *doi* registries, and *jupyter* notebooks, with the remaining fields handled through structured prompts or targeted manual input. This degree of automation minimises user burden and reduces the risk of omission, while retaining intentional control over ethically and contextually sensitive descriptors. Through rigorous ontology alignment, precise scope refinement, and automation-driven capture, the framework delivers a robust, provenance-aware solution that generates rich, standards-compliant metadata with minimal operational overhead.

5.3 RQ2: Reproducibility, Debugging, and Version Auditing

5.3.1 Use Case: Provenance and Reproducibility

Objective: Analyze if the information that the framework collects can be used to accurately replicate a previous machine learning run, and verify whether the dashboard tools can be used to track out any inconsistencies.

Method: For thorough traceability testing, `Wine_Evaluation_v20250812_233212`, a previously conducted experiment using the Wine dataset and a Logistic Regression model, was selected. During the user study, each participant was assigned to reproduce the other participant's previously executed run, using only the reproducibility guide and the metadata available in the dashboard. The reproducibility guide (Figure 5.5) is a plain-text file automatically generated by the framework that lists all parameters, dataset identifiers, preprocessing steps, model hyperparameters, and Git commit details required to rerun the experiment outside the dashboard. The aim was to verify that the dashboard's comparison features could highlight any differences in model setup or data provenance, and to check whether the same per-

formance metrics (e.g., accuracy, confusion matrix) could be replicated.

- The "Provenance Trace" tab on the dashboard interface (Figure 5.1, Figure 5.2) allowed the user to select the original and reproduced runs from a dropdown menu for comparison.
- The system retrieved all metadata: dataset DOI, schema version, preprocessing parameters (e.g., dropped columns, imputation strategy), model type, hyperparameters (e.g., `max_depth`, `n_estimators`), Git commit hash, and execution timestamps.
- Field-level differences were visualized in two tabular panels: one for provenance and configuration metadata (Figure 5.3), the other for model evaluation metrics and runtime (Figure 5.4). Color-coded highlights indicated mismatches across runs.
- A reproducibility guide (Figure 5.5) was auto-generated as a downloadable file, listing all parameters, dataset details, preprocessing steps, and model settings needed for an external rerun.

Figure Interpretation and Outcome:

- **Figure 5.1:** The *Provenance Trace* tab allows selection of a training run for detailed metadata inspection. Box #1 highlights the run selection dropdown, where users can choose a single run or enable the `Compare with another run` option to perform side-by-side comparison. In this example, Run 5 (`Wine_Evaluation_v20250812_233212`) was chosen as the original experiment, and Run 9 (`Wine_Evaluation_Repod_v20250812_233922`) was the independently re-executed reproduction. Selecting these runs triggers the retrieval of all related provenance metadata, including dataset identifier, version, model type, preprocessing parameters, runtime settings, and Git commit details, which will be compared in subsequent tables. Any differing values between the two runs are highlighted by the interface using a consistent color scheme to support visual identification of divergences.

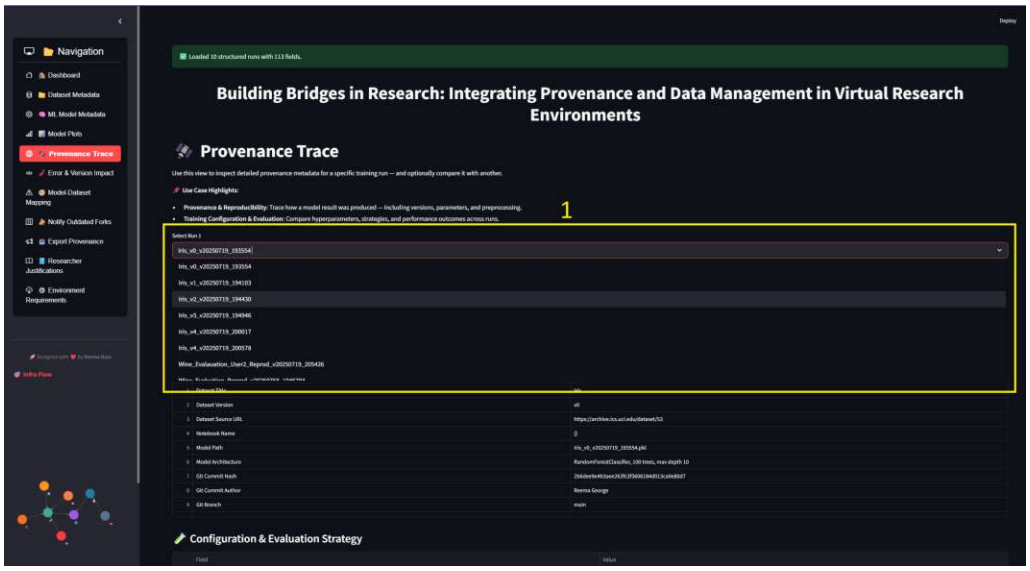


Figure 5.1: Dashboard interface for provenance trace. Run selection via a dropdown list.

- **Figure 5.2:** The activated comparison view in the *Provenance Trace* tab, shown after two runs have been selected. Box#**2** contains the controls for selecting a second run and downloading the reproducibility guide. The reproducibility guide is a plain-text file with all dataset, preprocessing, model, and Git commit details needed to reproduce the selected run outside the dashboard. Box#**3** displays the two structured comparison tables: *Provenance and Reproducibility Details* (top) lists metadata fields such as Git commit author, branch, commit time, repository URL, preprocessing timestamps, and dataset information. *Configuration and Evaluation Strategy* (bottom) shows target variable, split strategy, model architecture, serialization format, model version, and evaluation metrics. Cells highlighted in pink indicate differences between the selected runs (e.g., timestamps, model version tag), while unshaded cells indicate identical values. This view also supports search and filter operations for targeted inspection, and both tables can be exported as CSV for offline auditing or documentation.

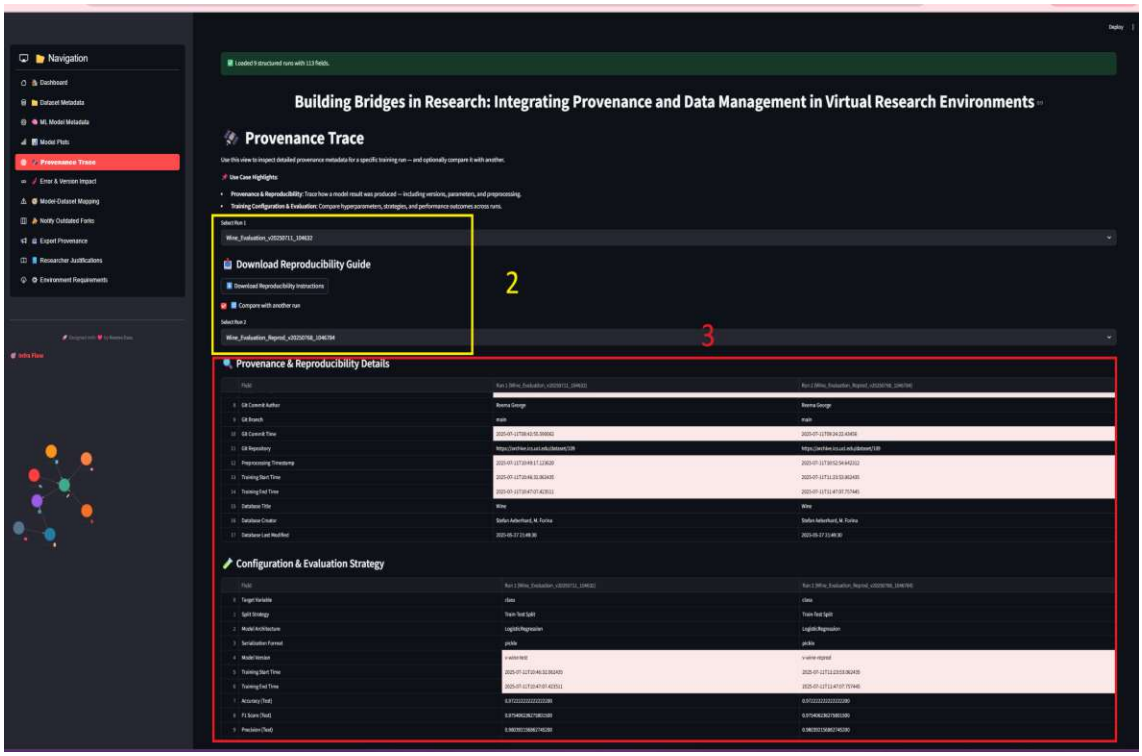


Figure 5.2: Dashboard interface for provenance trace. Two runs can be compared by clicking the tick box.

- **Figure 5.3:** The **Provenance and Reproducibility Details** table compares metadata for the two selected runs. Cells shaded in pink are automatically highlighted by the dashboard to indicate differences between runs, such as:
  - `Run ID` - unique to each execution.
  - `Git commit hash` - differs because the reproduction run was executed from a different commit state.
  - `Git commit time` - reflects when the commit was made in each case.
  - `Preprocessing timestamp` and `Training start/end time` - show when preprocessing and training occurred in each run.

  Fields that remained identical across both runs are outlined in red in the Figure 5.3. These include `Dataset Title`, `Dataset Version`, `Dataset Source URL`, `Model Path`, `Model Architecture`, `Git Commit Author`, `Git Branch`, `Git Repository`, `Database Title`, `Database Creator`, and `Database Last Modified`. The unchanged values confirm that the dataset, configuration, and

repository context were preserved between the original and reproduced runs, while the pink-shaded cells reflect changes arising from independent execution events.

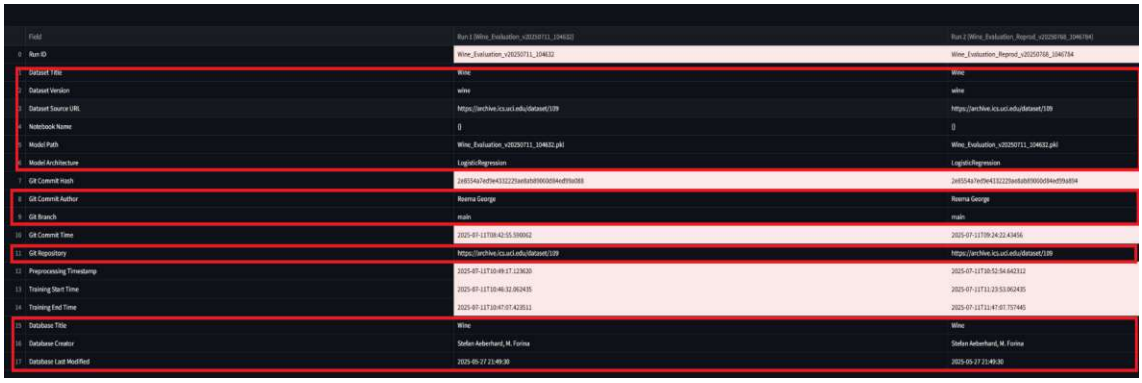


Figure 5.3: Provenance and Reproducibility Details table.

- **Figure 5.4:** The **Configuration and Evaluation Strategy** table presents a side-by-side comparison of two runs. Variations in selected metadata columns, such as `Model Version`, `Preprocessing Timestamp`, and `Training Start/End Time`, are highlighted in the unboxed rows. The difference in `Model Version` is intentional, as it reflects the user-defined Git tag provided during experiment setup (e.g., `v-wine-test` vs. `v-wine-reprod`), confirming these are separate runs executed within the same experimental structure.

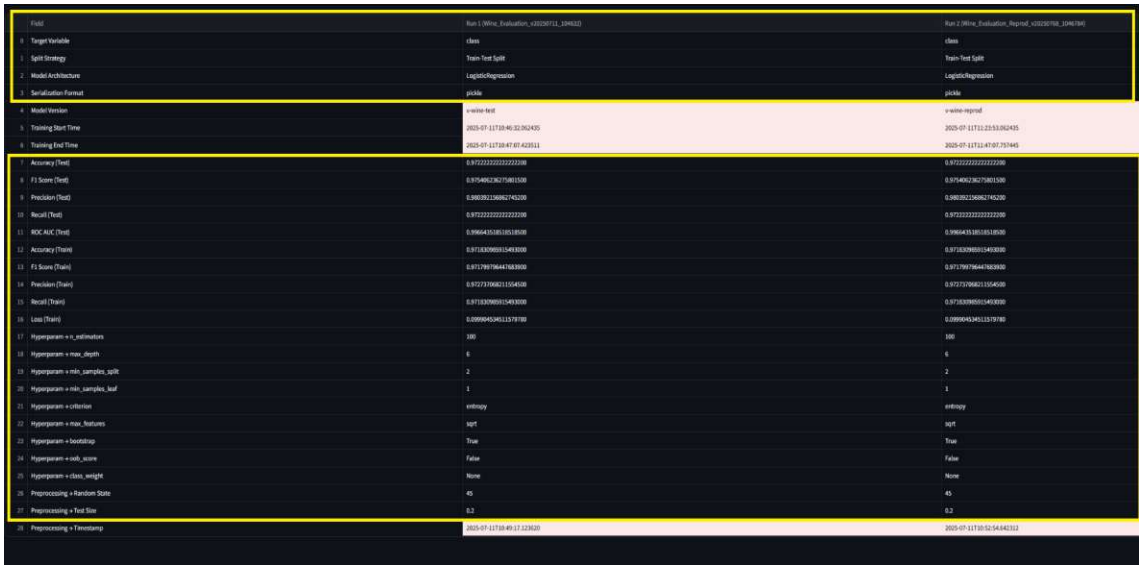


Figure 5.4: Configuration and Evaluation Strategy table with yellow boxes indicating unchanged fields across runs.

The yellow boxes indicate clusters of metadata that remained identical between

runs, providing strong *evidence* of repeatability. These include:

- Core configuration: `Target Variable`, `Split Strategy`, and `Serialization Format`.
- Evaluation metrics (test): `Accuracy (0.9722)`, `F1 Score (0.975)`, `Precision (0.980)`, `Recall (0.9722)`, and `ROC AUC (0.996)`.
- Evaluation metrics (train): `Accuracy (0.971)`, `F1 Score (0.971)`, `Precision (0.972)`, and `Recall (0.971)`.
- Hyperparameters: `n_estimators = 100`, `max_depth = 6`, `min_samples_split = 2`, `min_samples_leaf = 1`, `criterion = "entropy"`, `max_features = sqrt`, `bootstrap = True`, `oob_score = False`, `class_weight = None`, `random_state = 45`, and `test_size = 0.2`.

This degree of consistency supports the claim that experiments can be reliably repeated within the framework using the structured JSON metadata. *However*, this evidence is conditional on the computational environment remaining stable-changes to operating system, library versions or defaults, or hardware precision (e.g., single- vs. double-precision GPUs) can still affect results in ways not visible in this table.

- **Figure 5.5:** The plain-text file lists the exact combination of parameters used in Run #5(`Wine_Evaluation_v20250711_104632`). It was used along with the dashboard metadata to reproduce the experiment.

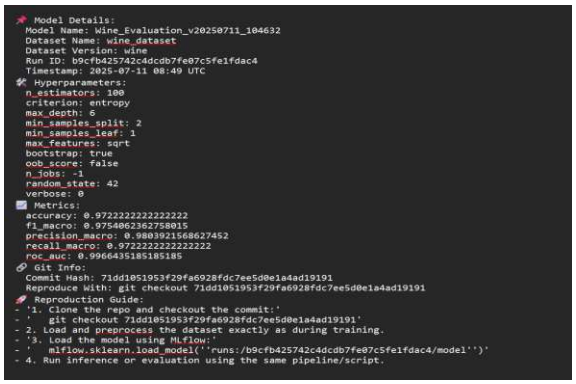


Figure 5.5: Reproducibility guide auto-generated from captured metadata, used to re-run the experiment externally.

Outcome: The framework's capacity to support consistent model reproduction is evidenced by the outcomes of this reproducibility test, which included participant-led repeats during the user study. The auto-generated reproducibility guide and the comparison tables together provided sufficient metadata to repeat the procedure under the same software and hardware conditions, resulting in matching assessment metrics. Specifically:

- **Provenance metadata is comprehensive and structured:** Key inputs such as runtime settings, model parameters, preprocessing setup, dataset version, and Git state were correctly logged and accessible in a structured format.
- **Side-by-side comparison enables effective debugging:** Field-level variations between two executions were made explicit through visual highlighting and semantic grouping, enabling quick identification of configuration or data differences.
- **Evaluation metrics and hyperparameters were consistently reproduced:** Training and test measures (accuracy, F1-score, precision, recall, and ROC AUC) and all recorded hyperparameters matched across runs performed in an equivalent execution environment.

These results indicate that the system captures the provenance metadata needed to repeat experiments when the computational environment (OS, library versions, hardware, and accelerator configurations) is held constant, and presents it in a way that facilitates replication. A detailed mapping of user study findings to each research question is provided in Section 5.6.

Limitations:

- **Scope of environment capture:** While software configurations, dataset versions, and Git states are logged, deeper environment layers-such as OS library versions, hardware specifications, GPU/TPU accelerator details, or low-level architectural differences-are not fully captured. Subtle changes at these levels, as well as implicit heuristics in some ML frameworks, may affect reproducibility despite identical recorded metadata.

– **Dashboard Structuring:** Although the dashboard effectively supports provenance inspection and repeatability, some metadata values appear in multiple views. Usability could be improved by de-duplicating values and refining the layout for reduced visual clutter.

5.3.2 Use Case: Experiment Versioning and Error Tracing

Objective: Determine which machine learning experiments used inaccurate or out-of-date dataset or model versions, evaluate the effect on model validity, and alert the responsible experimenters to initiate repetitions with updated versions.

Context and Motivation: Dataset versions may be changed over time in collaborative machine learning processes, for instance, when schemas are modified, incorrect items are eliminated, or missing information is fixed. The results of previous tests become untrustworthy if they were carried out with a faulty or corrupted dataset version. Such a situation is simulated in this use case: A purposefully distorted dataset version with missing values and inaccurate labels, `Iris_v4` was published as part of a test phase during development. Verifying that the framework can track all experiments based on this version, alert the relevant machine learning practitioners, and facilitate mistake rectification procedures are the objectives.

Method: The *Error & Version Impact Analysis* tab renders a curated table of *tracked runs currently loaded in the UI*. Users enter a deprecated tag (e.g., `iris_v4`); the UI filters the on-screen table and highlights matching rows, after which an optional GitHub notification can be issued.

Each run entry displays:

– **Run ID** — unique run identifier from MLflow, enabling precise traceback.

– **Git Commit** — commit SHA captured at run time, serving as a provenance anchor.

- **Version Tag** — user-entered code/data tag (e.g., `iris_v4`).
- **Model** — model name along with key hyperparameters for quick comparison.
- **Accuracy** — test-set score logged by the run.
- **Dataset** — dataset label and version used in the run.

The problematic dataset version `iris_v4` was detected by the author by entering it into the input box marked `Enter deprecated version tags`. Upon clicking `Detect Impacted Runs`, the system filtered and highlighted all matching runs (Figure 5.6).

1. A visual warning banner indicating the number of affected runs was automatically shown by the dashboard (Figure 5.7). To put the possible impact in context, these runs contained crucial metadata such as the dataset name, model accuracy, and Git hash.
2. The `Notify Affected Users via GitHub` functionality was optionally activated by the creator. A form with the GitHub owner and repository information prefilled opened. The form was submitted once the author input a GitHub token. Based on email-to-GitHub mapping, the system then tagged contributors connected to the impacted run and issued an issue on the appropriate GitHub repository. The problem was clearly described, along with its cause (such as a flawed dataset) and suggested solution (Figures 5.8 and 5.9).

Figure Interpretation:

- **Figure 5.6:** Displays the *Error & Version Impact Analysis* tab's initial interface. Users may view the whole table of previous experiments underneath the top part, which shows the most current Git commit details from the most recent run.
- **Figure 5.7:** Shows every historical run's complete summary table with metadata. It is evident that `iris_v4`, a known flawed version, is used in Run #4 and #5. This makes it possible to manually examine possibly tainted runs prior to starting any filtering.

– **Figure 5.8:** The system filters and shows just the impacted runs when you put `iris_v4` as a deprecated version tag. The GitHub notification form is displayed for additional action beneath the visual warning bar.

– **Figure 5.9:** End result of the alert procedure. This image displays a GitHub issue that the system generated on its own. It enables prompt resolution by explicitly documenting the version issue and tagging the experimenter in charge of the impacted run.

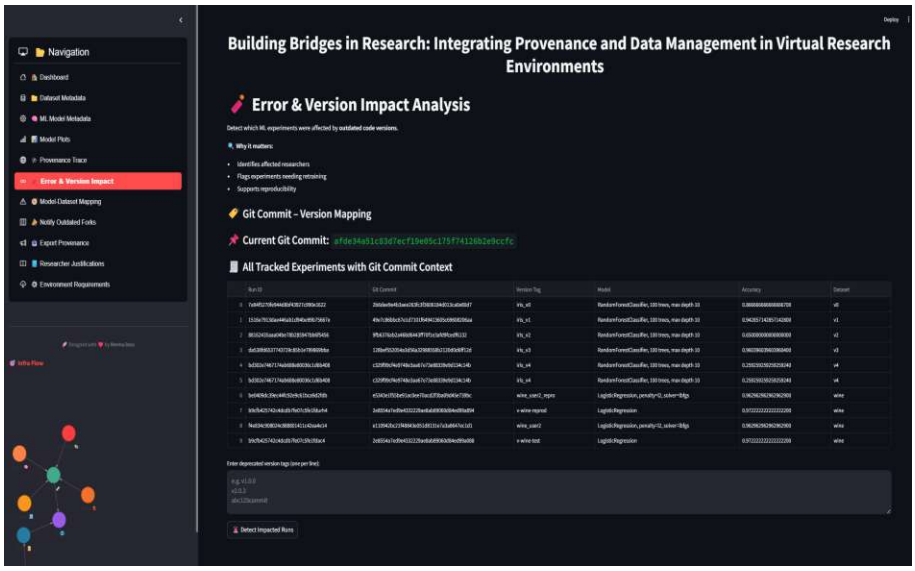


Figure 5.6: Error and Version Impact Analysis Dashboard.

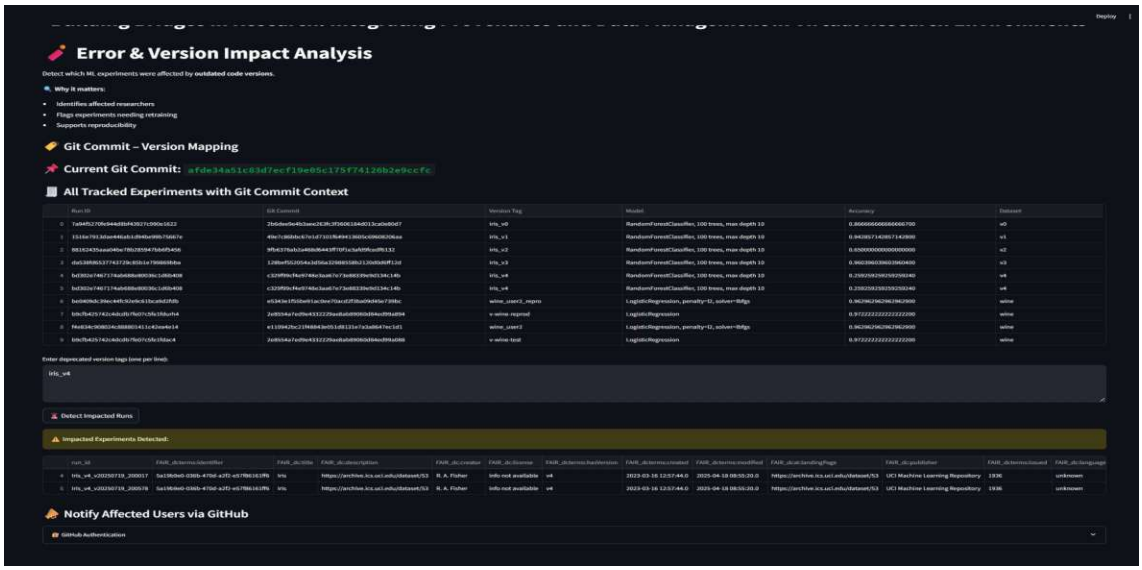


Figure 5.7: Detected faulty runs

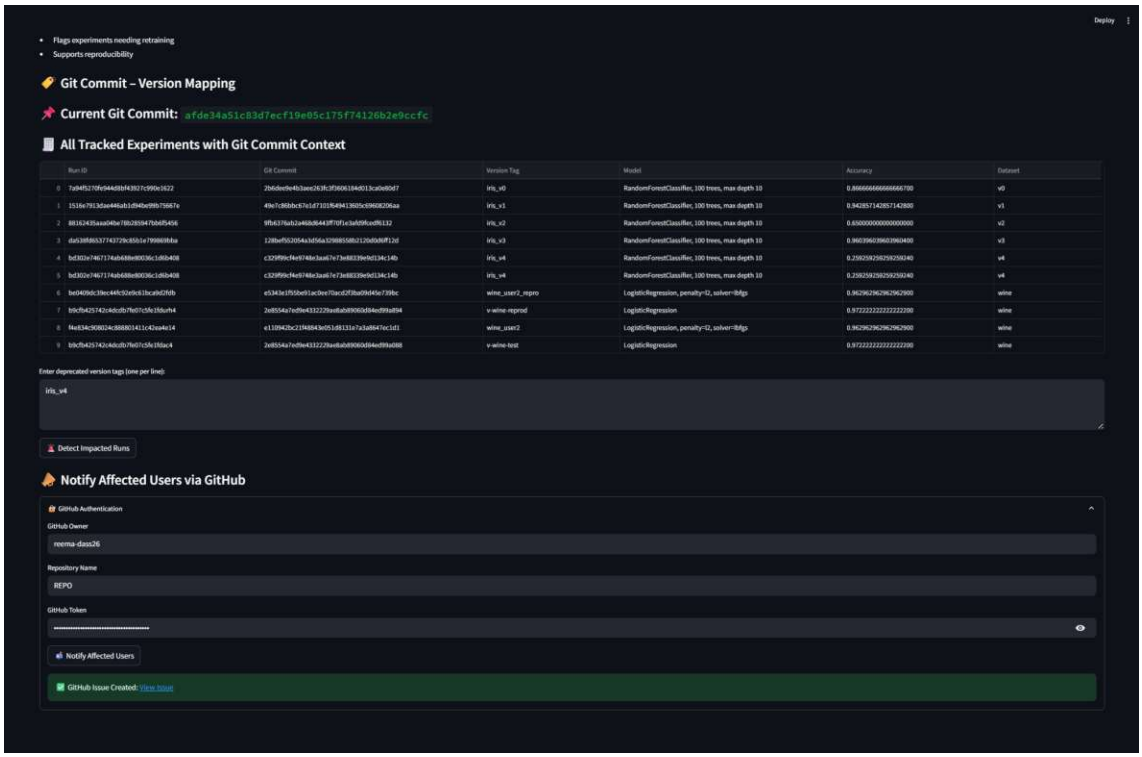


Figure 5.8: Notifying the Collaborator

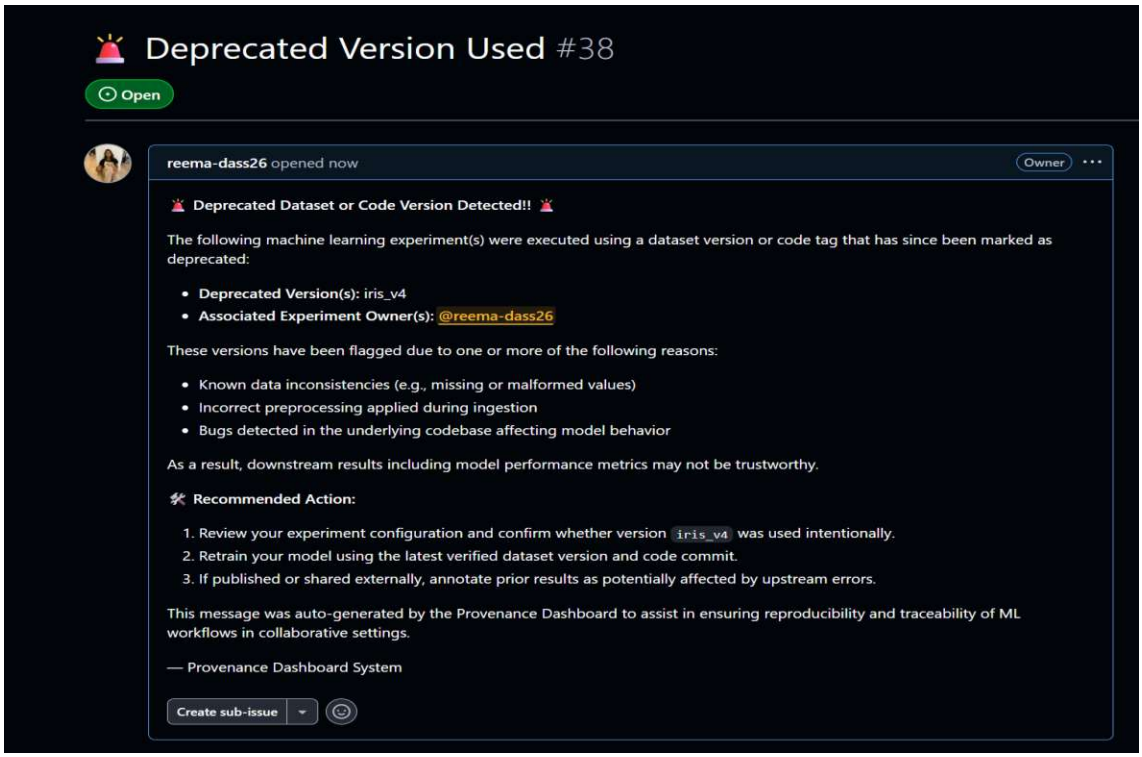


Figure 5.9: GitHub Issue notification

Outcome:

- Runs using the faulty dataset version `iris_v4` were correctly identified and flagged.
- The filtering process enabled quick detection of affected experiments.
- GitHub-based notifications alerted responsible collaborators for follow-up. All the required information needed for this step was retrieved from the saved metadata.
- The feature supports metadata-driven governance and prevents propagation of invalid results.

A detailed mapping of user study findings to each research question is provided in Section 5.6.

5.3.3 Use Case: Configuration and Evaluation Tracking

Task: Examine the effects of training setups and hyperparameters (such as test size, random seed, and model depth) on the results of model performance over several runs.

Method: For every training run, the *Model Metadata* tab of the dashboard offers an organized view of the configuration and evaluation metadata. When users choose a run from a drop-down menu, four structured tables representing various metadata categories are dynamically loaded:

- **Model Overview Table:** Box#**1** Includes fields such as `Model Name`, `Algorithm`, `Architecture`, `Serialization Format`, `Target Variable`, `Label Encoding`, `Model Path`, and `Model Version`. This table gives users a high-level understanding of what model was used, how it was serialized, and what label configuration was applied.
- **Hyperparameters Table:** Box#**2** Lists configuration values such as `n_estimators`, `criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_features`, `bootstrap`, `oob_score`, `class_weight`, `verbose`, and `n_jobs`. These values provide insight into model tuning choices and complexity.

- **Test Metrics Table:** Box#**3** Shows evaluation results on the test set, including `Accuracy`, `F1 Score`, `Precision`, `Recall`, and `ROC AUC`. These scores help assess model generalization under the current configuration.
- **Training Metrics Table:** Box#**4** Includes `Training Accuracy Score`, `Training F1 Score`, `Training Precision Score`, `Training Recall Score`, and `Training ROC AUC`, giving insight into possible overfitting or underfitting based on performance deltas.

After choosing a particular run from the dropdown menu at the tab's top, each of these tables is created dynamically. This enables users to identify model configurations, compare trials and monitor the impact of altering training approach or hyperparameters on assessment metrics downstream.

Outcome:

- The dashboard enabled Figure 5.10 users to observe the influence of parameters like `max_depth` and `min_samples_split` on model performance.
- For example, in Run `Iris_v0_v20250719_193554`, the model used `RandomFores tClassifier` with `100` trees, `max_depth=10`, and `min_samples_split=3`, achieving a test accuracy of `0.867` and ROC AUC of `0.979`.
- Training metrics were perfect (e.g., `1.0` for accuracy, precision, recall), while test metrics were lower - suggesting mild overfitting.
- Users might find hyperparameter selections that enhanced generalization or decreased overfitting by comparing several runs using the dropdown menu, facilitating experimental refining and performance optimization.

Implications: Users may associate certain hyperparameter values with performance differences by separating configuration, test, and training data into organized views. This offers detailed insight into the link between setup and performance, which aids with explainability, repeatability, and troubleshooting. A detailed mapping of user study findings to each research question is provided in Section 5.6.

Limitation: Although high-level metrics are displayed, this view does not yet incorporate deeper interpretability insights (such as confusion matrix analysis or SHAP ratings).

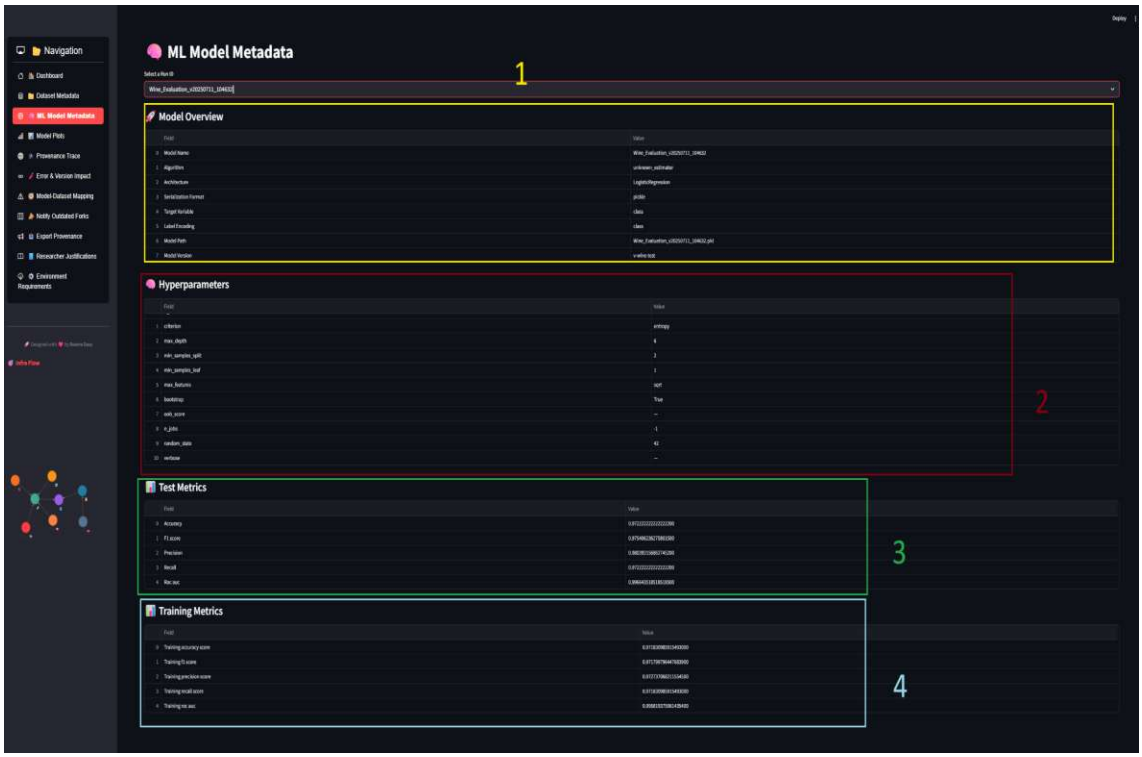


Figure 5.10: Tabular view showing the model metadata overview.

5.3.4 Use Case: Model-Data Relationship Mapping

Task: Examine how changes in input data affect assessment metrics and track a model's behavior over several datasets or dataset revisions.

Method: The *Model-Dataset Mapping* tab of the dashboard offers a consolidated table that documents the test results of each model training run and associates it with the relevant dataset version. This table allows users to examine sensitivity to dataset changes, compare findings across dataset versions, and study model generalization Box# in Figure 5.11.

The table contains the following fields:

- `Run ID` – Unique identifier for each training run.
- `Model Name` – Name of the model used in the run.

- `Architecture` – Full string representation of the model architecture (e.g., "RandomForestClassifier, 100 trees, max depth 10").
- `Dataset Title` – Human-readable title of the dataset.
- `Dataset Version` – The internal or semantic version identifier for the dataset used.
- `Dataset Access URL` – Resolvable URI for dataset access or citation.
- `Accuracy`, `Recall`, `Precision`, `ROC AUC` – Key evaluation metrics computed on the test set.

This view Figure 5.11 supports a comparative study of how models trained with identical architectures behave across varying dataset versions (e.g., `iris_v0`, `iris_v1`, `iris_v2`). 0.94, 0.86, 0.65, accuracy, respectively.

Outcome:

- The same model architecture (RandomForestClassifier, 100 trees, max depth 10) exhibited performance variation across dataset versions:
  - On `iris_v1`, the model achieved **Accuracy = 0.943**, **F1 Score ≈ 0.944**, and **ROC AUC = 0.995**.
  - On `iris_v0`, performance dropped slightly to **Accuracy = 0.867**, **F1 Score ≈ 0.852**, and **ROC AUC = 0.979**.
  - On the altered version `iris_v2`, results were significantly lower: **Accuracy = 0.650**, **F1 Score ≈ 0.583**, and **ROC AUC = 0.735**.
- These variations show how the framework may reveal how sensitive the model is to changes in the dataset, facilitating traceability and a reliable assessment of model generalization.
- Researchers may support their claims about whether a model needs to be retrained or is suitable for deployment on a new dataset variation when versioned dataset information and associated performance ratings are present.

Implications: This tab enables horizontal analysis by showing how the same model architecture behaves across different dataset versions, making it

straightforward to spot both expected and unexpected performance shifts. For example, a small drop in accuracy when moving from `iris_v1` to `iris_v0` may be expected if the dataset differs only slightly, whereas a substantial performance drop on `iris_v2` signals an unexpected change that may warrant retraining or deeper inspection of the dataset quality. By clearly associating performance changes with specific dataset versions, the tab supports decisions about retraining, transfer learning, or targeted data curation, and helps identify cases where the model fails to generalize as intended.

Limitation: The dashboard's analysis is now restricted to this tab; if more information is needed, it must be accessed through other tabs.

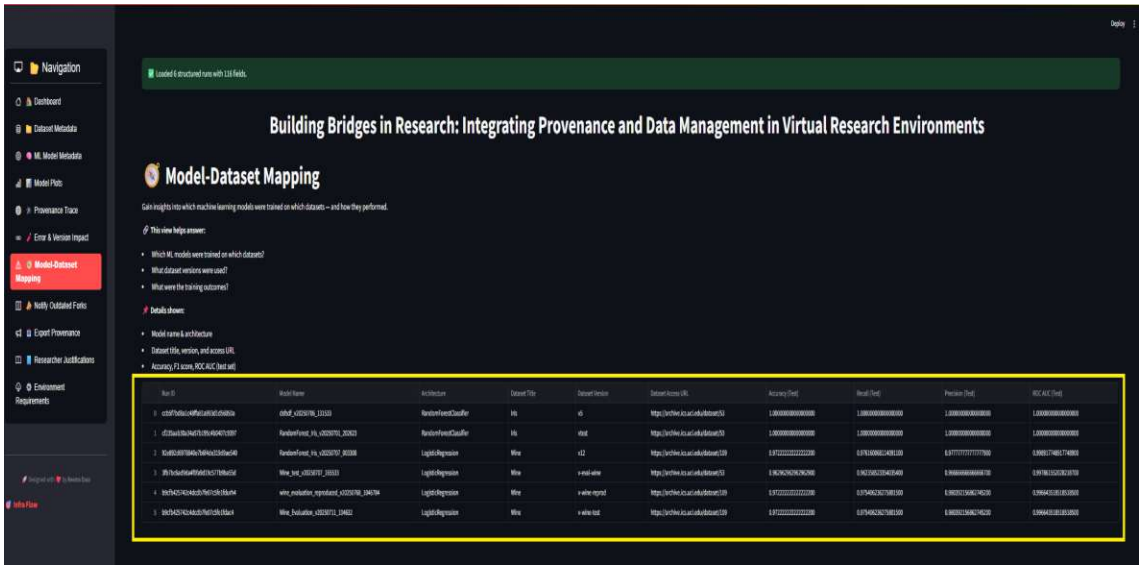


Figure 5.11: Tabular view showing the mapping between ML models and datasets.

5.3.5 Use Case: Repository Fork Awareness

Problem: In collaborative ML workflows, experiment reproducibility can be compromised when collaborators run experiments from outdated or diverged code forks. If a fork lags behind the main repository, model behaviour, preprocessing logic, or data handling scripts may differ from the intended baseline, even if the same dataset and hyperparameters are used. To ensure that experiment provenance remains trustworthy, the system must detect such divergences and alert affected collaborators (see Use Case 4.5) .

Task: Detect and highlight divergences in experiment provenance caused by code version drifts between branches.

Method: The system checks for commit mismatches between the fork and the main repository by comparing the commit hash recorded in the current run against the latest commit reference of the main branch. When a mismatch is detected, a GitHub issue is automatically created using the user-provided personal access token, tagging the repository owner of the outdated fork.

Outcome:

- Forked runs with stale or divergent code were identified via Git commit mismatches (Figure 5.12).
- Users were notified through a GitHub issue tagging mechanism (Figure 5.13).

A detailed mapping of user study findings to each research question is provided in Section 5.6.

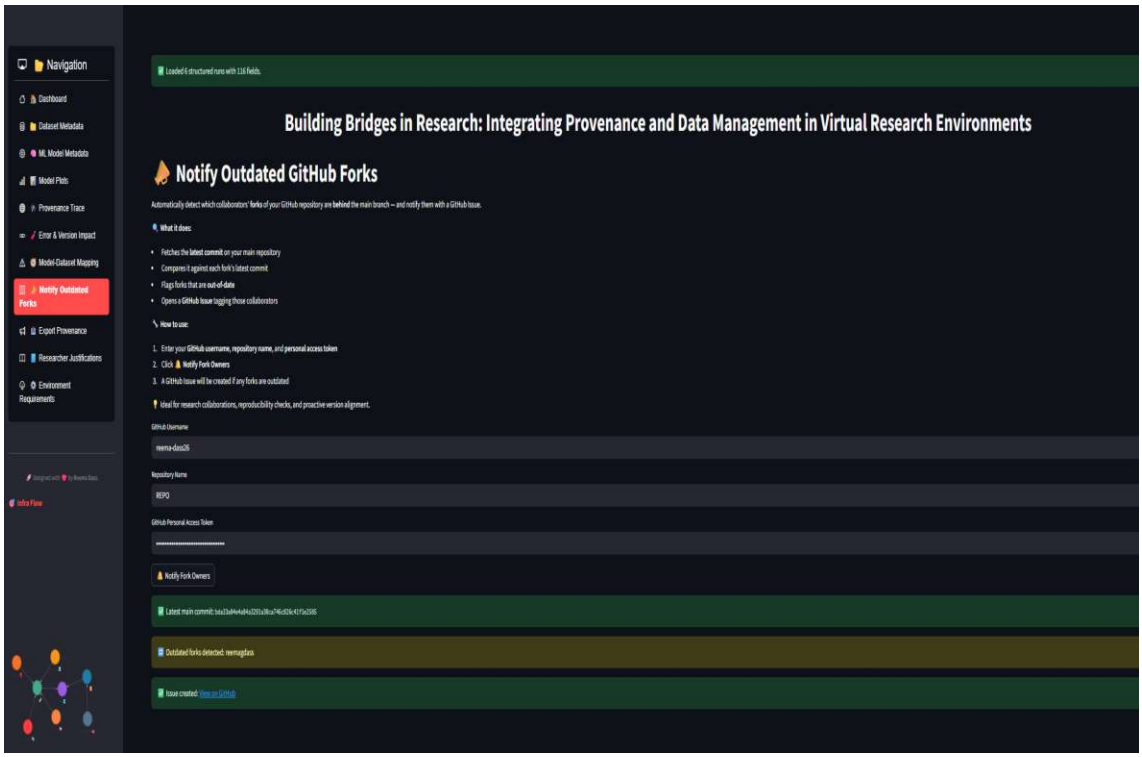


Figure 5.12: UI snippet displaying fork divergence detection.

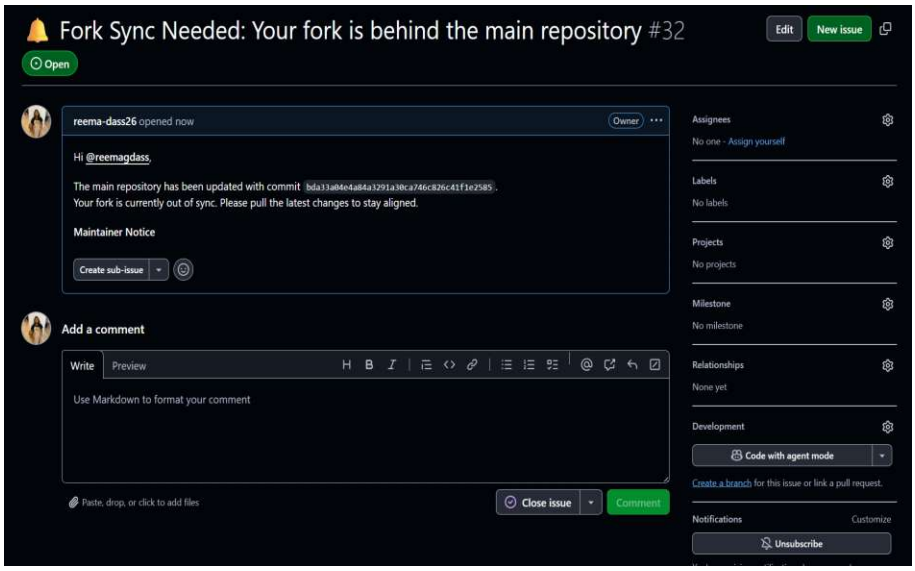


Figure 5.13: GitHub issue created tagging the collaborator who has the outdated fork

5.4 RQ3: Usability and Visualization Effectiveness

This section reports findings from a small, exploratory user study conducted with two ML practitioners. The aim was to gather preliminary design insights on dashboard usability and visualization clarity, rather than to perform a statistically powered evaluation. The study setup involved each participant using the framework's dashboard to complete guided interpretation and provenance inspection tasks. Given the limited scale and qualitative nature, the results are intended to inform design refinement and confirm basic usability, not to provide conclusive quantitative metrics.

5.4.1 Dashboard Evaluation

Objective: Evaluate whether the dashboard supports user understanding, error tracing, and decision justification in provenance-rich ML workflows.

Method: Participants were asked to explore the dashboard (Figure 5.14) and perform specific interpretation tasks, such as:

- Reviewing summaries of all the dashboard tabs and associated metrics.
- Inspecting visualizations of model performance across datasets (e.g., ROC AUC).
- Interpreting captured justifications tied to preprocessing and model selection.

Observations:

- The summary panel helped participants orient well with the dashboard layout and its purpose.
- Plot-based visualizations (Figure 5.15) effectively communicated performance variation. This tab also provided the subset of important metadata based on the run selection for context alignment.
- The embedded justification log (Figure 5.16) was recognized as increasing transparency, especially in understanding why certain hyperparameters or models were chosen.
- Requirements tabs (Figure 5.17) reduced onboarding time by clearly stating the expected components for each run.
- Other views (e.g., Provenance Trace, Export Explorer) are separately evaluated in RQ1, RQ2, and RQ4.

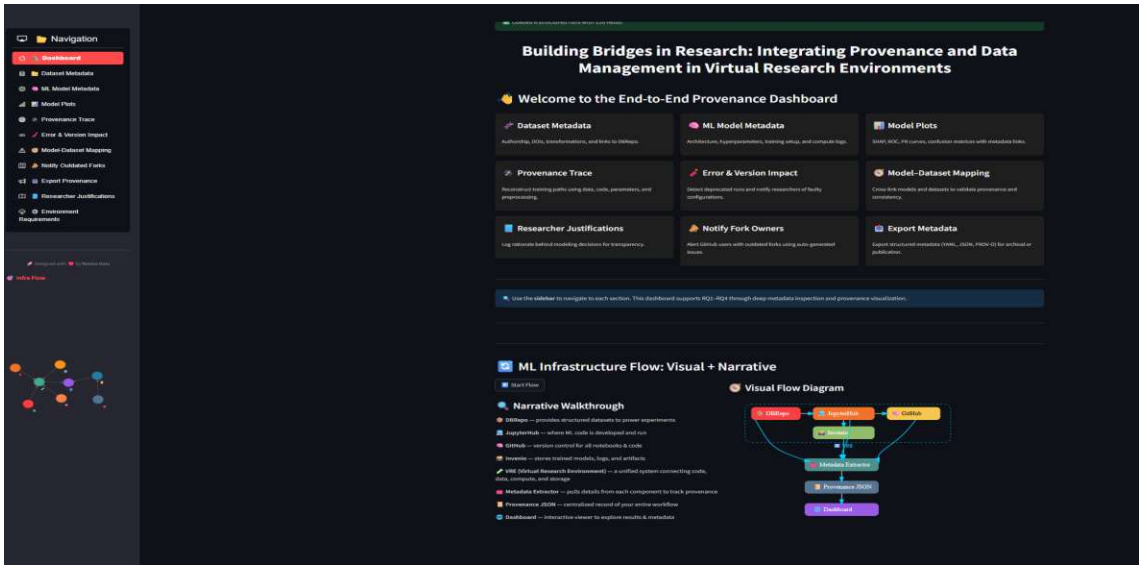


Figure 5.14: Dashboard summary panel.

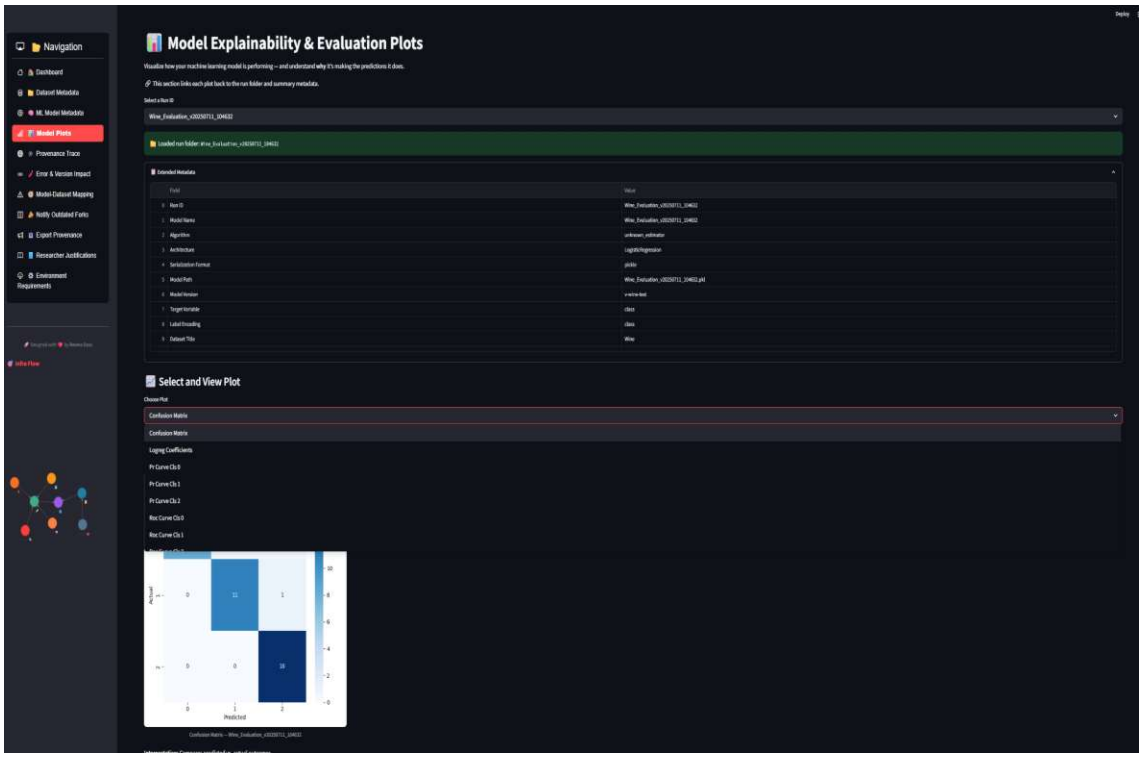
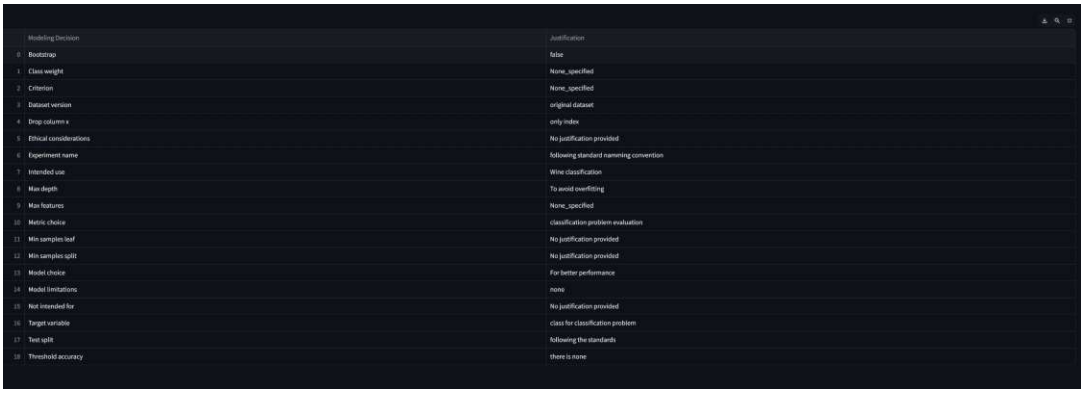
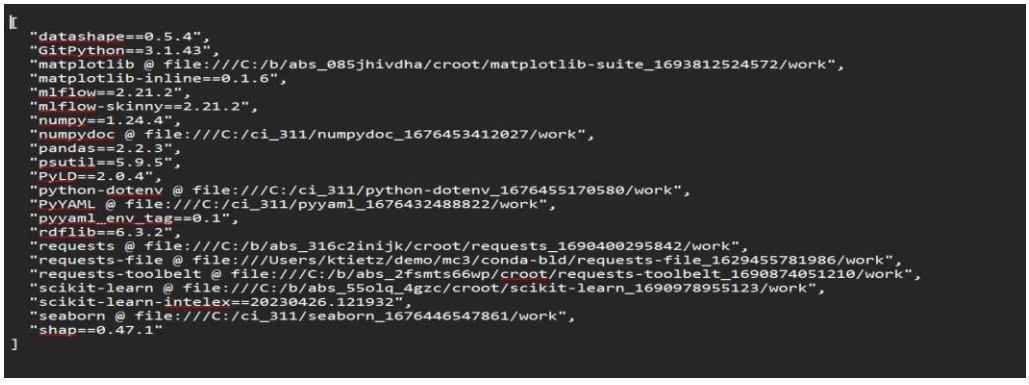


Figure 5.15: Model performance plot showing variation in accuracy and ROC AUC across datasets.



	Modeling Decision	Justification
0	Bootstrap	false
1	Class weight	None_specified
2	Criterion	None_specified
3	Dataset version	original dataset
4	Drop column x	only index
5	Ethical considerations	No justification provided
6	Experiment name	following standard naming convention
7	Intended use	Wine classification
8	Max depth	To avoid overfitting
9	Max features	None_specified
10	Metric choice	classification problem evaluation
11	Min samples leaf	No justification provided
12	Min samples split	No justification provided
13	Model choice	For better performance
14	Model limitations	none
15	Not intended for	No justification provided
16	Target variable	class for classification problem
17	Test split	following the standards
18	Threshold accuracy	there is none

Figure 5.16: Justification logging and explanation view.



```
[
  "datashape==0.5.4",
  "GitPython==3.1.43",
  "matplotlib @ file:///C:/b/abs_085jhivdha/croot/matplotlib-suite_1693812524572/work",
  "matplotlib-inline==0.1.6",
  "mlflow==2.21.2",
  "mlflow-skinny==2.21.2",
  "numpy==1.24.4",
  "numpydoc @ file:///C:/ci_311/numpydoc_1676453412027/work",
  "pandas==2.2.3",
  "psutil==5.9.5",
  "PyLD==2.0.4",
  "python-dotenv @ file:///C:/ci_311/python-dotenv_1676455170580/work",
  "PyYAML @ file:///C:/ci_311/pyyaml_1676432488822/work",
  "pyyaml_env_tag==0.1",
  "rdflib==6.3.2",
  "requests @ file:///C:/b/abs_316c2inijk/croot/requests_1690400295842/work",
  "requests-file @ file:///Users/ktietz/demo/mc3/conda-bld/requests-file_1629455781986/work",
  "requests-toolbelt @ file:///C:/b/abs_2fsmts66wp/croot/requests-toolbelt_1690874051210/work",
  "scikit-learn @ file:///C:/b/abs_55olq_4gzc/croot/scikit-learn_1690978955123/work",
  "scikit-learn-intelex==20230426.121932",
  "seaborn @ file:///C:/ci_311/seaborn_1676446547861/work",
  "shap==0.47.1"
]
```

Figure 5.17: Requirements tab outlining experiment setup expectations.

5.4.2 Usability Ratings and User Feedback

Participants rated several aspects of dashboard usability and visual clarity on a Likert scale (1–5):

- **Ease of preprocessing and metadata linking:** 4.0 average *"Embedding preprocessing and metadata capture was mostly intuitive"*
- **Ease of connecting MLflow and notebook logs:** 4.0 average *"Linking to Git and session metadata was straightforward"*
- **Clarity of configuration and experiment inspection:** 4.5 average *"Configuration tabs were helpful for interpreting runs"*
- **Ease of locating affected runs or errors:** 5.0 average *"Error tracing via logs worked smoothly"*

Particularly when it comes to examining setup details, provenance paths, and reasoning material, these grades show good perceived usefulness. The user study's full questionnaire may be found in Appendix D. A detailed mapping of user study findings to each research question is provided in Section 5.6.

Conclusion

Effective tools for browsing provenance data, seeing performance patterns, and comprehending experimental explanations were included in the dashboard design, which was judged as being both visually clear and straightforward. Clearer UX signals would help some parts but overall, the interface satisfied user demands and facilitated interpretation activities as required by RQ3.

5.5 RQ4: Export Interoperability and Standards Mapping

Objective

This assessment evaluates the framework's semantic export module's ability to convert structured ML metadata into ontology-aligned and syntactically correct forms (such as RDF/XML and JSON-LD), as well as how compatible these exports are with external tools and SPARQL-based querying systems.

Methodology

As outlined in Section 1, the export module serializes metadata aligned to five ontologies: **FAIR**, **FAIR4ML**, **PROV-O**, **MLSEA**, and **Croissant**. The evaluation focused on three aspects:

- **Syntactic validation:** Files were tested using the JSON-LD Playground and RDF Visualizer for testing interoperability.
- **Queryability:** The dashboard's SPARQL panel supported both predefined and custom queries over the exported triples i.e. for JSON-LD and RDF/XML.

5.5.1 Results

Export Validity and Ontology Alignment.

- All exported files passed JSON-LD (Figure 5.18) and RDF/XML schema (Figure 5.19) validation with no syntax errors or serialization failures.
- A sample of key predicates (e.g., `prov:used`, `prov:wasGeneratedBy`, `mls:modelName`, `mls:accuracy`) was checked and found to be correctly mapped to their intended namespaces.
- The graphs for both formats were rendered in the dashboard (Figure 5.20) to visually verify namespace binding, triple structure, and completeness.

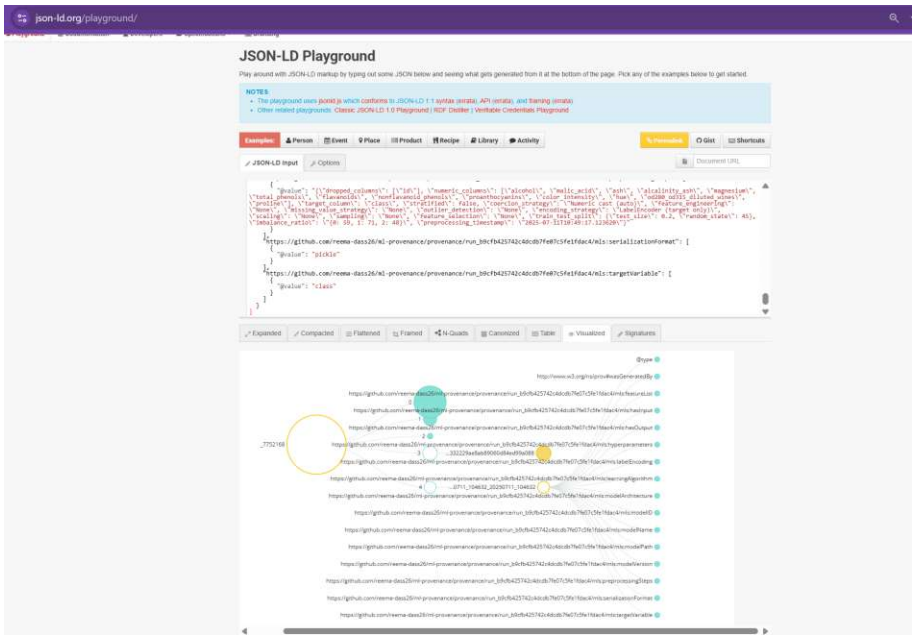


Figure 5.18: Full JSON-LD visualization in the open-source JSON-LD Playground, confirming that the generated file is syntactically valid and can be successfully rendered by external tools.

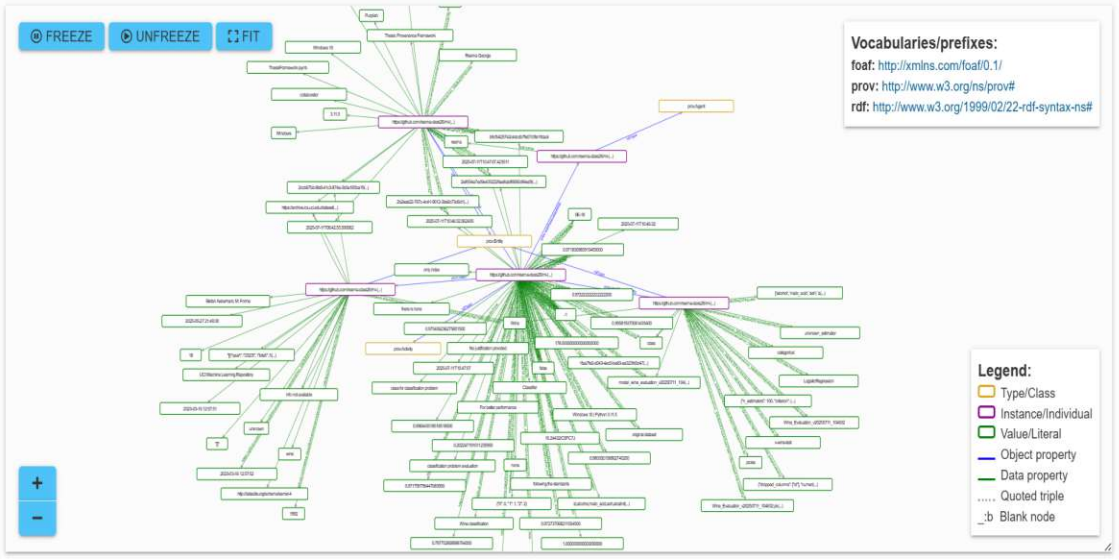


Figure 5.19: Full RDF/XML rendering in the open-source RDF Visualizer, verifying that the file passes syntax validation, preserves correct ontology mappings, and can be visualized without errors in third-party tools.

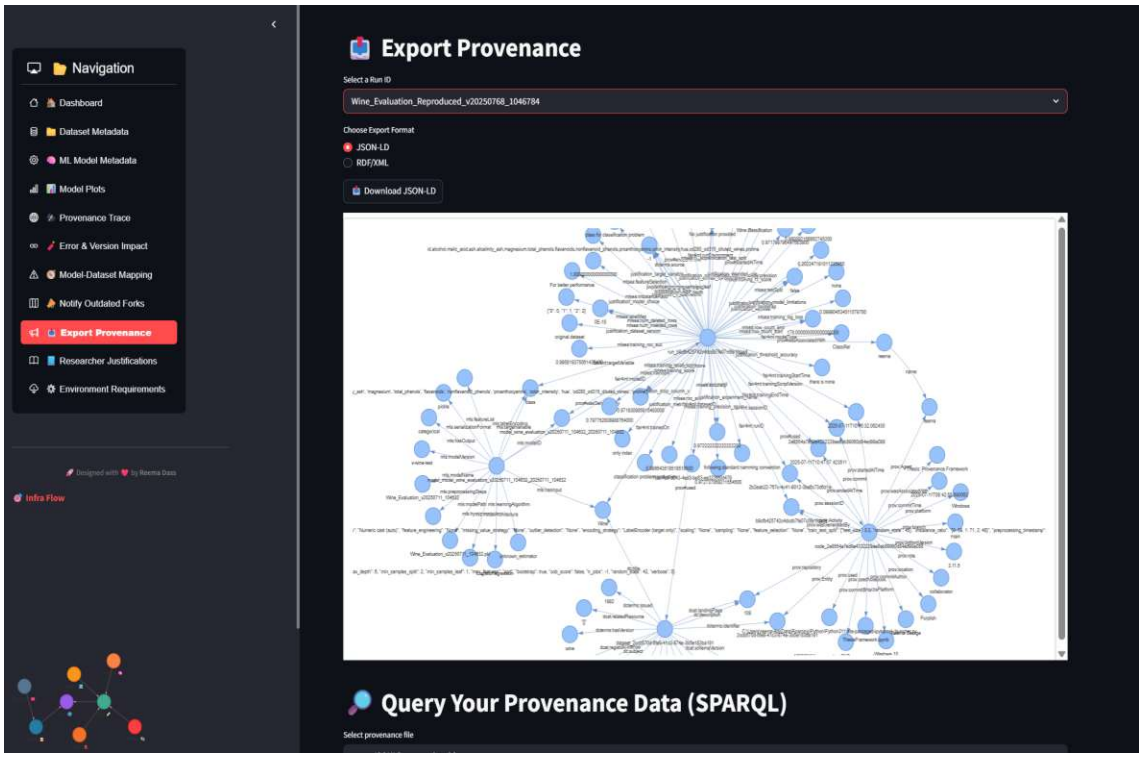


Figure 5.20: Semantic provenance visualization rendered within the framework dashboard.

SPARQL Query Execution. The exported RDF/XML and JSON-LD files were verified to be SPARQL-compatible, as evidenced by the execution of both predefined and custom queries in the dashboard interface. All eight predefined queries (e.g., "What dataset was used in this run?", "Who ran this experiment?", "List all training and test metrics", "Get model hyperparameters") returned valid results without parsing errors, demonstrating syntactic correctness of the semantic export. In addition, users can input arbitrary SPARQL queries through the dashboard, enabling flexible retrieval of any mapped provenance field (refer to yellow Box # in Figure 5.21.

– **Training Start-End Times (Figure 5.21):** The JSON-LD query retrieves the start and end timestamps for each training activity:

```
SELECT ?start ?end WHERE {
  ?run a <http://www.w3.org/ns/prov#Activity> ;
       <http://www.w3.org/ns/prov#startedAtTime> ?start ;
       <http://www.w3.org/ns/prov#endedAtTime> ?end .
}
```

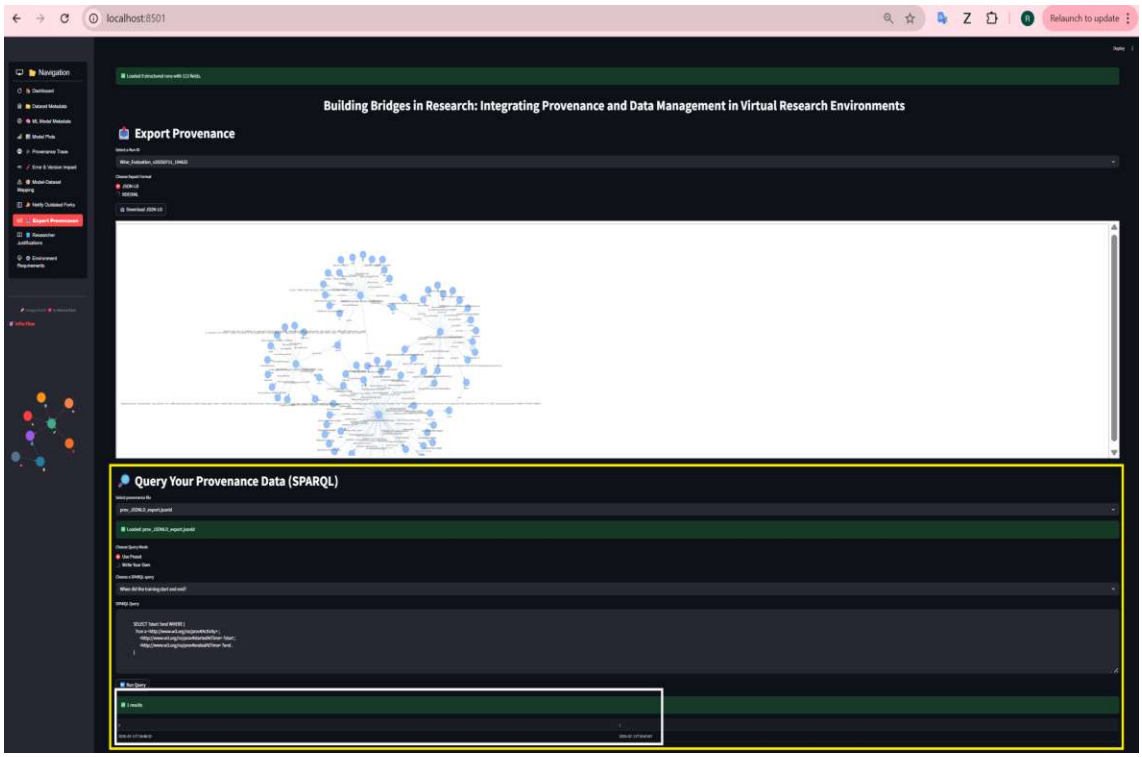


Figure 5.21: SPARQL query on JSON-LD retrieving `prov:startedAtTime` and `prov:endedAtTime` for each `prov:Activity`. Returned timestamps match MLflow logs, confirming preservation of training start–end times.

The result lists the timestamps for training start and completion, which match the times recorded in MLflow logs (refer to white Box # in Figure 5.21.

– **Agent Identification (Figure 5.22):** The RDF/XML query lists all agents who conducted experiments:

```
SELECT ?agentName WHERE {
  ?agent a <http://www.w3.org/ns/prov#Agent> ;
         <http://xmlns.com/foaf/0.1/name> ?agentName .
}
```

The result in the screenshot correctly returns the experimenter name (e.g., the MLflow user recorded during the run). This ensures that the provenance graph preserves the `prov:Agent-foaf:name` mapping, enabling attribution of each run to a specific user (refer to white Box # in Figure 5.22.

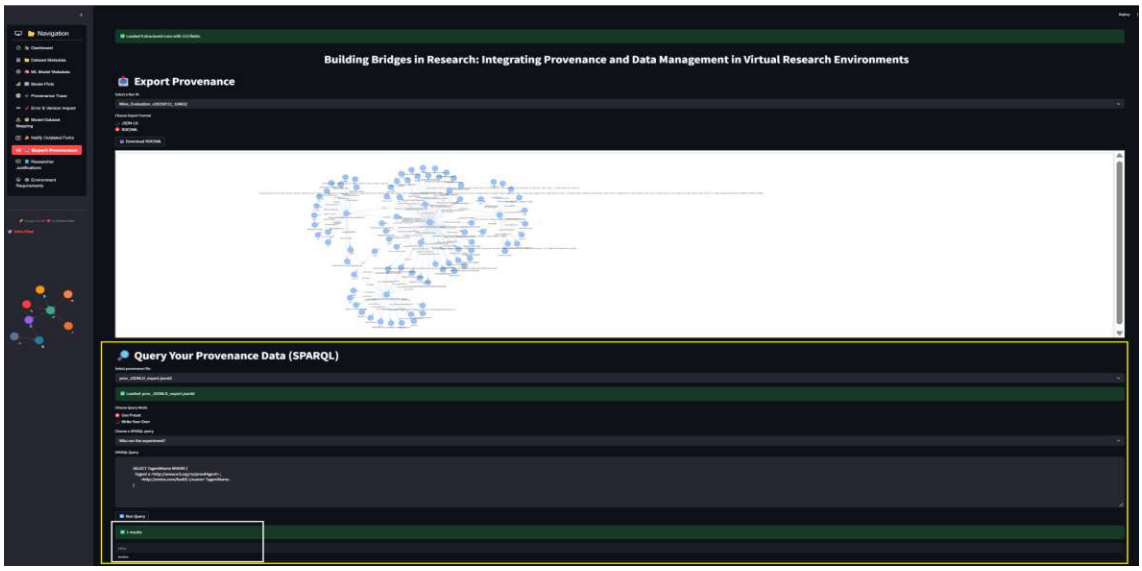


Figure 5.22: SPARQL query on RDF/XML retrieving `foaf:name` for each `prov:Agent`. Output matches the MLflow experimenter name, confirming correct agent attribution.

Verification was performed by cross-checking the query results with the corresponding MLflow run metadata. Exact matches for both agent names and activity timestamps ensure that the exported RDF/XML and JSON-LD representations have valid data.

Conclusion

The framework fulfills its export objectives:

- **Syntactic Validity:** JSON-LD and RDF/XML files are standards-compliant and SPARQL-compatible.
- **Query and Reuse:** Data is queryable via SPARQL and visualized through both internal and external tools.

These results show that linked-data interoperability is supported by the export method while permitting domain-specific enhancements. Future revisions may extend serialization to incorporate post-archival metadata.

5.6 User Study

To complement the framework's evaluation, a small-scale exploratory user study was conducted. Its purpose was to gather indicative qualitative and quantitative feedback on usability, reproducibility support, and semantic export features. The limited sample size means the study is not intended to provide statistically significant or technically quantifiable results; rather, it offers illustrative evidence of the framework's practical value.

Participants: Two machine learning practitioners participated:

- **User 1:** Data Science student, intermediate ML experience
- **User 2:** IT student, beginner ML experience

Neither had prior exposure to the framework or to provenance tools.

Study Design

Participants were instructed to:

- Run a complete ML experiment using a pre-configured notebook with the Wine dataset and a Logistic Regression model.
- Inspect generated metadata files, JSON-LD/RDF exports, and dashboard visualizations.
- Attempt to reproduce their prior run and debug issues based solely on captured metadata.
- Complete a structured questionnaire assessing coverage, usability, and interoperability.

Findings Mapped to Research Questions

RQ1 - Metadata Coverage and Standards

- Both users confirmed successful capture of dataset and model metadata.

- MLflow integration and ontology-based export (e.g., FAIR, PROV-O, MLSEA) were functional and interpretable.
- **Ratings:** Dataset-metadata linkage: 4/5 (User 1), 3/5 (User 2); MLflow linkage clarity: 5/5 (User 1), 3/5 (User 2).
- **Feedback:**
  - "All metadata was captured. MLflow made tracking easy." (User 1)
  - "Information was dense-some DOI metadata missing." (User 2)

RQ2 - Debugging and Reproducibility

- Both users successfully reproduced a prior run using only the captured metadata and dashboard information.
- Dataset name mismatches were detected due to incorrect mapping and resolved.
- **Ratings:** Debugging provenance: 5/5 (User 1), 4/5 (User 2); Sync/mismatch detection: 3/5 (User 1), 4/5 (User 2); Traceability (model → dataset): 5/5 (both).
- **Notable feedback:**
  - **User 1 (3/5 on sync/mismatch detection):** "Detected mismatches in some field key values, which were corrected, but the interface required scrolling between multiple tables to see all details."
  - **User 2:** "I could follow the entire execution path but with some help."

RQ3 - Visualization and Usability

- The dashboard was rated intuitive, especially the summary and model panels.
- Users found justifications useful but requested collapsibility and tooltips.
- **Ratings:** Dashboard usability: 5/5 (User 1), 4/5 (User 2); Visual clarity (e.g., ROC–AUC plots): 5/5 (both).

– **Suggestions:** “Great views but too much information on screen sometimes.” (User 1); “Add tooltips and make justification panels collapsible.” (User 2)

RQ4 - Export and Interoperability

– JSON-LD and RDF/XML files were corectly imported in external tools.

– Dashboard-based SPARQL queries returned correct results.

– **Ratings:** Schema comprehension: 4/5 (both); Export success: Yes (both); Interoperability via queries: Yes (both).

– **Feedback:** “Covers everything but needs simpler views.” (User 1); “It works well but is overwhelming for new users.” (User 2)

	C	D	E	F
	Question / Metric	Response Type(5 being best)	Response: USER1	Response: USER2
	User ID	Text		
	Experience Level (ML)	Categorical (None, Beginner, Intermediate, Expert)	Intermediate	Beginner
	Experience Level (Provenance tools)	Categorical (None, Beginner, Intermediate, Expert)	None	None
	Role	Text	Data science student	IT Student
	Prior Familiarity with Framework	Boolean (Yes/No)	None	None
	Dataset used	Text	Wine	Iris
	Model used	Text	Logistic Regression	Random forest
	Start Date/Time	DateTime	45 mins(exp + exploration)	1 hour 15 mins
	Total Adaptation	Computed (Duration)	30 mins	45 mins
	Were all relevant dataset metadata captured automatically?	Boolean (Yes/No)	Yes	yes
	Rate ease of embedding preprocessing & linking external datsets	Likert (1-5)	4	3
	Were you able to export data provenance metadata in standardized formats?	Boolean (Yes/No)	yes	yes
	Comments on metadata completeness	Open text	everrthing needed	Missing metadata from the DOI werent availaboe
	Was model training info (config, metrics, weights) captured fully?	Boolean (Yes/No)	yes	Yes
	Ease of connecting MLflow or notebook metadata	Likert (1-5)	5	3
	Did the metadata export support multiple formats (e.g., PROV-O, MLSEA)?	Boolean (Yes/No)	yes	yes
	Comments on export and metadata formats	Open text	over all functional but with few minor discrepencies	overwhelming info
	Could you trace exactly how a specific result was produced (e.g., which data, code version, params)?	Boolean (Yes/No)	yes	yes
	How confident are you in the completeness of provenance for debugging?	Likert (1-5)	5	4
	Were you able to identify experiments with outdated or faulty data/code versions?	Boolean (Yes/No)	yes	yes
	How easy was it to find affected researchers/users for notification?	Likert (1-5)	5	5
	Were you able to review training configs (e.g., data splits, hyperparams) across experiments?	Boolean (Yes/No)	yes with minor mapping issues	yes
	Rate clarity of experiment configuration inspection	Likert (1-5)	5	4
	Could you link which models were trained on which datasets (and versions)?	Boolean (Yes/No)	yes	yes
	Were performance metrics sufficiently detailed and accessible?	Likert (1-5)	5	5
	Could you detect outdated forks or divergent versions in data/code repos?	Boolean (Yes/No)	yes	yes
	Rate ease of identifying and prompting synchronization	Likert (1-5)	3	4
	How did provenance metadata assist in overall auditing, debugging, and reproducibility?	Open text	clear and helpul to do the above things esp with req and dataset info and hyper params	It clearly documented each step of the workflow, making it easy to trace errors, verify results, and repeat experiments.
	Rate intuitiveness of provenance dashboards and widgets	Likert (1-5)	5	4
	Did visualizations help you identify inefficiencies or errors?	Boolean (Yes/No)	yes	yes
	Rate clarity of traceability flows and preprocessing traces	Likert (1-5)	5	5
	Describe any workflow improvements or corrections made due to provenance insights	Open text	unsure,	More narrowed to the specific scope,
	Any improvements to visualization or interaction?	Open text	no, everything seems easily understandable, easy to navigate	The visualizations were informative but could be more interactive and customizable for better usability.
	Rate how easy it was to understand the metadata schema	Likert (1-5)	4	
	Were you able to export metadata to standard/open formats easily?	Boolean (Yes/No)	yes	yes
	Is the metadata usable for integration with other platforms/tools?	Boolean (Yes/No)	yes	yes
	Thoughts on schema design and export capabilities	Open text	could be simplified. but provides necessary info	The schema was comprehensive and the export worked well, though simplifying the structure could help new users.
	Learning Curve (time/effort to onboard)	Numeric (hours/minutes)	2-3 hours	
	ease of use	Likert (1-5)	5	4
	visual clarity	Likert (1-5)	4.5	4.5
	Overall satisfaction	Likert (1-5)	4	4
	Documentation quality	Likert (1-5)	5	4
	Recommendations	Open text	few mapping issues, complexity of dbrepo linking, visualization of export really help but it can be simplified, Queries helped to navigate	Improve interactivity in the UI, streamline export options, and offer quick-start guides or tooltips.
	feedback	Open text	makes more sense for a bigger project with more collaborator as the functionality are detailed, not relevent for small group , for small group is time consuming	

Figure 5.23: User study feedback summary across RQ dimensions.

Summary

This short user study provided early validation of the framework’s utility:

- Automated metadata capture, reproducibility features, and semantic export were easily used by both participants with minimal guidance.
- Average usability across key aspects was **4.2/5**.
- Key improvement areas: simplifying export structures, collapsible justification panels, more compact and precise views, and adding tool-tips.
- Although **not statistically representative**, the study illustrates practical feasibility in real-world usage and identifies areas for improvement prior to broader deployment.

5.7 Summary

This chapter offered a systematic evaluation of the provenance-aware machine learning architecture in four main research domains. Using executed runs, dashboard walkthroughs, metadata audits, semantic export validations, and a user survey with two participants, the system was assessed for metadata capture quality, repeatability support, usability, and semantic interoperability.

- **RQ1 - Metadata Capture and Standards Alignment:** The evaluation demonstrates, with quantitative evidence, that the framework achieves comprehensive provenance capture aligned with established semantic vocabularies while remaining scoped to notebook-native, tabular classification workflows. By constructing a reproducible ontology union from official releases and refining it to a task-scoped subset U_{cls}, the analysis isolates fields relevant to this experimental context. The resulting **70.09%** completeness on U_{cls}, compared to **55.76%** on the full union U_{all}, indicates that remaining gaps are largely due to out-of-scope lifecycle terms, rather than deficiencies in capture. Although higjer coverage can be achieved if need be by including more fields. Over **80%** of U_{cls} fields are captured automatically from trusted sources (*git*, *mlflow*, *dbrepo*, *doi* registries, and *jupyter* notebooks), with the remainder covered via structured prompts or targeted manual input. This balance ensures both

standards compliance and inclusion of operational metadata necessary for reproducibility, interpretability, and responsible model governance.

- **RQ2 - Reproducibility, Version Tracking, and Debugging:** Using just the recorded metadata, the dashboard enabled full replication of earlier runs with similar metrics between the original and replicated executions. The *Version Impact* and *Provenance Trace* tabs provided GitHub-based notifications to inform contributors of deprecated inputs, compared field-level changes, and identified out-of-date dataset versions (e.g., `iris_v4`). Additionally highlighted were discrepancies between fork divergence and Git commits.
- **RQ3 - Usability and Visualization Effectiveness:** Participants gave the dashboard high marks (**4.2/5** average), especially for model traceability, run inspection, and provenance-based debugging. ROC charts, configuration panels, justification logs, and summary views all helped to clarify the behavior of the ML pipeline. The input included suggestions for reducing screen clutter, foldable panels, and tooltips.
- **RQ4 – Semantic Export and Interoperability:** The dashboard successfully produced structured metadata, verified it with external tools, and exported it to **RDF/XML** and **JSON-LD**. Each file passed syntactic testing and had fields that were aligned with the ontology. Both preset and custom SPARQL searches produced valid results, demonstrating interoperability with linked-data ecosystems.

Overall, the approach shown significant promise in each evaluation domain. With the use of visual inspection tools, it facilitates repeatability and error monitoring, automates metadata extraction while preserving flexibility for human input, and produces exportable, ontology-compliant representations that can be connected with other repositories and semantic web platforms. The user study supports the system's utility, usability, and adaptability for group machine learning research. This evaluation lays the foundation for future enhancements and verifies that the framework aligns with the thesis's goals.

6 Discussion

This chapter, which is organized around the framework's primary architectural elements, offers reflections on the results of the system's installation and assessment. It draws attention to what succeeded, what failed, and what these findings mean for practical application.

6.1 Session and Environment Metadata

The system's full automation and accurate ontology mapping allowed it to dependably gather session-level metadata (user, timestamp, and platform). Without human input, this layer operated consistently during all runs.

Implication: In controlled notebooks, environment capture may be completely automated. However, extra logic would be required to monitor roles or containers in cloud settings or multi-user systems.

6.2 Dataset Metadata and DOI Enrichment

Most anticipated fields were filled in by dataset information from DBRepo and optional DOI enrichment. However, because to gaps in the source information, several fields (such `dc:license` and `dc:language`) were often absent.

6.3 Model and Justification Capture

Performance indicators, training information, and model settings were automatically/semi-automatically recorded. Users filled in the justification

blocks manually, although not always.

Implication: Automating the provenance of core models is possible. However, human logic is still difficult to capture; templates or suggestions may be useful in this regard.

6.4 Git Lineage and Version Tracking

Run-time recording of Git information allowed for input commit tracing and the identification of out-of-date forks. GitHub was used to show version-aware inspection and collaborator notification.

Implication: For simple lineage, git integration works well. Deeper integration could be required for more complex workflows (such CI/CD or multi-branch histories).

6.5 Dashboard and SPARQL Features

Users may visually investigate provenance and filter by models, datasets, and commits using the dashboard. Valid responses from SPARQL searches validated the semantic export structure. According to feedback, usability was OK, however UI enhancements (such foldable panels) would be beneficial.

Implication: Visual interfaces are essential for understanding provenance. Although SPARQL performs well internally, it is still too complex for non-experts.

6.6 Ontology Mapping and Field Alignment Impact

This work aligned the framework's internal metadata schema with five key vocabularies: `PROV-O`, `FAIR`, `FAIR4ML`, `MLSEA`, and `Croissant`. A unified ontology was built from official releases and refined to the task-specific subset U_{cls} for notebook-based, tabular classification.

Key steps:

- Mapped 100+ fields across datasets, models, sessions, Git history, and experiment logs.
- Categorised capture as automatic, semi-automatic, or manual from sources such as *git*, *mlflow*, *dbrepo*, *doi*, and *jupyter*.
- Verified JSON-LD and RDF/XML exports via SPARQL queries and external validators.

Impact: Achieved **70.09%** coverage on U_{cls} vs. **55.76%** on U_{cls}, with over **80%** of U_{cls} fields captured automatically. The remaining fields required targeted manual input. The mapping enables linked-data interoperability, machine-actionable discovery, and reusable alignment templates for other ML workflows.

6.7 Limitations

Notwithstanding the system's advantages, a number of drawbacks surfaced during testing and deployment, mainly in relation to initial setup, user input, metadata quality, and semantic abstraction:

- **Initial setup and generalization:** Even though the system is modular, it takes technical expertise and careful coordination to configure every component (such as MLflow, DBRepo, Git integration, and export logic). It could be difficult to modify the configuration for various organizational or infrastructure environments. Additionally, the initial setup takes time, and clarity about the project scope is required to design a personalized framework.
- **Inconsistent justification capture:** User input, which varied in depth and detail between tests, is what the justification block relies on. The system's capacity to completely explain parameter selections or preprocessing justifications is weakened as a result.

- **External metadata dependency:** The completeness of dataset metadata is mostly dependent on other sources, such as DOI services or DBRepo. Without human interaction, the framework's capacity to meet FAIR principles is limited by missing areas like license, language, or subject.
- **Ontology and SPARQL complexity:** Semantic exports were legitimate, but understanding RDF syntax and ontology structure is necessary to write SPARQL queries. This may discourage wider adoption and restrict usage to people with more technical expertise.
- **Ontology alignment trade-offs:** Certain useful fields, such as user comments, Git-specific logic, and custom annotations, lacked direct matches in the chosen ontologies. To maintain standards compliance, these were simplified or omitted during ontology mapping. While this preserved formal conformance, it reduced the granularity and context available in the semantic representation.
- **Task scope bias:** As this framework focuses only on simple classification tasks and workflow, additional configurations will be required to adapt to different ML tasks.

These restrictions are a reflection of the inherent trade-offs between interoperability, completeness, and usefulness. More abstraction layers, improved user advice, and wider metadata enrichment support would be needed to address these.

7 Conclusion

This thesis presented and assessed a modular, provenance-aware machine learning framework meant to promote transparency, reproducibility, and semantic interoperability in experimental ML processes. By incorporating automatic metadata collection, structured export, and visual inspection capabilities into a notebook-native Virtual Research Environment (VRE), the solution offers a practical and adaptable way to align ML research with FAIR principles and widely accepted ontologies. The complete code implementation of the framework can be found in Github. The artifacts mentioned in this report can be found in Zenodo.

7.1 Key Contributions

This work made the following key contributions:

- A layered metadata architecture that maps to many semantic ontologies (PROV-O, FAIR4ML, MLSEA, Croissant) that records session, dataset, model, experiment, and versioning details.
- MLflow is used to build automatic and semi-automated metadata logging, including output to RDF/XML, JSON, and JSON-LD formats that are compatible with semantic web tools.
- A dashboard that improves the interpretability and auditability of machine learning processes by supporting SPARQL-based querying, error tracing, version impact analysis, and run-level examination.

- A reusable connection between linked-data representations and raw machine learning objects by methodically mapping more than 100 metadata fields to semantic standards.
- Metadata coverage (70%), degree of automation (80%), repeatability, usability, and standards alignment were demonstrated in this thorough examination, which was both simulated and user-driven.

7.2 Research Questions Revisited

- **RQ1:** Through high automation levels, the system achieved coverage of task-relevant metadata fields. Alignment gaps were primarily due to missing details, irrelevant fields to the scope of the thesis, and selectively provided justification inputs, rather than deficiencies in capture.
- **RQ2:** Version impact tracking, hash-based comparison, and run re-execution were effective methods for proving reproducibility. Proactive collaborator notifications for out-of-date versions were made possible via Git integration.
- **RQ3:** Although UI improvements were suggested, users gave the dashboard good marks for clarity and usefulness in debugging, exploration, and provenance tracking.
- **RQ4:** Exporting to semantic formats allowed for successful SPARQL querying and interoperability of exported files.

7.3 Looking Forward

Although the framework performs well, it serves primarily as a foundation for building provenance-aware systems. Real-world scenarios can be far more complex, and ontologies may evolve over time. Future work may focus on:

- Automated completion of missing metadata, particularly for dataset provenance and licensing.

- Traceability and reproducibility are maintained when integrating with distributed execution engines.
- Use ontology-agnostic templates or visual SPARQL builders to speed semantic searching.
- Using the metadata mapping approach to create reusable tools or libraries that may be included into platforms and pipelines for machine learning.

7.4 Final Remarks

This research shows that provenance in machine learning doesn't have to be an academic ideal or an afterthought; it is possible to incorporated into everyday procedures. By integrating visual tools, semantic export, and useful information collection, the solution creates the foundation for more visible, repeatable, and explicable machine learning experiments. By doing this, it contributes to closing the gap between machine-actionable research objects and experimental repeatability, which is a need that is becoming more and more important in data-driven science.

Bibliography

[1] Enis Afgan et al. "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update". In: *Nucleic Acids Research* 46.W1 (2018), W537–W544. DOI: 10.1093/nar/gky379. URL: https://doi.org/10.1093/nar/gky379.

[2] Tibor Šimko et al. "REANA: A platform for reproducible research data analyses". In: *EPJ Web of Conferences* 214 (2019), p. 06034. DOI: 10.1051/epjconf/201921406034. URL: https://doi.org/10.1051/epjconf/201921406034.

[3] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas. Proceedings of the 20th International Conference on Electronic Publishing (ELPUB 2016)*. IOS Press, 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87. URL: https://doi.org/10.3233/978-1-61499-649-1-87.

[4] Arcot Rajasekar et al. *iRODS Primer: Integrated Rule-Oriented Data System*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Springer Cham, 2022. ISBN: 9783031022708. DOI: 10.1007/978-3-031-02271-5.

[5] Katy J. Wolstencroft et al. "The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud". In: *Nucleic Acids Research* 41.W1 (2013), W557–W561. DOI: 10.1093/nar/gkt328. URL: https://doi.org/10.1093/nar/gkt328.

[6] Carole A. Goble et al. “myExperiment: a repository and social network for the sharing of bioinformatics workflows”. In: *Nucleic Acids Research* 38.suppl_2 (2010), W677–W682. DOI: 10.1093/nar/gkq429. URL: https://doi.org/10.1093/nar/gkq429.

[7] Comet, Inc. *Comet: Track and Visualize Machine Learning Experiments*. Accessed: 2025-07-08. 2023. URL: https://www.comet.com/.

[8] Timothy Lebo et al. *PROV-O: The PROV Ontology*. W3C Recommendation. W3C Recommendation 30 April 2013. Apr. 2013. URL: https://www.w3.org/TR/prov-o/.

[9] Runzhou Han et al. “PROV-IO+: A Cross-Platform Provenance Framework for Scientific Data on HPC Systems”. In: *arXiv preprint arXiv:2308.00891* (Aug. 2023). DOI: 10.48550/arXiv.2308.00891. URL: https://doi.org/10.48550/arXiv.2308.00891.

[10] Mubashara Akhtar et al. “Croissant: A Metadata Format for ML-Ready Datasets”. In: *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS 2024), Datasets and Benchmarks Track*. Curran Associates, Inc., 2024. DOI: 10.1145/3650203.3663326. URL: https://doi.org/10.1145/3650203.3663326.

[11] Mark D. Wilkinson et al. “The FAIR Guiding Principles for scientific data management and stewardship”. In: *Scientific Data* 3 (2016), p. 160018. DOI: 10.1038/sdata.2016.18. URL: https://doi.org/10.1038/sdata.2016.18.

[12] Dhwani Solanki et al. *FAIR4ML: A Vocabulary to Describe Machine/Deep Learning Models*. Conference paper (Version v1, published August 4, 2025). Editors: York Sure-Vetter, Paul Groth. 2025. URL: https://rda-fair4ml.github.io/FAIR4ML-schema/.

[13] MLflow Project, a Series of LF Projects, LLC. *MLflow: Open-source Platform for the Machine Learning Lifecycle*. Accessed: 2025-08-17. 2025. URL: https://mlflow.org/.

[14] Iterative, Inc. *Data Version Control (DVC)*. Accessed: 2025-08-17. 2025. URL: https://dvc.org/.

[15] Ali Darejeh et al. “A Critical Analysis of Cognitive Load Measurement Methods for Evaluating the Usability of Different Types of Interfaces: Guidelines and Framework for Human–Computer Interaction”. In: *arXiv preprint arXiv:2402.11820* (2024). DOI: 10.48550/arXiv.2402.11820. URL: https://doi.org/10.48550/arXiv.2402.11820.

[16] Alan R. Hevner et al. “Design Science in Information Systems Research”. In: *MIS Quarterly* 28.1 (Mar. 2004), pp. 75–105. DOI: 10.2307/25148625. URL: https://doi.org/10.2307/25148625.

[17] Ioannis Dasoulas, Duo Yang, and Anastasia Dimou. *MLSea: A Semantic Layer for Discoverable Machine Learning*. Accessed: 2025-08-17. 2024. DOI: 10.5281/zenodo.10286868. URL: https://zenodo.org/records/10286868.

[18] Barbara Kitchenham and Stuart Charters. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Tech. rep. EBSE-2007-01. EBSE Technical Report, Keele University and University of Durham, Jan. 2007. URL: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

[19] Martin Weise and Andreas Rauber. “DBRepo: A Data Repository System for Research Data in Databases”. In: *Proceedings of the 2024 IEEE International Conference on Big Data (BigData)*. Washington, DC, USA: IEEE, 2024, p. 322. DOI: 10.1109/BigData62323.2024.10825401. URL: https://doi.org/10.1109/BigData62323.2024.10825401.

[20] Chunbo Liang, Gang Zhou, and Ning Li. “ProTracker: A Provenance Tracking System for Scientific Workflows”. In: *Proceedings of the 2010 International Conference on Educational and Information Technology (ICEIT)*. IEEE, 2010. DOI: 10.1109/ICEIT.2010.5607496. URL: https://doi.org/10.1109/ICEIT.2010.5607496.

[21] Zach Cutler, Kiran Gadhave, and Alexander Lex. “Trrack: A Library for Provenance-Tracking in Web-Based Visualizations”. In: *2020 IEEE Visualization Conference (VIS)*. IEEE, 2020. DOI: 10.1109/VIS47514.2020.00030. URL: https://doi.org/10.1109/VIS47514.2020.00030.

[22] Yong Liu and Matei Zaharia. *Practical Deep Learning at Scale with MLflow: Bridge the Gap Between Offline Experimentation and Online Production*. Packt Publishing, 2022. ISBN: 9781803241333. URL: https://ieeexplore.ieee.org/document/10163438.

[23] Natu Lauchande. *Machine Learning Engineering with MLflow: Manage the End-to-End Machine Learning Life Cycle with MLflow*. Packt Publishing, 2021. ISBN: 9781800560796. URL: https://ieeexplore.ieee.org/book/10162559.

[24] Renan Souza et al. "Provenance Data in the Machine Learning Lifecycle". In: *2019 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS)*. Denver, CO, USA: IEEE, 2019. DOI: 10.1109/WORKS49585.2019.00006. URL: https://doi.org/10.1109/WORKS49585.2019.00006.

[25] Bertram Ludäscher et al. "ProvONE+: A PROV Extension for Scientific Workflow Provenance". In: *Provenance and Annotation of Data and Processes (IPAW 2020)*. Ed. by Boris Glavic, Abdussalam Alawini, and Seda Ogrenci-Memik. Vol. 12064. Lecture Notes in Computer Science. Springer, 2020, pp. 249–263. DOI: 10.1007/978-3-030-62008-0_30. URL: https://link.springer.com/chapter/10.1007/978-3-030-62008-0_30.

[26] Paolo Ciccarese et al. "PAV ontology: Provenance, Authoring and Versioning". In: *Journal of Biomedical Semantics* 4.1 (2013), p. 37. DOI: 10.1186/2041-1480-4-37. URL: https://doi.org/10.1186/2041-1480-4-37.

[27] Justin Turnau et al. "Provenance-based Explanations for Machine Learning (ML) Models". In: *2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW)*. Anaheim, CA, USA: IEEE, 2023, pp. 25–32. DOI: 10.1109/ICDEW58674.2023.00011. URL: https://doi.org/10.1109/ICDEW58674.2023.00011.

[28] Jorge Piazentin Ono et al. *PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines*. Accessed: 2025-08-17. 2020.

DOI: 10.48550/arXiv.2005.00160. arXiv: 2005.00160 [cs.HC]. URL: https://arxiv.org/abs/2005.00160.

[29] Kai Petersen et al. "Systematic Mapping Studies in Software Engineering". In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. Swindon, UK: BCS, The Chartered Institute for IT, 2008, pp. 68–77. URL: https://dl.acm.org/doi/10.5555/2227115.2227123.

[30] Anthony Scerri et al. "Systematic Mapping of Metadata Standards for Scientific networks". In: *Future Generation Computer Systems* 107 (2020), pp. 503–518. DOI: 10.7717/peerj-cs.421. URL: https://www.researchgate.net/publication/349956890_A_systematic_metadata_harvesting_workflow_for_analysing_scientific_networks.

[31] Katharine Y. Chen, Maria Toro-Moreno, and Arvind Rasi Subramaniam. *GitHub is an effective platform for collaborative and reproducible laboratory research*. arXiv:2408.09344 [q-bio.OT]. 2024. DOI: 10.48550/arXiv.2408.09344. URL: https://doi.org/10.48550/arXiv.2408.09344.

[32] Margaret Mitchell et al. "Model Cards for Model Reporting". In: *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT*)*. Atlanta, GA, USA: ACM, 2019, pp. 220–229. DOI: 10.1145/3287560.3287596. URL: https://doi.org/10.1145/3287560.3287596.

[33] Timnit Gebru et al. "Datasheets for Datasets". In: *Communications of the ACM* 64.12 (2021), pp. 86–92. DOI: 10.1145/3458723. URL: https://doi.org/10.1145/3458723.

[34] Joelle Pineau et al. "Improving Reproducibility in Machine Learning Research: A Report from the NeurIPS 2019 Reproducibility Program". In: *Journal of Machine Learning Research* 22.164 (2021), pp. 1–20. URL: https://jmlr.org/papers/v22/20-303.html.

[35] Michiel van den Brand, Ronald Kemp, and Jan van der Veen. *Metadata for Machines (M4M): Lightweight Semantic Interoperability*. GO FAIR Foundation Workshop Report. Background document for M4M workshops on JSON-LD and semantic dashboarding. 2020. URL: https://www.go-fair.org/resources/go-fair-workshop-series/metadata-for-machines-workshops/.

[36] W3C. *RDF 1.1 XML Syntax*. W3C Recommendation REC-rdf-syntax-grammar-20140225. Editors: Fabien Gandon, Guus Schreiber. Defines RDF/XML format for RDF 1.1. World Wide Web Consortium (W3C), Feb. 2014. URL: https://www.w3.org/TR/rdf-syntax-grammar/.

[37] Emilio Rivera-Landos, Foutse Khomh, and Amin Nikanjam. "The Challenge of Reproducible ML: An Empirical Study on the Impact of Bugs". In: *arXiv preprint arXiv:2109.03991* (2021). SWAT Lab., Polytechnique Montreal. Version v1, 9 Sep 2021. URL: https://arxiv.org/abs/2109.03991.

[38] Jeffrey M. Perkel. "Why Jupyter is Data Scientists' Computational Notebook of Choice". In: *Nature* 563.7729 (2018), pp. 145–146. DOI: 10.1038/d41586-018-07196-1. URL: https://www.researchgate.net/publication/328613542_Why_Jupyter_is_data_scientists'_computational_notebook_of_choice.

[39] Victoria Stodden et al. "Enhancing Reproducibility for Computational Methods". In: *Science* 354.6317 (2016), pp. 1240–1241. DOI: 10.1126/science.aah6168. URL: https://doi.org/10.1126/science.aah6168.

[40] Rafael S. Gonçalves et al. "Aligning Biomedical Metadata with Ontologies Using Clustering and Embeddings". In: *Proceedings of the 10th International Conference on Biomedical Ontology (ICBO 2019)*. Ed. by Alexander D. Diehl, William D. Duncan, and Gloria Sanso. Vol. 2314. CEUR Workshop Proceedings. Buffalo, NY, USA: CEUR-WS, July 2019, pp. 1–12. URL: https://doi.org/10.48550/arXiv.1903.08206.

Appendices

A Evaluation

A.1 Ontology Snapshots and Retrieval Metadata

Table A.1: Snapshots used to construct the ontology union U_{all}

Ontology	**Source URL**	**Snapshot (ver/tag/commit)**	**Retrieved (UTC)**	**File / Format**
PROV-O	https://www.w3.org/ns/prov.ttl	W3C Recommendation 2013-04-30	2025-08-13	prov.ttl / Turtle
FAIR Vocabulary	https://w3id.org/fair/principles/terms/FAIR-Vocabulary	5.1.11	2025-08-13	FAIR-Vocabulary.ttl / Turtle
Croissant	https://raw.githubusercontent.com/mlcommons/croissant/v1.0.21/docs/croissant.ttl	v1.0.21	2025-08-13	croissant.ttl / Turtle
FAIR4ML	https://rda-fair4ml.github.io/FAIR4ML-schema/release/0.1.0/fair4ml.jsonld	0.1.0	2025-08-13	fair4ml.jsonld / JSON-LD
MLSEA	http://w3id.org/mlsea	1.0.0	2025-08-13	mlsea.ttl / Turtle

The copy of the above files can be found in Zenodo

Table A.2: Checksums for reproducibility

Ontology	File	SHA-256
PROV-O	*prov-o.ttl*	*7d203989f67b38bca572253942acc5a1bf24ce3ccfece16f072dcb4be2b79a96*
FAIR	*fair.ttl*	*1adff6bfa4764be997fb2d547fe118358ec184102b0c953a837444a091d3a1fd*
Croissant	*croissant.ttl*	*bd55bee774564247bb000f2078487687c25119b697437790ff17071f4d29bb59*
FAIR4ML	*fair4ml.jsonld*	*0bc6bcc960574b1d4990c7a6399a72f7cdc55e6ff68edd825b2599d821ca5bb7*
MLSEA	*mlsea.ttl*	*ae8079c75d4d7feea950039235d4548e5347e942840214f3334adc4a2dbdfa67*

A.2 Runs and their corresponding datasets

Table A.3: Executed runs: IDs, timing, and links to datasets/artifacts

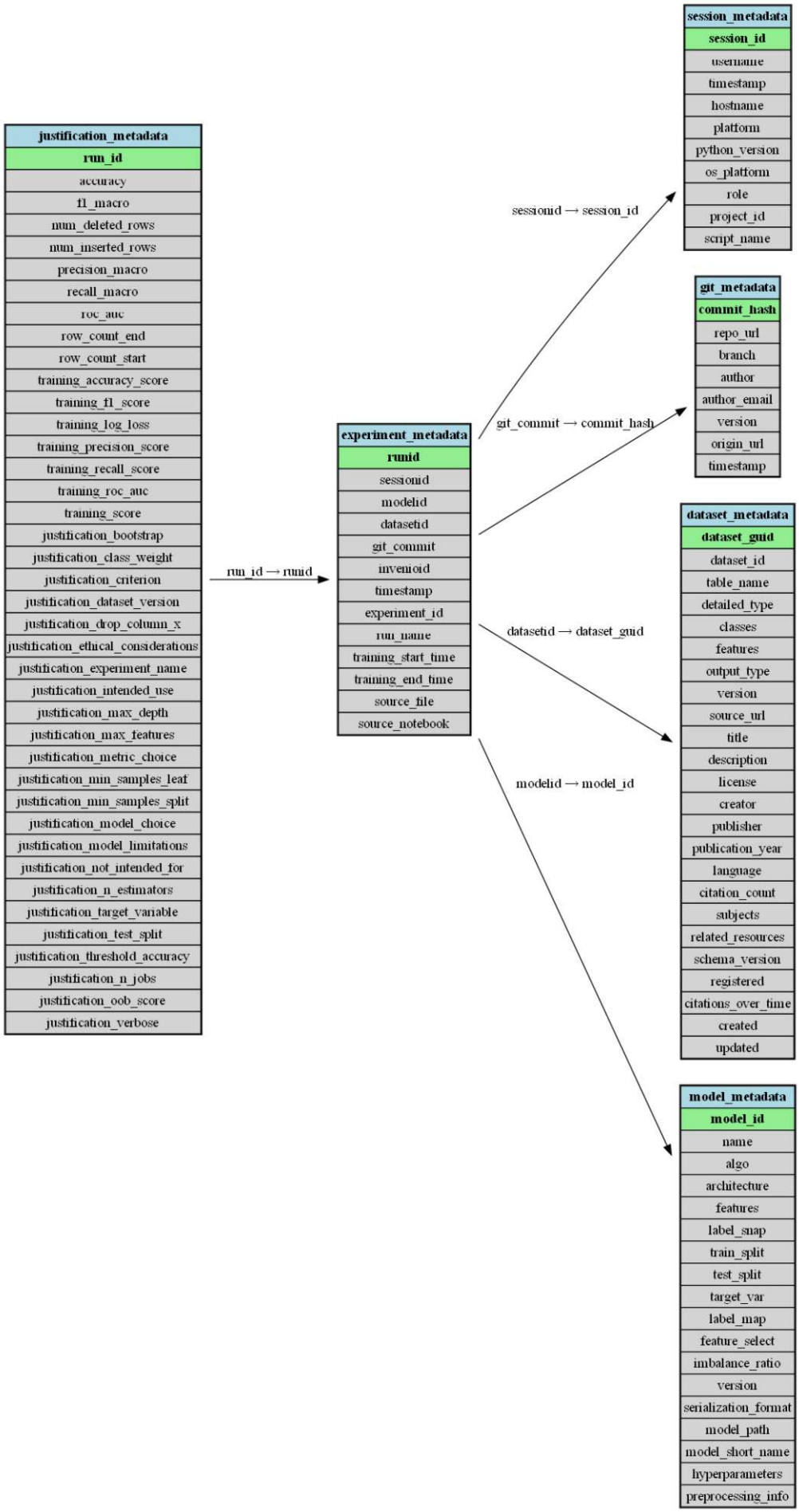
Run ID	Dataset / Variant	Artifacts
Iris_V0_v20250812_231232	Iris (Original)	https://zenodo.org/records/16874761/files/Iris_Version_A_Original.csv?download=1
Iris_v1_v20250812_231652	Iris (Duplicated)	https://zenodo.org/records/16874761/files/Iris_Version_1_Duplicated.csv?download=1
Iris_v2_v20250812_232012	Iris (First 100)	https://zenodo.org/records/16874761/files/Iris_Version_2_First100.csv?download=1
Iris_v3_v20250812_232355	Iris (Normalized)	https://zenodo.org/records/16874761/files/Iris_Version_4_Normalized.csv?download=1
Iris_v4_v20250812_232815	Iris (Distorted)	https://zenodo.org/records/16874761/files/distorted_iris_variant.csv?download=1
Wine_Evaluation_v20250812_233212	Wine (Evaluation)	https://zenodo.org/records/16874761/files/wine.csv?download=1
Wine_Evaluation_Repod_v20250812_233922	Wine (Reproduction)	https://zenodo.org/records/16874761/files/wine.csv?download=1
Wine_Evaluation_User2_v20250812_234338	Wine (User 2)	https://zenodo.org/records/16874761/files/wine.csv?download=1
Wine_Evaluation_User2_reprod_v20250813_163951	Wine (User 2, Reproduction)	https://zenodo.org/records/16874761/files/wine.csv?download=1

The actual run data can be found in the Github

B System architecture and Implementation

B.1 Database Schema Overview

The DBRepo database structure is intended to collect all metadata related to machine learning experiment tracking in several dimensions, including dataset attributes, model settings, execution environments, user sessions, Git provenance, and experiment justifications. The 'experiment_metadata' table is the core anchor, connecting all other metadata tables via foreign keys. Each table has standardized fields that comply with semantic metadata standards such as FAIR4ML, MLSEA, and PROV-O. This relational architecture supports organized storing, cross-referencing, and queryable access to important parts of experimental context.



C Framework Setup and Configuration

This chapter outlines how to set up and configure the ML provenance framework using a Python virtual environment and standard ‘pip‘ package management. The setup ensures reproducibility and consistency across deployments.

C.1 Quick-Start Installation Script

The following bash script creates a virtual environment, installs all dependencies from ‘requirements.txt‘, and launches the Streamlit dashboard:

```
# Clone the framework repository
git clone https://github.com/reema-dass26/Framework_Evaluation.git
cd notebook/RQ_notebook
cd ThesisFramework.ipynb


# Set up virtual environment
python3 -m venv .venv
source .venv/bin/activate   # On Windows use: .venv\Scripts\activate

# Install dependencies
pip install --upgrade pip
pip install -r requirements.txt

# Launch the Streamlit dashboard
streamlit run vizualization.py --server.headless true
```

Note: Python 3.10 or later is recommended. Make sure you have ‘pip‘, ‘venv‘ and ‘streamlit‘ installed system-wide before running the script.

C.1.1 MLflow Configuration

The MLflow configuration file defines the tracking URI, experiment names, and artifact locations for reproducible experiment logging.

```
artifact_location: file:///C:/Users/reema/REPO/notebooks/RQ_notebooks/mlrunlogs/mlflow.db/0
creation_time: 1745329164510
experiment_id: 'VRE_FRAMEWORK'
last_update_time: 1745329164510
lifecycle_stage: active
name: Default
```

C.1.2 Streamlit UI Configuration

This TOML file allows customization of dashboard elements like title, theme, and active modules.

```
# [theme]
# base = "dark"
# primaryColor = "#00d4ff"
# backgroundColor = "#0e1117"
# secondaryBackgroundColor = "#262730"
# textColor = "#fafafa"
# font = "monospace"
```

Note: These configuration files are expected under the ‘config/‘ directory. You may also override parameters via environment variables or CLI flags when needed.

D User Study Instruments

D.1 Blank Questionnaire

Participants were asked to evaluate the framework based on their hands-on experience. The questionnaire focused on usability, metadata completeness, and reproducibility support.

1. **Your Machine Learning Experience Level (e.g., Beginner / Intermediate / Expert):** _____
2. **Was the metadata automatically captured correctly? (1 = No, 5 = Perfect):** _____
3. **Was the MLflow UI easy to understand and navigate? (1 = Confusing, 5 = Very intuitive):** _____
4. **Were the semantic exports (JSON-LD / RDF) understandable? (1 = Not at all, 5 = Very clear):** _____
5. **What did you like most about the system?** _____
6. **What improvements would you suggest?** _____

D.2 Anonymized Responses

C	D	E	F
Question / Metric	**Response Type(5 being best)**	**Response: USER1**	**Response: USER2**
User ID	Text		
Experience Level (ML)	Categorical (None, Beginner, Intermediate, Expert)	Intermediate	Beginner
Experience Level (Provenance tools)	Categorical (None, Beginner, Intermediate, Expert)	None	None
Role	Text	Data science student	IT Student
Prior Familiarity with Framework	Boolean (Yes/No)	None	None
Dataset used	Text	Wine	Iris
Model used	Text	Logistic Regression	Random forest
Start Date/Time	DateTime	45 mins(exp + exploration)	1 hour 15 mins
Total Adaptation	Computed (Duration)	30 mins	45 mins
Were all relevant dataset metadata captured automatically?	Boolean (Yes/No)	Yes	yes
Rate ease of embedding preprocessing & linking external datsets	Likert (1-5)	4	3
Were you able to export data provenance metadata in standardized formats?	Boolean (Yes/No)	yes	yes
Comments on metadata completeness	Open text	everrthing needed	Missing metadata from the DOI werent availaboe
Was model training info (config, metrics, weights) captured fully?	Boolean (Yes/No)	yes	Yes
Ease of connecting MLflow or notebook metadata	Likert (1-5)	5	3
Did the metadata export support multiple formats (e.g., PROV-O, MLSEA)?	Boolean (Yes/No)	yes	yes
Comments on export and metadata formats	Open text	over all functional but with few minor discrepencies	overwhelming info
Could you trace exactly how a specific result was produced (e.g., which data, code version, params)?	Boolean (Yes/No)	yes	yes
How confident are you in the completeness of provenance for debugging?	Likert (1-5)	5	4
Were you able to identify experiments with outdated or faulty data/code versions?	Boolean (Yes/No)	yes	yes
How easy was it to find affected researchers/users for notification?	Likert (1-5)	5	5
Were you able to review training configs (e.g., data splits, hyperparams) across experiments?	Boolean (Yes/No)	yes with minor mapping issues	yes
Rate clarity of experiment configuration inspection	Likert (1-5)	5	4
Could you link which models were trained on which datasets (and versions)?	Boolean (Yes/No)	yes	yes
Were performance metrics sufficiently detailed and accessible?	Likert (1-5)	5	5
Could you detect outdated forks or divergent versions in data/code repos?	Boolean (Yes/No)	yes	yes
Rate ease of identifying and prompting synchronization	Likert (1-5)	3	4
How did provenance metadata assist in overall auditing, debugging, and reproducibility?	Open text	clear and helpul to do the above things esp with req and dataset info and hyper params	It clearly documented each step of the workflow, making it easy to trace errors, verify results, and repeat experiments.
Rate intuitiveness of provenance dashboards and widgets	Likert (1-5)	5	4
Did visualizations help you identify inefficiencies or errors?	Boolean (Yes/No)	yes	yes
Rate clarity of traceability flows and preprocessing traces	Likert (1-5)	5	5
Describe any workflow improvements or corrections made due to provenance insights	Open text	unsure,	More narrowed to the specific scope,
Any improvements to visualization or interaction?	Open text	no, everything seems easily understandable, easy to navigate	The visualizations were informative but could be more interactive and customizable for better usability.
Rate how easy it was to understand the metadata schema	Likert (1-5)	4	
Were you able to export metadata to standard/open formats easily?	Boolean (Yes/No)	yes	yes
Is the metadata usable for integration with other platforms/tools?	Boolean (Yes/No)	yes	yes
Thoughts on schema design and export capabilities	Open text	could be simplified. but provides necessary info	The schema was comprehensive and the export worked well, though simplifying the structure could help new users.
Learning Curve (time/effort to onboard)	Numeric (hours/minutes)	2-3 hours	
ease of use	Likert (1-5)	5	4
visual clarity	Likert (1-5)	4.5	4.5
Overall satisfaction	Likert (1-5)	4	4
Documentation quality	Likert (1-5)	5	4
Recommendations	Open text	few mapping issues, complexity of dbrepo linking, visualization of export really help but it can be simplifieed, Queries helped to navigate	Improve interactivity in the UI, streamline export options, and offer quick-start guides or tooltips.
feedback	Open text	makes more sense for a bigger project with more collaborator as the functionality are detailed, not relevent for small group , for small group is time consuming	

Figure D.1: User study form

Description: The above screenshot is the anonymized version of the user study questionnaire. It outlines the questions asked.

E Ontologies Used

Table E.1: Ontologies and Their Purpose

Ontology	**Purpose in System**	**Example Term**
PROV-O	Provenance: activity, agent, entity	`prov:wasGeneratedBy`
FAIR	Basic metadata descriptors	`dcterms:title`
FAIR4ML	ML-specific workflow logging	`fair4ml:trainingEndTime`
MLSEA	Training logic + metrics	`mls:maxDepth`
Croissant	Schema-level structure	`croissant:featureList`

F Glossary

VRE Virtual Research Environment — A collaborative, web-based infrastructure that enables researchers to access data, run computational workflows, and share results in a reproducible manner.

FAIR Findable, Accessible, Interoperable, Reusable — A set of guiding principles to improve the reusability of digital assets through better metadata and semantic standards.

MLflow An open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

MLSEA Machine Learning Scientific Experimentation and Annotation — A domain ontology for annotating ML-specific experiment metadata including metrics, hyperparameters, and provenance.

FAIR4ML Extension of FAIR for machine learning pipelines, enabling finer-grained semantic description of ML workflows and artifacts.

PROV-O W3C Provenance Ontology — A standard for representing provenance information about entities, activities, and agents.

Croissant A metadata model developed by Hugging Face for describing ML models in a way that supports discovery, reuse, and explainability.

DCAT Data Catalog Vocabulary — A W3C vocabulary used to describe datasets and data catalogs, often used with FAIR data portals.

Dublin Core (DC) A set of vocabulary terms used to describe web resources such as datasets, publications, and tools in a standardized way.

JSON-LD JavaScript Object Notation for Linked Data — A lightweight syntax to serialize linked data using JSON, compatible with RDF and se-

mantic web tools.

RDF Resource Description Framework — A standard model for data interchange on the web using subject-predicate-object triples.

RDF/XML An XML-based syntax for expressing RDF graphs, enabling compatibility with legacy systems and semantic tools.

SPARQL A query language and protocol for retrieving and manipulating RDF data across diverse linked datasets.

InvenioRDM A turn-key research data management platform built on top of Invenio, used for publishing FAIR-aligned datasets and metadata.

DBRepo A custom PostgreSQL-backed database used in this framework for storing ML metadata, linked to experiment runs and structured ontologies.

Streamlit An open-source Python library for building custom web apps for machine learning and data science workflows with minimal effort.

Git A distributed version control system for tracking changes in source code and configurations, essential for reproducible research.

Ontology A formal representation of knowledge as a set of concepts and relationships within a domain, used to enable semantic interoperability.

G Code and Repository Snapshot

Repository and DOI

- GitHub: https://github.com/reemadass26/Framework_Evaluation.git

G.1 Repository Directory Tree

```
Framework_Evaluation/
 - notebook/
    -RQ_notebook/
        - Thesis_Framework.py
    - Evaluation/
        - RQ_Evaluation.ipynb
    - visualization.py
```