

Article

Verifying Machine Learning Interpretability and Explainability Requirements Through Provenance

Lynn Vonderhaar * , Juan Couder , Tyler Thomas Procko , Eva Lueddeke, Daryela Cisneros and Omar Ochoa 

Department of Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA; ortizcoj@my.erau.edu (J.C.); prockot@my.erau.edu (T.T.P.); lueddeke@my.erau.edu (E.L.); cisnerd4@my.erau.edu (D.C.); ochoao@erau.edu (O.O.)

* Correspondence: vonderhl@my.erau.edu

Abstract

Machine learning (ML) engineering increasingly incorporates principles from software and requirements engineering to improve development rigor; however, key non-functional requirements (NFRs) such as interpretability and explainability remain difficult to specify and verify using traditional requirements practices. Although prior work defines these qualities conceptually, their lack of measurable criteria prevents systematic verification. This paper presents a novel provenance-driven approach that decomposes ML interpretability and explainability NFRs into verifiable functional requirements (FRs) by leveraging model and data provenance to make model behavior transparent. The approach identifies the specific provenance artifacts required to validate each FR and demonstrates how their verification collectively establishes compliance with interpretability and explainability NFRs. The results show that ML provenance can operationalize otherwise abstract NFRs, transforming interpretability and explainability into quantifiable, testable properties and enabling more rigorous, requirements-based ML engineering.

Keywords: machine learning; requirements engineering; interpretability; explainability; provenance; verification

1. Introduction

As Machine Learning (ML) becomes increasingly ubiquitous, it is critical that its development embodies the same level of rigor as that of traditional software systems. This is especially true as safety-critical domains—which tend to have stricter rules and regulations, e.g., the European Union’s General Data Protection Regulation (GDPR)—explore the potential uses of ML [1,2]. Towards this goal, the existing literature has drawn parallels between ML Engineering and Requirements Engineering (RE), including defining ML-specific Non-Functional Requirements (NFRs) [3]. However, these definitions often lack a clear description of how the NFRs can be measured [3,4]. Some literature notes that in addition to NFR measurement being difficult, decompositions from one NFR to another might also change for ML systems [4]. This work addresses both challenges for interpretability and explainability, which are defined in the literature as ML NFRs [3]. Both NFRs will be verified by decomposing them into provenance and using that provenance to implicitly verify it [5]. In the context of this work, verification refers to software verification by testing the system to verify functional requirements. Note that this paper does not claim to use formal software verification, but rather verification of requirement specifications.



Academic Editor: Haipeng Cai

Received: 26 November 2025

Revised: 29 January 2026

Accepted: 12 February 2026

Published: 14 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)

[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Specifically, this work focuses on interpretability and explainability requirements because of their importance throughout the literature.

Provenance is the origin and lineage of something [6,7]. In the case of ML, it is the origin and lineage of model behavior. It includes various data throughout the model development process that contributes to the development of model behavior including datasets, preprocessing steps, and hyperparameters [5]. This contributes to behavioral transparency, which lends itself well to ML interpretability and explainability methods, e.g., anchors or saliency maps. Therefore, this work proposes using provenance to decompose and verify model interpretability and explainability requirements by specifying which provenance data to save.

It is important to note that although interpretability and explainability are often used synonymously in the literature, they are distinct ML model qualities. The distinction between the two will become more important as regulations for ML systems emerge and RE for ML becomes commonplace. This work will also contribute to the literature by categorizing interpretability and explainability methods in line with their NFR definitions.

The contributions of this work are as follows:

1. Categorizing interpretability and explainability methods in line with their NFR definitions.
2. Decomposing ML interpretability and explainability requirements into provenance requirements by specifying what provenance will be saved.
3. Verifying ML interpretability and explainability requirements using provenance Functional Requirements (FRs).
4. Identifying trends in the decomposition of interpretability and explainability NFRs.

The rest of this paper is structured as follows: Section 2 describes existing work in the field, Section 3 gives important background information for understanding the approach, Section 4 gives a detailed description of the approach, Section 5 verifies the approach with case studies, Section 6 discusses the results, Section 7 analyzes limitations and future work for this paper, and Section 8 concludes the paper.

2. Related Work

Our previous work has identified provenance as an ML-specific NFR that can decompose interpretability [5]. However, it does not verify the method with case studies which this work does. Other authors have linked provenance to interpretability or other similar ML quality attributes [6,8]. Kale et al. show how provenance contributes to model transparency and by extension, to model explainability [6]. Their literature review discusses how provenance can increase the trustworthiness of ML systems and also mentions how PROV-O and JSON can be used to format and store ML provenance. Although this has greatly influenced this work, it does not show practical examples of how provenance can be used to verify such quality attributes. Meanwhile, Nakagawa et al. connect provenance and model traceability to ML quality assurance [8]. The authors extend W3C's PROV model to include ML quality characteristics. Although the authors connect provenance to quality verification, this work is distinct in that it frames provenance as a requirement specification that can be used to verify interpretability NFRs. Additionally, Schlegel and Sattler provide a detailed discussion of provenance data that is generated during the ML development lifecycle and that should be saved [9]. The authors describe some phases of ML development and provide many suggestions on what provenance data should be saved within each development phase. However, Schlegel and Sattler do not relate provenance to the RE process, nor do they give much discussion on how provenance relates to other ML quality attributes, e.g., transparency and traceability.

More broadly, the existing literature notes the importance of provenance for software because of the use of external source code, e.g., libraries, and the evolution of software

functionality over time [10–13]. Software provenance can become complex based on package versions, licenses, and tweaks [10,11,14]. Davies et al. offer a provenance tracking system based on early forensic approaches that builds software provenance using homing questions to narrow the search space for building a system's provenance [10,11]; Pham et al. collected package versions and dependencies to help in system reproducibility [14]. Meanwhile Godfrey notes that provenance is also important to understand the evolution of a software system [13].

Some authors have focused scopes for software provenance collection. For instance, Chittimalli and Naik propose variable provenance to trace the values of variables through software [15]. Other authors use a broader provenance scope [12,16–18]. Wendel et al. use provenance to answer the question of why the build fails currently [16]. Their approach includes provenance not only from their code version control system, but also from their issue tracker and documentation system. Costa et al. describe an extension of the W3C PROV model for software processes to include provenance from product data, process data, and organizational data [17]. Similarly, Sáenz-Adán et al. offer a method to convert Universal Modeling Language (UML) diagrams to fit the PROV model [18]. Using a different approach, Bose et al. provide a framework for capturing and documenting software provenance using blockchain technology [12]. Meanwhile Schreiber et al. extract provenance data from code versioning tools and visualize the data within graphs [19].

Prior work on XAI verification has primarily focused on evaluating explanation properties such as faithfulness, robustness, and stability. Early studies demonstrated that many popular explanation methods fail basic sanity checks and can be largely independent of the trained model, raising concerns about their correctness [20]. Subsequent work showed that explanations are often highly sensitive to small input perturbations or adversarial manipulation, even when model predictions remain unchanged [21]. To address these issues, researchers have proposed verification and evaluation techniques based on perturbation analysis, robustness testing, and quantitative faithfulness metrics [22].

Despite these advances, existing XAI verification approaches remain largely model-centric and insufficiently grounded in RE principles. Most methods implicitly assume generic explanation objectives and do not rely on explicitly specified explanation requirements derived from stakeholder goals or domain constraints. Consequently, explanation correctness is assessed using heuristic metrics rather than being verified against declared system requirements. Moreover, current tools lack traceability between explanation properties and stakeholder-driven requirements such as auditability, safety assurance, or regulatory compliance, allowing explanations to satisfy verification metrics while failing their intended purpose [23]. Verification is also typically performed post hoc and independently of the system lifecycle, ignoring requirement evolution and the presence of multiple stakeholders with distinct explanation needs [24]. As a result, while existing XAI verification methods expose important weaknesses in explanation techniques, they fall short of ensuring that explanations satisfy real-world requirements, motivating the need for requirements-aware XAI verification frameworks.

3. Background

Before describing this paper's approach, it is critical to first have clear definitions for RE, ML interpretability, ML explainability, and ML provenance.

3.1. Requirements Engineering

RE is the systematic process of defining, documenting, and managing the requirements of a software or system. It ensures that the final product meets the needs of its stakeholders (customers, users, regulators, developers, etc.) and reduces the risk of building the “wrong”

system [25]. RE is a crucial phase in the traditional Software Development Life Cycle (SDLC) because requirements act as the foundation for design, implementation, and testing. RE includes a series of activities which lead to requirement specifications, i.e., the formal, documented statement of what the system must do and the constraints under which it must operate. After these specifications are complete and finalized, they are used for Validation and Verification (V&V), ensuring the requirements are correct, complete, and reflect the stakeholders' needs [26].

RE for ML-based systems is still an emerging topic that requires additional application and adaptation of RE practices to the development of ML-based systems, with attention to data, model behavior, and uncertainty [27]. RE for ML is still the process of gathering, analyzing, documenting, validating, and managing requirements for ML-based systems, but it now handles not only functional needs, but also data quality, model performance, and fairness. At the end of this process, the requirements are still captured in requirement specifications, much like for traditional software [28]. However, the content of requirements specifications is different from those of traditional software as it must now capture data requirements, ML evaluation metrics, newly emerging NFRs, and system monitoring and retraining [29]. V&V is also still important in RE for ML-based systems, but the literature notes that there are not yet methods for verifying some of the newly emerging NFRs [3,4,30]. Habibullah et al. present a quality model for ML-based systems, but note that some ML-specific NFRs, e.g., interpretability and explainability, do not yet have measurements and are therefore difficult to verify [3]. This work focuses on the verification of interpretability and explainability NFRs for ML models.

3.2. Interpretability Versus Explainability

The literature often uses interpretability and explainability synonymously, but they are distinct qualities. The two NFRs occupy distinct logical categories: while interpretability concerns transparency of data and model semantics, explainability concerns communicability of model decisions [3]. Interpretability is not merely a premise from which explainability necessarily follows; one may have an interpretable model that fails to be explainable to non-experts, or conversely, an opaque model that offers solid post hoc explanations. The two NFRs are orthogonal yet complementary, engendering, respectively, epistemic access (knowing how) and semantic communication (knowing why). Definitions are provided in the context of ML for a common understanding.

- Interpretability: typically defined as the degree to which a human can understand internal model mechanics and data patterns, i.e., how inputs, features, parameters, etc., map to outputs.
- Explainability: typically defined as the degree to which a human can understand the reasons for model output, i.e., why a model made some decisions.

In terms of causal realism, explanations (operationalizations of explainability [31]) are descriptions of causality, i.e., chains of causes and effects. Explainability is difficult to quantify, as it is subjective from human to human; however, it may be asserted that explainability is proportional to the number of accurate explanations provided [32]. The next two sub-sections will identify specific methods for interpretability and explainability to make decomposition accessible.

3.3. ML Interpretability

ML model interpretability encompasses methods that allow model users to understand the underlying patterns within the data [3,33]. This is distinct from explainability methods, which are means of understanding the model mechanics when making decisions. In other

words, interpretability aims to provide insight into the data, while explainability provides insight into the model.

There are many existing interpretability methods and various surveys that organize them in unique ways [1,34–36]. Some examples are:

- Global versus local interpretability where global interpretability methods show the model's generalized behavior, while local interpretability methods clarify reasons for a specific decision [33,34].
- Pre hoc versus post hoc interpretability where pre hoc interpretability design takes place before and during model training, while post hoc interpretability methods are applied only after model training [1,35].
- Model-specific versus model-agnostic interpretability where model-specific methods change based on the architecture of the black box model while model-agnostic methods do not [36].
- Explanation type where interpretability methods are categorized based on their output, e.g., rule-based interpretability and saliency maps [37].

For the purposes of this work, interpretability methods are categorized by their underlying description mechanism so that a single example from each category will suffice to represent a provenance decomposition for each method in that category. Therefore, each category maps to similar provenance decompositions. The categorization of popular interpretability methods is shown in Table 1. Table 1 is not meant to be a full survey of interpretability methods, but rather examples of popular methods for the purpose of verifying this work.

Table 1. Categorization of popular interpretability methods. Synthesized from [1,4,15–18,20–24].

Category	Interpretability Method	Output Format
Inherently interpretable models	Linear regression	Mathematical function
	Logistic regression	Mathematical function
	Decision trees	If-then rule(s)
	Decision rules	If-then rule(s)
Surrogate models	Local Interpretable Model-agnostic Explanations (LIME)	Any output format of inherently interpretable models, e.g., mathematical function or if-then rules
Feature attribution	Shapley values	Mathematical function
	SHapley Additive exPlanations (SHAP)	Mathematical function
	Counterfactual explanations	If-then rule(s)
	Anchors	If-then rule(s)
	Occlusion tests	Image regions of importance
Simplification	Functional decomposition	Mathematical function

The interpretability methods in Table 1 are separated into three categories based on their underlying description mechanism: inherently interpretable models, surrogate models, and feature attribution. Inherently interpretable models have a simpler structure that allows developers and users to simulate their behavior [35]. Surrogate models are inherently interpretable models that train alongside more complex models and provide an approximate explanation for the complex model's behavior [34]. Feature attribution includes methods that directly describe the black box model's behavior with respect to

input values [38]. While Table 1 is not a complete survey of interpretability methods, it provides some popular methods with which to verify this paper's approach.

The choice of which interpretability method to use depends on the problem's complexity, the input type, and the desired output format. Simple problems that can use inherently interpretable models generally should. If the problem is too complex for inherently interpretable models, but the engineer would like to simulate the model's behavior, they may choose a surrogate model. Otherwise, method choice relies partially on input type, e.g., occlusion tests can only be used for images, and partially on the desired output format, e.g., choosing between Shapley values or counterfactual explanations.

3.4. ML Explainability

As stated previously, explainability refers to the ability of a human to understand a model's mechanics regarding specific decisions. Therefore, the explainability methods presented in this brief survey are those that describe specific model decisions. Table 2 shows a brief survey of popular explainability methods. As with the interpretability survey, this is not meant to be a systematic survey of explainability methods, but rather a brief survey to show examples of how provenance can decompose explainability NFRs. As with Table 1, the explainability methods in Table 2 are organized by their underlying description mechanism. As with the interpretability methods, the categorization of these explainability methods maps to similar provenance decompositions.

Table 2. Categorization of popular explainability methods. Synthesized from [39,40].

Category	Explainability Method	Output Format
Feature attribution	Saliency mapping	Pixel map
	Explanatory graphs	Image regions of importance
	Deconvolutional Neural Network	Pixel map
	Feature inversion	Image
	Visual backpropagation	Pixel map

Interestingly, because explainability methods look at internal model mechanics for specific decisions, all of the explainability methods in this survey can be grouped into the feature attribution category. The methods describe how the model used features from a specific input to make a decision. There are, however, still reasons that engineers may choose one method over another for their application. Some output formats are different, e.g., feature inversion outputs a reconstructed image while saliency mapping provides a pixel map. Additionally, though various methods output a pixel map, they use varying methods for calculating feature relevance, which may be important to method selection.

3.5. ML Provenance

The final necessary background topic is provenance. As previously stated, provenance means the origin of something [6,7]. In ML, this refers to the origin, history, and transformations of data, models, and processes throughout the ML lifecycle [41]. In our previous work, we defined many types of provenance data in ML development, but this work focuses on two main categories: data and model [5]. The specific provenance in each of these categories is shown in Table 3.

Data provenance is the record that describes the origins and processing of data. Data provenance helps with fairness, accountability, transparency, and explainability in AI systems [42]. Data provenance focuses on the lineage and history of data, answering questions about its source or origin, the methods of collection (such as human input, sensors, or web scraping), and the types of transformations applied, including cleaning,

normalization, feature extraction, augmentation, or aggregation. It also captures who made modifications to the data, essentially tracing how the data changed and reached its current state through specific contexts or decisions [42]. Data provenance focuses on documenting the history of the data, from its origin to all the evolution and transformations until it reached its final state, so that it can be trusted, reused, and reproduced. Model provenance is very similar to data provenance, but it focuses on the model rather than the data. It captures the creation, origin, and evolution of the model by addressing questions related to its architecture, different versions, the changes it undergoes across those versions, the parameters and hyperparameters used, as well as the code, libraries, and environments involved [9]. Model provenance focuses on the history of how a model was built, trained, and modified, including its architecture, parameters, datasets, and environment, by keeping a record of all the changes the model itself has gone through.

ML provenance—just like regular provenance—is a key aspect for increasing the authenticity of, and trust in, an ML model by increasing the chances of reproducing the same conditions the model had when training, and making the results as reproducible as possible [41]. Without being able to get similar results as the creators, there is no way to guarantee that the behavior demonstrated by the model is not an outlier instead of the norm.

Table 3. Types of ML provenance data [5].

Requirements	Data	Model
- Customer requirement artifacts [9] - Model quality attributes - Requirement prioritization - Requirement specifications - Constraints (computational, data, privacy, time, budget) - Developers and dates associated with the requirements phase - Decisions associated with the requirements phase	- Datasets [9] - Data type - Data source - Data authenticity (real or synthetic) - Data metadata, as applicable (resolution, size/length, contrast, etc.) - Synthetic data metadata, as applicable (prompt, seed, etc.) - Method of collection, collector, date of collection [9] - Inclusion/exclusion criteria (person, date) - Preprocessing steps (transformations, cleaning, denoising, resizing, cropping, normalization) (person, date for each) [9] - Labeling (annotation type, annotator, date of annotating) [9] - Feature selection, if relevant (person, date) [9] - Training/validation/testing split - Training/validation/testing dataset class proportions	- Pretraining provenance, if applicable and available (model comparison and selection) (person and date) - Model size - Model speed - Parameters (weights and biases) [9] - Hyperparameters (number of layers, learning rate, batch size, dropout) [9] - Architecture comparisons - Optimization algorithms [9] - Pipelines [9] - Execution logs and statistics [9] - Environment configurations [9] - Source code (Jupyter notebooks, library dependencies) [9] - Gradients

Table 3. Cont.

Requirements	Data	Model
Verification and Behavior	Deployment	Maintenance and Revisions
- Performance metrics (accuracy, precision, recall, F1-Score) [9] - Synthetic dataset verification results, if applicable [43] - Testing history - Model versions	- Online or offline mode - Execution logs [9] - Prediction logs - Operation location (local, edge device, server, cloud) [9] - Licenses [9] - Environmental dependences [9] - Containers (Docker, Conda, virtual machines)	- Data changes - Model retraining - Versioning (data, model, verification) [9] - Requirement changes - Reverts [9]

4. Approach

This approach uses ML provenance as a lower-level NFR to decompose interpretability, then it decomposes provenance into verifiable FRs by defining what provenance data a system will collect. The verification of these FRs then by extension verifies that the provenance and interpretability NFRs have been met for the system. In software, FRs are verified by directly ensuring that the code satisfies the requirements. Therefore, saving provenance in the code satisfies and verifies the provenance requirements.

Given that interpretability is the ability for humans to draw connections among model data, training, and behavior, the storage of model provenance throughout development allows for humans to trace behavior from training data to model deployment, making the model interpretable. For the purpose of verifying this approach, this section includes requirement specifications and implementation for one interpretability and explainability method from each format category. For maximum scalability, the ideal amount of provenance to be stored is the least amount necessary to verify the NFR in question.

Provenance Storage

There are several platforms meant to help developers save model provenance including MLFlow, Data Version Control (DVC), Weights & Biases (W&B), Neptune AI, and Comet [44–48]. Each of these tools can track dataset versions, model versions, hyperparameters, and other common provenance data. However, they are not built to handle more unique provenance data as defined in our previous work, e.g., preprocessing steps or feature selection choices [5]. To accommodate additional necessary provenance tracking, this approach uses yProv4ML, which is compliant with the World Wide Web Consortium's (W3C's) PROVenance Ontology (PROV-O) [49].

PROV-O is an ontology defined using the Web Ontology Language (OWL) that expresses the PROV Data Model (PROV-DM) [50,51]. Its three “Starting Point” classes, Activity, Entity, and Agent, are shown in Figure 1 along with some of their subclasses [50]. Ontologies represent the structure of entities within a domain [52]. This ontology provides a schema, or structure, for the collected provenance data. yProv4ML then instantiates the structure using PROV-JSON, which can easily be formatted into a graph for readability [49].

yProv4ML, version 2.0.3, is a Python library that handles all the provenance formatting, making this approach accessible to ML developers, even those without ontology experience [49]. Its usage requires only a small learning curve to know the appropriate function calls for each provenance data type. The output can be viewed either as JSON or as a graph.

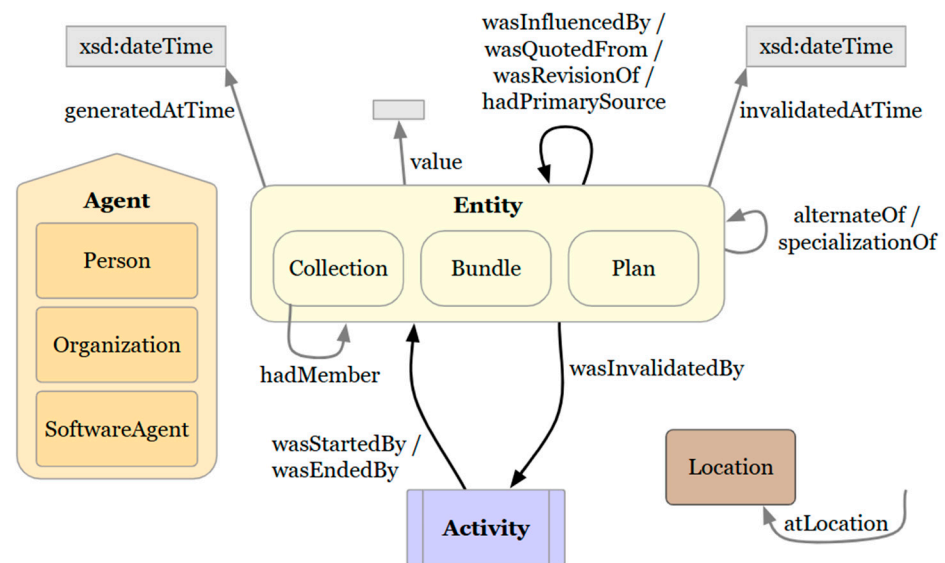


Figure 1. The three “Starting Point” classes and some of their subclasses in PROV-O [50].

5. Verification

The following sub-sections will verify the approach via interpretability and explainability case studies.

5.1. Linear Regression Case Study

Linear regression models are inherently interpretable models. Therefore, saving development provenance allows the model’s training and predictions to be simulated and verifies the interpretability NFR. This process is shown in this section.

5.1.1. Requirement Specifications–Linear Regression

The first step is deciding what provenance to save to decompose and verify the interpretability NFR. This provenance may not be the same for all models. The purpose of interpretability is to gather information about the underlying trends in the data and to simulate the model’s behavior as a result. To do that for linear regression, one must be able to reproduce the line of best fit for the data and make future predictions using that function. Therefore, the necessary provenance includes the training and testing datasets, the preprocessing steps, e.g., any data trimming, and the resulting function for the line of best fit. This example uses simple linear regression, so these provenance types are sufficient; however, additional provenance may be important for multiple linear regression problems, e.g., feature selection. The requirement specifications, with NFRs in *italics*, are as follows:

[Req. 1] *The model’s development pipeline shall incorporate prediction interpretations for the user.*

[Req. 2] *The model’s development artifacts shall be traceable throughout its lifecycle to ensure interpretability.*

[Req. 3] *The model’s development pipeline shall save development provenance to ensure development artifact traceability to model specifications.*

[Req. 4] The model’s development pipeline shall save the training dataset to the provenance documentation.

[Req. 5] The model’s development pipeline shall save the testing dataset to the provenance documentation.

[Req. 6] The model’s development pipeline shall save the data preprocessing steps to the provenance documentation.

[Req. 7] The model's development pipeline shall save the model's linear regression function to the provenance documentation.

5.1.2. Requirement Specification Verification–Linear Regression

A simple Kaggle example of linear regression is sufficient for the purposes of this case study [53]. Per the requirement specifications for this case study, the model's development pipeline needs to save the training and testing datasets, the data preprocessing steps, and the model's linear regression function. Figure 2 is an example of how the training dataset is stored to the provenance documentation.

```
yprov4ml.start_run(
    prov_user_namespace="www.example.org",
    experiment_name="linear_regression",
    provenance_save_dir="/save/path/"
)
yprov4ml.log_dataset(train_data, "train_data")
yprov4ml.log_dataset(test_data, "test_data")
yprov4ml.end_run(create_graph=True, create_svg=True)
```

Figure 2. Example Python code for saving provenance data.

The full code for this case study, along with the full provenance JSON file and provenance graph can be found at <https://github.com/lynndalou/ProvInterpretExplain>, accessed on 13 December 2025. Two sections of the provenance graph are shown in Figures 3 and 4. Figure 3 shows how the model is saved to the graph and Figure 4 shows the model parameters are saved to the graph.

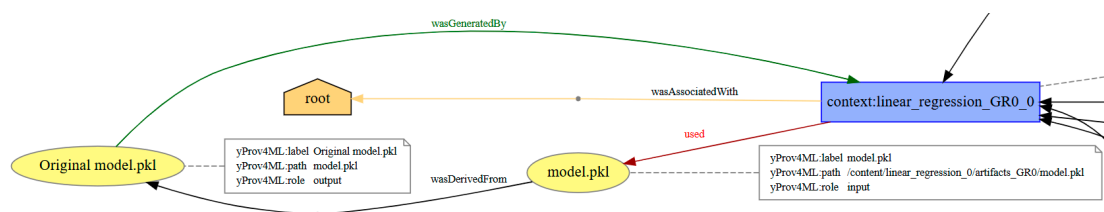


Figure 3. Section of the linear regression provenance graph. The nodes and arrows align with the W3C's PROV-O.

dropna_testing_data_cleanup	x	y0	77	79.7751521	21	23.1772792	22	25.6092623	20	17.8573884	...
dropna_training_data_cleanup	x	y0	24.0	21.5494521	50.0	47.4644632	15.0	17.2186563	38.0	36.58...	
execution_end_time				1760554687.6064146							
execution_start_time				1760554649.3482456							
expand_dims_testing_data				[[9.32668894e-01] [-1.00278099e+00] [-9.68219386e-01] [-1.03734260e+00] [-4.84356915e-01] [-1....							
expand_dims_training_data				[[-8.99096176e-01] [-4.94443564e-04] [-1.21015062e+00] [-4.15233705e-01] [1.27828495e+00] [-4....							
standardize_testing_data				[9.32668894e-01 -1.00278099e+00 -9.68219386e-01 -1.03734260e+00 -4.84356915e-01 -1.21015062e+00 4...							
standardize_training_data				[-8.99096176e-01 -4.94443564e-04 -1.21015062e+00 -4.15233705e-01 1.27828495e+00 -4.84356915e-01 -1...							
yProv4ML:artifact_uri				/content/linear_regression_0/artifacts_GR0							
yProv4ML:experiment_dir				/content/linear_regression_0							
yProv4ML:experiment_name				linear_regression_GR0_0							
yProv4ML:global_rank				0							
yProv4ML:level				0							
yProv4ML:provenance_path				/content/							
yProv4ML:python_version				3.12.12 (main, Oct 10 2025, 08:52:57) [GCC 11.4.0]							
yProv4ML:run_id				0							

Figure 4. Model parameters saved to the provenance graph.

The purpose of saving this provenance data is to interpret and simulate the model's behavior to verify the overarching interpretability NFR. This verification follows the provenance step-by-step to achieve the same result as the Kaggle linear regression example [33]. Microsoft Excel is used for verifying this experimental result. The full Excel file can be

found at <https://github.com/lynndalou/ProvInterpretExplain>, accessed on 13 December 2025. Per the saved provenance data, the following steps were taken:

1. Removing N/A or empty datapoints from the training and testing datasets.
2. Standardizing the independent variable for the training and testing datasets.
3. Using the Analysis Toolpak to calculate the linear regression line [54].

These steps lead to the same linear regression function as the Kaggle code, thereby simulating the behavior of the ML model.

5.2. Counterfactual Explanation Case Study

Inherently interpretable models are unique in that provenance allows their behavior to be simulated, as was shown in the linear regression case study. Other methods are more complex because they can no longer simulate the model's behavior directly. However, provenance can still be used to map the model's behavior to an input space as a way to understand it. Counterfactual Explanations (CEs) will be used in this case study to demonstrate how to do this.

5.2.1. Requirement Specifications–Counterfactual Explanations

[Req. 1] The model's development pipeline shall incorporate prediction interpretations for the user.

[Req. 2] The model's development artifacts shall be traceable throughout its lifecycle to ensure interpretability.

[Req. 3] The model's development pipeline shall save development provenance to ensure development artifact traceability to model specifications.

[Req. 4] The model's development pipeline shall save the training dataset to the provenance documentation.

[Req. 5] The model's development pipeline shall save the testing dataset to the provenance documentation.

[Req. 6] The model's development pipeline shall save the data preprocessing steps to the provenance documentation.

[Req. 7] The model's development pipeline shall save the selected features to the provenance documentation.

[Req. 8] The model's development pipeline shall save the counterfactual explanations to the provenance documentation.

5.2.2. Requirement Specification Verification–Counterfactual Explanations

This case study utilizes a simple binary classification example with Diverse Counterfactual Explanations (DiCE) [55,56]. The full code for this case study and the provenance data saved during the case study can be found at <https://github.com/lynndalou/ProvInterpretExplain>, accessed on 13 December 2025. The goal of this case study is still to understand the black box model's behavior in human-understandable terms, but it is accomplished a bit differently than the linear regression example because the behavior can no longer be simulated. However, provenance data can still help to verify this interpretability requirement.

CEs offer ways to change the inputs to a model that would produce a different output [55,57]. Therefore, these explanations can be used to map the input space that produces the possible outputs, as is shown in Figure 5. The provenance data required to accomplish this understanding is outlined in the requirement specifications for this case study. Some of the provenance data is the same as in the first case study, but this case study requires feature selection data. As is shown in Figure 5, three of the selected features are "age", "education", and "hours per week". The space then shows "positive" predications,

in this case predictions of “high income”, within the sphere and “negative” predictions outside of the sphere. Model developers and users can utilize this provenance-based technique to visualize how input changes affect the model’s predictions.

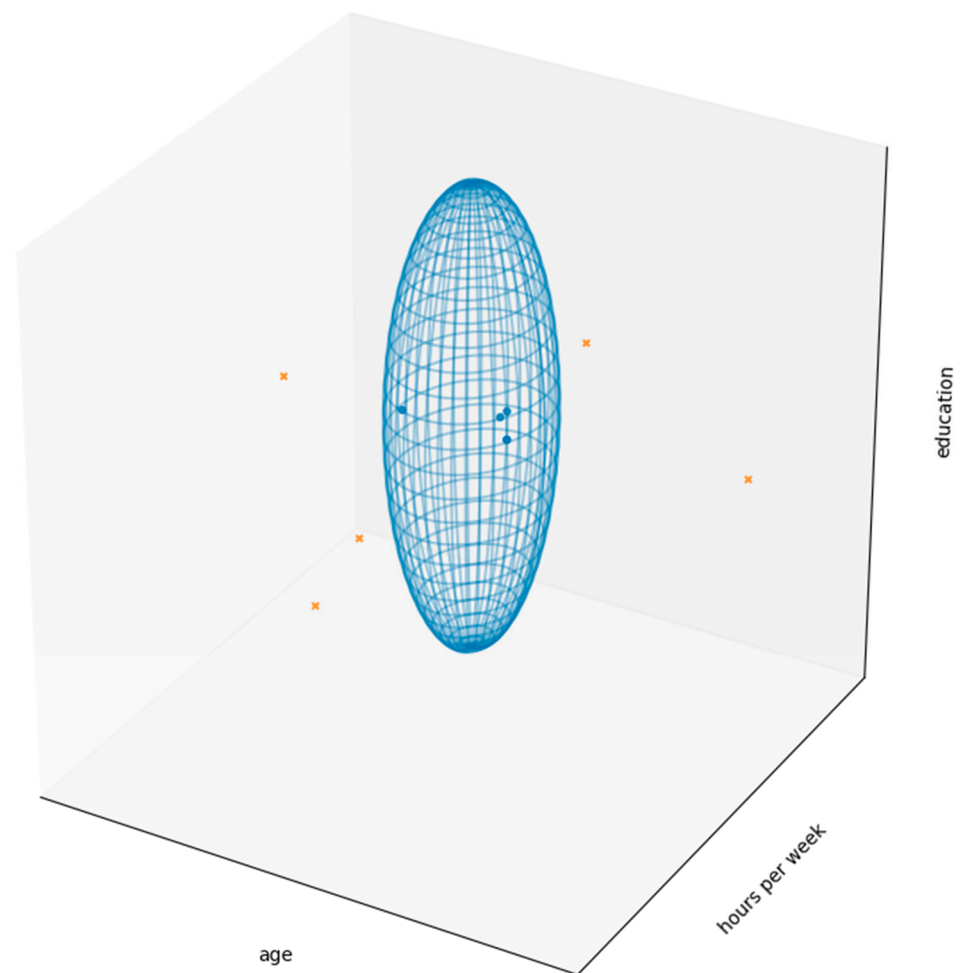


Figure 5. A depiction of how counterfactuals can verify an interpretability requirement. The blue dots show positive predictions and the orange dots show negative predictions.

5.3. Functional Decomposition Case Study

Functional decomposition mathematically simplifies complex functions to show underlying patterns [57,58]. There are several methods for functional decomposition. This case study follows a Kaggle example for a Simple Moving Average (SMA) decomposition [58]. An SMA decomposition involves taking the rolling mean of time series data, thereby forming a trendline. The residual function can be found by subtracting the rolling mean from each individual data point.

5.3.1. Requirement Specifications–Functional Decomposition

[Req. 1] *The model’s development pipeline shall incorporate prediction interpretations for the user.*

[Req. 2] *The model’s development artifacts shall be traceable throughout its lifecycle to ensure interpretability.*

[Req. 3] *The model’s development pipeline shall save development provenance to ensure development artifact traceability to model specifications.*

[Req. 4] *The model’s development pipeline shall save the training dataset to the provenance documentation.*

[Req. 5] The model's development pipeline shall save the rolling mean to the provenance documentation.

[Req. 6] The model's development pipeline shall save the residual data to the provenance documentation.

5.3.2. Requirement Specification Verification–Functional Decomposition

As mentioned previously, this case study uses a simple Kaggle example to show how provenance can be saved to perform SMA decomposition [58]. The requirement specifications state what data is required to produce the visualization shown in Figure 6, which decomposes the time series data into a trendline and the residual.

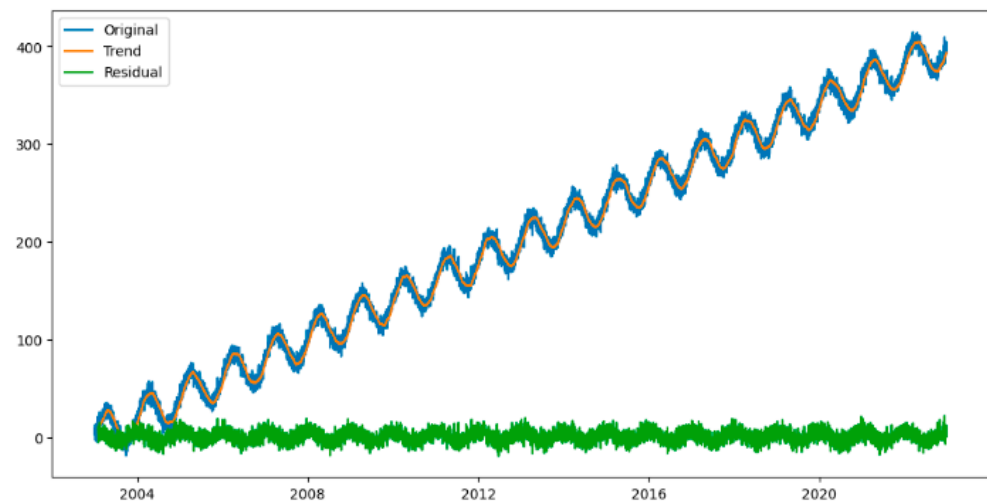


Figure 6. Functional decomposition example [58].

5.4. Saliency Mapping Case Study

The requirement specifications for explainability methods are simpler than those for interpretability methods because they are only meant to understand the model mechanics for a specific decision as opposed to understanding the underlying patterns within the data and the model. Therefore, the only necessary provenance will be the model weights and whatever provenance is necessary for building the desired visualization. This case study shows the verification for saliency mapping.

In the context of ML, saliency maps are a method of feature attribution for a specific model decision [59]. This method calculates a relevance score for each pixel within an image in relation to the final decision. There are various relevance calculations to choose from, e.g., Layer-wise Relevance Propagation (LRP), with the saliency map being the visualization of the pixel relevancy. The provenance requirement specifications will be identical regardless of the chosen relevancy calculation. Though the layer gradients may vary slightly depending on the calculation, it does not change what provenance must be saved to verify the explainability NFR.

5.4.1. Requirement Specifications–Saliency Mapping

[Req. 1] The model's development pipeline shall incorporate prediction explanations for the user.

[Req. 2] The model's development artifacts shall be traceable throughout its lifecycle to ensure explainability.

[Req. 3] The model's development pipeline shall save development provenance to ensure development artifact traceability to model specifications.

[Req. 4] The model's development pipeline shall save the model weights to the provenance documentation.

[Req. 5] The model's development pipeline shall save the final layer gradients to the provenance documentation.

5.4.2. Requirement Specification Verification–Saliency Mapping

This case study uses a simple Kaggle example of saliency mapping on the MNIST dataset with some small versioning modifications [40]. Based on the requirement specifications, the only two pieces of necessary provenance data are the model weights and the gradients from the final network layer. This provenance data can be used to recreate the model's decision process and generate a saliency map as means of explanation. An example saliency map is shown in Figure 7 where red pixels are the most relevant to the decision and the dark blue pixels are the least relevant.

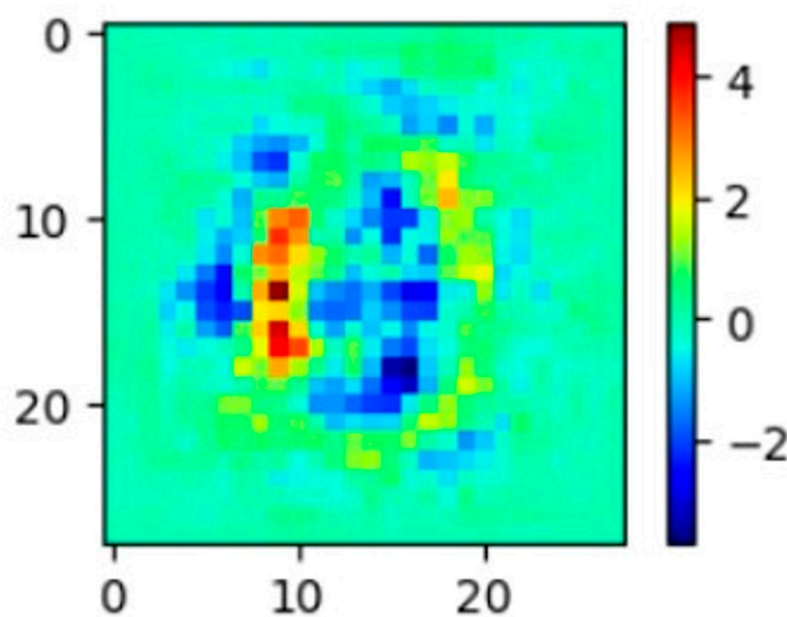


Figure 7. Example saliency map for the MNIST dataset [40]. The red pixels are the most relevant to the decision and the dark blue pixels are the least relevant.

5.5. Visual Backpropagation Case Study

For the purpose of identifying trends in what provenance needs to be saved for each NFR, visual backpropagation will offer another chance to verify an explainability NFR. As with the saliency mapping case study, it was found that visual backpropagation requires simpler specifications than the interpretability methods. Because it seeks only to understand a specific decision, only the model and decision-specific provenance is needed.

As with the saliency mapping case study, visual backpropagation is a feature attribution method, so the provenance specifications will be similar. Visual backpropagation generates a similar visualization to saliency mapping using a different relevance calculation. The relevance scores used for this method involve calculating activation gradients in each layer [60]. As the name implies, this is similar to backpropagation during neural network training, but merely to visualize the activations as opposed to adjusting the model weights.

5.5.1. Requirement Specifications–Visual Backpropagation

[Req. 1] The model's development pipeline shall incorporate prediction explanations for the user.

[Req. 2] The model's development artifacts shall be traceable throughout its lifecycle to ensure explainability.

[Req. 3] *The model's development pipeline shall save development provenance to ensure development artifact traceability to model specifications.*

[Req. 4] The model's development pipeline shall save the model weights to the provenance documentation.

[Req. 5] The model's development pipeline shall save the layer activation gradients to the provenance documentation.

5.5.2. Requirement Specification Verification–Visual Backpropagation

This case study uses the same Kaggle example as the saliency mapping case study with a model trained on the MNIST dataset [40]. The requirements' specifications necessitate saving the model weights and the layer activation gradients in order to create a relevancy map, such as the one shown in Figure 8. The light pixels in the relevancy map indicate regions of high activation.

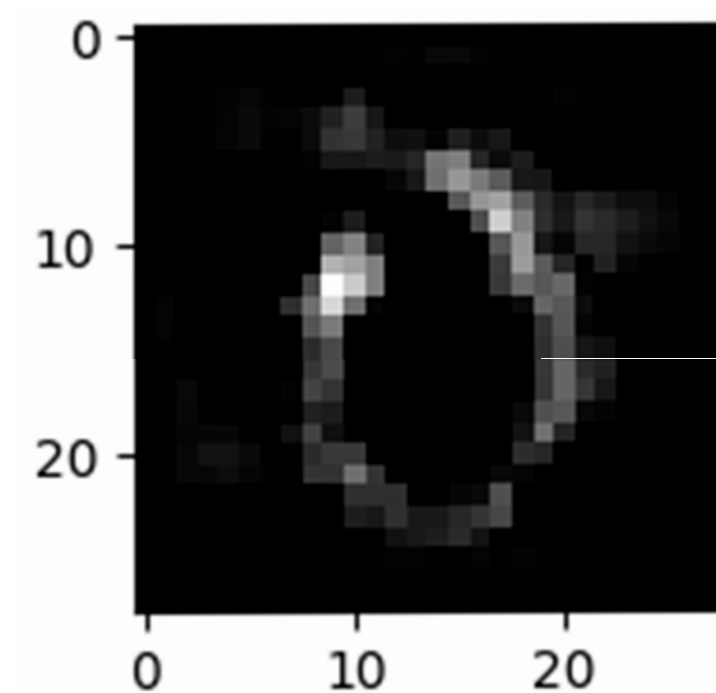


Figure 8. Example relevancy map formed using visual backpropagation [40]. The lighter the pixel, the more relevant it is to the decision.

6. Discussion

These case studies utilize a method from each category in Tables 1 and 2, with the exception of surrogate models because they are also inherently interpretable models, just used in a different way. Each case study is meant to represent how provenance decomposes an interpretability or explainability requirement for any method within their category. Therefore, the decomposition for a decision tree would be similar to that of linear regression because the goal of both is to replicate and simulate the model's behavior. Similarly, the goal of any feature attribution method is to understand how changes in the model's inputs affect the model's predictions, so any method within that category would follow a similar decomposition to CE.

These case studies are proofs of concept; developers can save additional provenance based on their needs. Figure 5 is also an example visualization that developers and users may find useful in understanding their model's behavior but is by no means the only way to verify interpretability using the saved provenance. Developers may choose to adjust the visualization as needed to fit their needs.

The case studies work with yProv4ML, which saves most of the data as needed. However, as Figure 4 shows, the preprocessing steps had to be saved as model parameters, which is incorrect. Therefore, this research area would benefit from a new provenance tool that could handle more data types, e.g., preprocessing steps.

6.1. Provenance Trends

Through the verification of three interpretability NFRs and two explainability NFRs, it is possible to find trends in what provenance data is necessary to verify a generalized version of interpretability and explainability requirements.

Based on the examples presented in this work, interpretability NFRs most notably require provenance regarding the processing of the training data. This includes the training data itself, along with preprocessing steps and any necessary feature selection. This makes sense when considering that interpretability entails understanding data trends and how the model represents them.

In contrast, explainability NFRs, which are concerned with the model's mechanics for a single data point, need provenance regarding the final model along with its decision process for that specific data instance. Such provenance includes the model weights and the relevancy scores relevant to the desired visualization method.

While requirement engineers are free to store any provenance data that their application deems necessary, these trends may offer guidance for how to decompose and verify interpretability and explainability NFRs using provenance. Because of the vast amount of provenance data generated, it is recommended to only save the provenance necessary for NFR verification to avoid system slowdown.

6.2. Method Scalability

The case studies presented in this work offer simple, but effective, examples of how provenance can be used to verify interpretability and explainability requirements. However, most problems in the real world are more complex than the ones presented here. As such, the precise type and amount of saved provenance data may vary for real world examples. This brings method scalability into consideration.

As with many other NFRs, provenance has some trade-offs, including one with scalability. As problems become more complex, the amount of provenance data increases, making it more difficult to store and analyze the data. Although an in-depth analysis of this trade-off is outside of the scope of this work, future work will study the point of diminishing return between provenance and scalability. A brief analysis of the scalability of this approach can be done by showing the runtime for graph generation, which is shown in Table 4.

Table 4. Runtime for graph generation over each case study.

Case Study	Graph Generation Time
Linear Regression	11 min
Counterfactual Explanations	14 min
Functional Decomposition	11 min
Saliency Mapping	15 min
Visual Backpropagation	15 min

7. Limitations and Future Work

This work successfully verifies three interpretability NFRs and two explainability NFRs. However, this work does not include a complete survey of interpretability or explainability methods. Future work could expand to encompass additional methods to

ensure that this approach verifies all of them. A more in-depth survey and analysis will also provide additional evidence for the trends discussed in this work.

Additionally, it was noted in the approach section that none of the available ML development and provenance tracking tools were capable of correctly tracking all of the necessary provenance for this work, e.g., data preprocessing steps. Therefore, future work will include the development of a more complete ML provenance tracking tool that is compatible with other development platforms.

As was mentioned in the discussion, the trade-off between scalability and provenance poses a limitation to this method. Because real world problems are often more complex than the problems discussed in this work, the amount of provenance data necessary to verify requirements may increase dramatically, causing storage and computational concerns. Future work will analyze this limitation and evaluate its threat to the validity of this work.

8. Conclusions

This work presents a requirements-based method for measuring and verifying interpretability and explainability non-functional requirements (NFRs) in machine learning (ML) systems. The approach decomposes these NFRs into provenance requirements and defines corresponding functional requirements (FRs) that specify which model and data provenance must be captured. The resulting provenance enables developers to simulate or visualize model behavior in human-understandable terms, providing a concrete mechanism for verification.

The approach was validated through three interpretability case studies and two explainability case studies. Provenance was used to verify interpretability requirements for linear regression, counterfactual explanations, and functional decomposition, as well as explainability requirements for saliency mapping and visual backpropagation. In the linear regression case, saved provenance enables direct simulation of model behavior, while in the remaining case studies, provenance supports visualizations that expose internal decision-making processes. Collectively, these results demonstrate that provenance can operationalize interpretability and explainability as verifiable properties rather than abstract qualities.

Future work includes extending the approach to additional interpretability and explainability techniques and developing provenance tracking tools capable of capturing a broader range of ML-specific provenance artifacts. This would further strengthen the applicability of requirements-driven verification for complex ML systems.

Author Contributions: Conceptualization, L.V.; methodology, L.V.; software, L.V.; validation, L.V., J.C., T.T.P. and O.O.; formal analysis, L.V.; investigation, L.V., J.C., T.T.P., E.L., D.C. and O.O.; resources, L.V. and D.C.; data curation, L.V. and E.L.; writing—original draft preparation, L.V., J.C., T.T.P., E.L. and D.C.; writing—review and editing, L.V., T.T.P. and O.O.; visualization, L.V.; supervision, O.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original data presented in the study are openly available at: <https://github.com/lynndalou/ProvInterpretExplain>, accessed on 13 December 2025. All code and provenance graphs.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ML	Machine Learning
NFR	Non-Functional Requirement
FR	Functional Requirement
RE	Requirements Engineering
SDLC	Software Development Life Cycle
V&V	Validation and Verification
LIME	Local Interpretable Model-agnostic Explanations
SHAP	SHapley Additive exPlanations
DVC	Data Version Control
W&B	Weights and Biases
W3C	World Wide Web Consortium
PROV-O	PROVenance Ontology
OWL	Web Ontology Language
PROV-DM	PROVenance Data Model
PROV-JSON	PROVenance JavaScript Object Notation
CE	Counterfactual Explanations
DiCE	Diverse Counterfactual Explanations
SMA	Simple Moving Average
LRP	Layer-wise Relevance Propagation
MNIST	Modified National Institute of Standards and Technology

References

1. Carvalho, D.V.; Pereira, E.M.; Cardoso, J.S. Machine Learning Interpretability: A Survey on Methods and Metrics. *Electronics* **2019**, *8*, 832. [\[CrossRef\]](#)
2. Blanco-Justicia, A.; Domingo-Ferrer, J. Machine Learning Explainability Through Comprehensible Decision Trees. In Proceedings of the International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE 2019), Canterbury, UK, 26–29 August 2019.
3. Habibullah, K.M.; Gay, G.; Horkoff, J. Non-functional requirements for machine learning: An exploration of system scope and interest. In Proceedings of the SE4RAI '22: Proceedings of the 1st Workshop on Software Engineering for Responsible AI, Pittsburg, PA, USA, 19 May 2022.
4. Horkoff, J. Non-Functional Requirements for Machine Learning: Challenges and New Directions. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju, Republic of Korea, 23–27 September 2019.
5. Vonderhaar, L.; Elvira, T.; Ochoa, O. Provenance: A Machine Learning Non-Functional Requirement. In Proceedings of the IEEE Artificial Intelligence x Software Engineering (AIxSE), Laguna Hills, CA, USA, 1–3 October 2025.
6. Kale, A.; Nguyen, T.; Harris, F.C., Jr.; Li, C.; Zhang, J.; Ma, X. Provenance Documentation to Enable Explainable and Trustworthy AI: A Literature Review. *Data Intell.* **2023**, *5*, 139–162. [\[CrossRef\]](#)
7. Scherzinger, S.; Seifert, C.; Wiese, L. The Best of Both Worlds: Challenges in Linking Provenance and Explainability in Distributed Machine Learning. In Proceedings of the IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–9 July 2019.
8. Nakagawa, T.; Narita, K.; Kim, K.-S. How Provenance helps Quality Assurance Activities in AI/ML Systems. In Proceedings of the AIMLSystems '22: Proceedings of the Second International Conference on AI-ML Systems, Bangalore, India, 12–15 October 2022.
9. Schlegel, M.; Sattler, K.-U. Management of Machine Learning Lifecycle Artifacts: A Survey. *ACM SIGMOD Rec.* **2023**, *51*, 18–35. [\[CrossRef\]](#)
10. Davies, J.; German, D.M.; Godfrey, M.W.; Hindle, A. Software Bertillonage: Determining the provenance of software development artifacts. *Empir. Softw. Eng.* **2013**, *18*, 1195–1237. [\[CrossRef\]](#)
11. Davies, J.; German, D.M.; Godfrey, M.W.; Hindle, A. Software bertillonage: Finding the provenance of an entity. In Proceedings of the MSR '11: Proceedings of the 8th Working Conference on Mining Software Repositories, Waikiki, Honolulu, HI, USA, 21–22 May 2011.
12. Bose, R.P.C.; Phokela, K.K.; Kaulgud, V.; Podder, S. BLINKER: A Blockchain-Enabled Framework for Software Provenance. In Proceedings of the 2019 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2–5 December 2019.
13. Godfrey, M.W. Understanding software artifact provenance. *Sci. Comput. Program.* **2015**, *97*, 86–90. [\[CrossRef\]](#)

14. Pham, Q.; Malik, T.; Foster, I. Auditing and Maintaining Provenance in Software Packages. In Proceedings of the Provenance and Annotation of Data and Processes (IPAW), Cologne, Germany, 9–13 June 2014.
15. Chittimalli, P.K.; Naik, R. Variable provenance in software systems. In Proceedings of the RSSE 2014: Proceedings of the 4th International Workshop on Recommendation Systems for Software Engineering, Hyderabad, India, 3 June 2014.
16. Wendel, H.; Kunde, M.; Schreiber, A. Provenance of Software Development Processes. In Proceedings of the Provenance and Annotation of Data Processes (IPAW), Troy, NY, USA, 15–16 June 2010.
17. Costa, G.C.B.; Dalpra, H.L.O.; Teixeira, E.N.; Werner, C.M.L.; Braga, R.M.M.; Miguel, M.A. Software Processes Analysis with Provenance. In Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES), Wolfsburg, Germany, 28–30 November 2018.
18. Sáenz-Adán, C.; Moreau, L.; Pérez, B.; Miles, S.; García-Izquierdo, F.J. Automating Provenance Capture in Software Engineering with UML2PROV. In Proceedings of the Provenance and Annotation of Data and Processes (IPAW), London, UK, 9–10 June 2018.
19. Schreiber, A.; von Kurnatowski, L.; Meinecke, A.; de Boer, C. Visualization of Software Development Provenance. In Proceedings of the International Conference on Human-Computer Interaction, Washington DC, USA, 29 June–4 July 2024.
20. Vallayil, M.; Nand, P.; Yan, W.Q.; Allende-Cid, H. Explainability of Automated Fact Verification Systems: A Comprehensive Review. *Appl. Sci.* **2023**, *13*, 12608. [\[CrossRef\]](#)
21. Slack, D.; Hilgard, S.; Jia, E.; Singh, S.; Lakkaraju, H. Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods. In Proceedings of the AIES '20: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, New York, NY, USA, 7–8 February 2020.
22. Mahmud, S.; Saisubramanian, S.; Zilberstein, S. Verification and Validation of AI Systems Using Explanations. In Proceedings of the 2024 AAAI Fall Symposia, Arlington, VA, USA, 7–9 November 2024.
23. Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; Pedreschi, D. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* **2018**, *51*, 1–42. [\[CrossRef\]](#)
24. Doshi-Velez, F.; Kim, B. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv* **2017**, arXiv:1702.08608. [\[CrossRef\]](#)
25. Nuseibeh, B.; Easterbrook, S. Requirements engineering: A roadmap. In Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 4–11 June 2000.
26. Macaulay, L.A. *Requirements Engineering*; Springer Science & Business Media: Cham, Switzerland, 2012.
27. Villamizar, H.; Escovedo, T.; Kalinowski, M. Requirements engineering for machine learning: A systematic mapping study. In Proceedings of the 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy, 1–3 September 2021.
28. Rahimi, M.; Guo, J.L.; Kokaly, S.; Chechik, M. Toward requirements specification for machine-learned components. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), Jeju, Republic of Korea, 23–27 September 2019.
29. Pei, Z.; Liu, L.; Wang, C.; Wang, J. Requirements engineering for machine learning: A review and reflection. In Proceedings of the 2022 IEEE 30th International Requirements Engineering Conference Workshops (REW), Melbourne, Australia, 15–19 August 2022.
30. Kaminwar, S.R.; Goschenhofer, J.; Thomas, J.; Thon, I.; Bischl, B. Structured verification of machine learning models in industrial settings. *Big Data* **2023**, *11*, 181–198. [\[CrossRef\]](#) [\[PubMed\]](#)
31. Chazette, L.; Schneider, K. Explainability as a non-functional requirement: Challenges and recommendations. *Requir. Eng.* **2020**, *25*, 493–514. [\[CrossRef\]](#)
32. Sovrano, F.; Vitali, F. An objective metric for explainable AI: How and why to estimate the degree of explainability. *Knowl.-Based Syst.* **2023**, *278*, 110866. [\[CrossRef\]](#)
33. Namatēvs, I.; Sudars, K.; Dobrājs, A. Interpretability versus Explainability: Classification for Understanding Deep Learning Systems and Models. *Eng. Optim.* **2022**, *29*, 297–356.
34. Elshawi, R.; Sherif, Y.; Al-Mallah, M.; Sakr, S. Interpretability in Healthcare: A Comparative Study of Local Machine Learning Interpretability Techniques. *Comput. Intell.* **2021**, *37*, 1633–1650. [\[CrossRef\]](#)
35. Murdoch, W.J.; Singh, C.; Kumbier, K.; Abbasi-Asl, R.; Yu, B. Interpretable Machine Learning: Definitions, Methods, and Applications. *arXiv* **2019**, arXiv:1901.04592. [\[CrossRef\]](#) [\[PubMed\]](#)
36. Ribeiro, M.T.; Singh, S.; Guestrin, C. Model-Agnostic Interpretability of Machine Learning. *arXiv* **2016**, arXiv:1606.05386. [\[CrossRef\]](#)
37. Marcinkevičs, R.; Vogt, J.E. Interpretable and explainable machine learning: A methods-centric overview with concrete examples. *WIREs Data Min. Knowl. Discov.* **2023**, *13*, e1493. [\[CrossRef\]](#)
38. Gilpin, L.H.; Bau, D.; Yuan, B.Z.; Bajwa, A.; Specter, M.; Kagal, L. Explaining Explanations: An Overview of Interpretability of Machine Learning. In Proceedings of the 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA), Turin, Italy, 1–4 October 2018.
39. Vonderhaar, L.; Elvira, T.; Ochoa, O. An analysis of explainability methods for convolutional neural networks. *Eng. Appl. Artif. Intell.* **2023**, *117*, 105606.

40. Chang, E. MNIST with Keras (Visualization and Saliency Map). Kaggle. 2017. Available online: <https://www.kaggle.com/code/ernie55ernie/mnist-with-keras-visualization-and-saliency-map> (accessed on 13 December 2025).
41. Samuel, S.; Löffler, F.; König-Ries, B. Machine learning pipelines: Provenance, reproducibility and FAIR data principles. In Proceedings of the International Provenance and Annotation Workshop, Virtual, 22 June 2020.
42. Werder, K.; Ramesh, B.; Zhang, R. Establishing data provenance for responsible artificial intelligence systems. *ACM Trans. Manag. Inf. Syst.* **2022**, *13*, 1–23. [CrossRef]
43. Vonderhaar, L.; Elvira, T.; Ochoa, O. Generating and Verifying Synthetic Datasets with Requirements Engineering. In Proceedings of the 4th International Conference on AI Engineering—Software Engineering for AI (CAIN), Ottawa, ON, Canada, 27–28 April 2025.
44. MLflow. Deliver Production-Ready AI. 2025. Available online: <https://mlflow.org/> (accessed on 13 December 2025).
45. Data Version Control (DVC). 2025. Available online: <https://dvc.org/> (accessed on 13 December 2025).
46. Weights & Biases. CoreWeave. 2025. Available online: <https://wandb.ai/site/> (accessed on 13 December 2025).
47. Neptune.AI. Neptune.AI, 2025. Available online: <https://neptune.ai/> (accessed on 13 December 2025).
48. Where AI Developers Build. Comet. 2025. Available online: <https://www.comet.com/site/> (accessed on 13 December 2025).
49. Padovani, G.; Anantharaj, V.; Fiore, S. yProv4ML: Effortless provenance tracking for machine learning systems. *SoftwareX* **2025**, *31*, 102298. [CrossRef]
50. World Wide Web Consortium (W3C). PROV-O: The PROV Ontology. 2013. Available online: <https://www.w3.org/TR/prov-o/> (accessed on 13 December 2025).
51. PROV-DM: The PROV Data Model. World Wide Web Consortium. 2013. Available online: <https://www.w3.org/TR/2013/REC-prov-dm-20130430/> (accessed on 13 December 2025).
52. Arp, R.; Smith, B.; Spear, A.D. *Building Ontologies with Basic Formal Ontology*; MIT Press: Cambridge, MA, USA, 2015.
53. Elmenshawii, F. Linear Regression from Scratch. Kaggle. 2023. Available online: <https://www.kaggle.com/code/fareselmenshawii/linear-regression-from-scratch/notebook> (accessed on 13 December 2025).
54. Use the Analysis ToolPak to Perform Complex Data Analysis. Microsoft. 2025. Available online: <https://support.microsoft.com/en-us/office/use-the-analysis-toolpak-to-perform-complex-data-analysis-6c67ccf0-f4a9-487c-8dec-bdb5a2cefab6> (accessed on 13 December 2025).
55. Mothilal, R.K.; Sharma, A.; Tan, C. Diverse Counterfactual Explanations (DiCE) for ML. InterpretML. 2020. Available online: <https://interpret.ml/DiCE/> (accessed on 13 December 2025).
56. Sharma, A. Quick Introduction to Generating Counterfactual Explanations Using DiCE. GitHub. 2022. Available online: https://github.com/interpretml/DiCE/blob/main/docs/source/notebooks/DiCE_getting_started.ipynb (accessed on 13 December 2025).
57. Molnar, C. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*; Self-Published: Munich, Germany, 2025.
58. Aghili, S.A.S. Time Series Decomposition Methods. Kaggle. 2023. Available online: <https://www.kaggle.com/code/alisadeghiaghili/time-series-decomposition-methods> (accessed on 13 December 2025).
59. Alqaraawi, A.; Schuessler, M.; Weiß, P.; Costanza, E.; Berthouze, N. Evaluating saliency map explanations for convolutional neural networks: A user study. In Proceedings of the IUI '20: Proceedings of the 25th International Conference on Intelligent User Interfaces, Cagliari, Italy, 17–20 March 2020.
60. Bhat, A.; Sangbone Assoa, A.; Raychowdhury, A. Gradient Backpropagation based Feature Attribution to Enable Explainable-AI on the Edge. In Proceedings of the 2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC), Patras, Greece, 3–5 October 2022.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.