Documenting AI Systems under the EU AI Act: A UML Framework for Post-Hoc XAI Compliance

José Uetanabara Júnior

Independent Researcher, miklotovx@gmail.com

Artificial Intelligence (AI) has gained prominence in recent years, with widespread adoption in academic and industrial contexts raising challenges related to the auditability of AI-based systems. Explainable Artificial Intelligence (XAI) addresses this issue through post-hoc methods that provide insight into model decisions. However, the integration of XAI mechanisms into software engineering artifacts and architectural representations remains limited. At the same time, the European Union's AI Act (EU AI Act, Regulation 2024/1689) demands extensive technical requirements for high-risk AI systems, which in practice often lead to large, fragmented, and costly compliance efforts that are difficult to maintain, verify, and trace back to concrete system implementations. To address this gap, this work proposes a UML-based framework for documenting post-hoc XAI systems aligned with EU AI Act requirements. The framework introduces a minimal set of Unified Modeling Language (UML) stereotypes, tagged values, and relationships to represent data sources, training orchestration, trained models, and associated explainability mechanisms, relying on architectural information directly derivable from object-oriented (OO) source code. As an additional contribution, this work introduces the UMLOOModeler, a tool that automates the generation of UML class diagrams from OO Python implementations, ensuring consistency between code-level artifacts and architectural representations. The framework is illustrated through examples involving heterogeneous data modalities, demonstrating support for architectural traceability and auditability across different XAI pipelines.

Disclaimer: *This work is shared for early dissemination and academic discussion and has not yet undergone peer review, since it is an ongoing research effort.*

1 INTRODUCTION

The EU AI Act represents the first comprehensive regulatory framework specifically designed to regulate the development, deployment, and use of AI systems within the European Union. One of its central pillars is the obligation imposed on providers of high-risk AI systems to produce extensive technical documentation, enabling regulatory authorities, auditors, and authorized users to assess compliance, reliability, and accountability [1].

The regulation does not prescribe a standardized format for this material, allowing organizations to define how it is structured and presented, provided that all mandatory elements specified in the annexes are fully addressed. While assessment instruments such as checklists may support high-level conformity evaluations, they do not provide a systematic, system-level architectural representation suitable for engineering traceability or auditability [2, 3].

As a consequence, the cost of compliance with the EU AI Act has become a major concern. Producing, maintaining, and auditing hundreds of pages of technical material is a complex, time-consuming, and error-prone process. This difficulty is further amplified in systems whose implementation lacks explicit architectural structure or relies heavily on procedural code, as compliance-relevant information must often be reconstructed manually from heterogeneous sources. In such contexts, the absence of architectural models

increases the human effort required to establish traceability between system components, data flows, and decision-making mechanisms [3, 4].

Among the mandatory elements required by the EU AI Act, XAI is explicitly recognized as a key requirement for high-risk AI systems, as it enables the interpretation and justification of automated decisions. Although a wide range of post-hoc XAI techniques has been extensively studied in the literature, their incorporation into compliance-oriented artifacts remains largely unstructured. As a result, explainability is often treated as an isolated analytical activity rather than as an integral, traceable component of system engineering. From a compliance perspective, this separation hinders the integration of explainability-related artifacts into the required technical material [1, 5, 6].

From a software engineering perspective, system specification and analysis are commonly supported by standardized modeling languages, with the UML being one of the most widely adopted. However, standard UML does not natively support the representation of XAI concepts, nor does it explicitly address the requirements introduced by the EU AI Act [7, 8]. To the best of the author's knowledge, no existing UML-based framework integrates post-hoc XAI mechanisms into the software development lifecycle in a model-agnostic, data-agnostic, and explainer-agnostic manner while simultaneously supporting compliance with the EU AI Act.

To address this gap, this work introduces a UML-based framework that connects post-hoc XAI mechanisms to the obligations imposed by the EU AI Act, focusing on systems developed following OO design principles, for which explicit architectural elements can be identified and modeled. The framework defines a set of UML extensions, including stereotypes, tagged values, and explicitly defined relationships, to support the systematic representation of explainability mechanisms. While the proposal is motivated by the requirements applicable to high-risk AI systems, it is not restricted to a specific application domain and is expected to be applicable to a wide range of AI-based systems. In addition, an initial supporting tool is introduced to assist in generating such UML artifacts. Rather than replacing existing compliance processes, the proposed framework and tool are intended to complement mandatory textual documentation by improving consistency, traceability, and transparency across explainability-related elements.

2 KEY CONCEPTS

In order to understand this work, some fundamental concepts must be addressed. These concepts will be presented in this chapter.

2.1 Explainable Artificial Intelligence (XAI)

Machine learning models have become complex. As examples, we can mention Deep Neural Networks, Convolutional Neural Networks (CNN), Recurrent Neural Network (RNN), Transformers, Ensembles, among others. These systems often operate as black boxes, which makes it difficult to understand the process that leads to a given decision. Therefore, when an explanation is required, an XAI approach is used to interpret the learning model and provide the appropriate explanation [9, 10].

2.2 Local and Model-Agnostic Explanation Methods

Local model-agnostic interpretability methods share a central characteristic: they are capable of generating explanations without accessing or modifying the internal structure of the model. Examples include techniques such as LIME and kernel-based variants of SHAP. This independence makes such approaches widely applicable to different algorithms and deployment contexts [9, 10]. In this work, the following methods were considered:

- Local Interpretable Model-Agnostic Explanations (LIME) generates locally interpretable surrogate models around a specific instance to approximate the behavior of the model within a small neighborhood [9].

- Shapley Additive Explanations (SHAP) applies cooperative game theory to assign each feature a shapley value, representing its average marginal contribution to the model's prediction [11].

2.3 EU AI ACT Overview

The European Union's AI Act establishes the first comprehensive regulatory framework for artificial intelligence systems. The regulation classifies AI applications according to risk levels and imposes specific obligations for high-risk systems, such as AI-assisted medical diagnostics [1].

In the context of this work, particular attention is given to Annex IV of the EU AI Act, which defines the mandatory elements of technical documentation for high-risk AI systems, while Annex I is considered as a contextual reference for system classification. It must enable regulatory authorities, auditors, and authorized users to understand how the system operates, assess its reliability, and verify its compliance with legal requirements [1, 12].

In summary, the key elements to be documented are:

- Purpose and intended use: description of the system's objective and application context.
- Technical architecture: overview of the system's structure and its main components.
- Training, validation, and testing data: description of the datasets used, their origin, and quality.
- Models and parameters: identification of the models employed, their versions, and relevant parameters.
- Logging and traceability: mechanisms for recording and auditing decisions.
- Metrics and testing: performance indicators, robustness, and fairness.
- Explainability: description of the methods used to interpret the model's decisions.
- Risk management: identification of associated risks and mitigation strategies.
- Governance and quality: internal procedures, roles, and organizational responsibilities.

3 A UML FRAMEWORK FOR POST-HOC XAI COMPLIANCE UNDER THE EU AI ACT

As discussed in the previous sections, the documentation requirements imposed by the EU AI Act pose significant challenges for software engineers. To address these challenges, this section introduces a UML-based framework designed to systematically represent post-hoc XAI mechanisms as part of the system's technical documentation [1]. To illustrate its applicability, the framework is instantiated using an illustrative breast cancer diagnosis system, which serves as a reference implementation for demonstrating how heterogeneous data sources, models, and post-hoc explanation methods can be documented within a UML representation.

3.1 Illustrative Breast Cancer Diagnosis System

Figure 1 shows an illustrative breast cancer diagnosis system used as a reference implementation to instantiate the proposed framework. The figure represents a modular neural network at a high level of abstraction, serving as a concrete basis for demonstrating how post-hoc explainability mechanisms can be documented using UML. The `DataEntry` interface represents the data sources consumed by three modules, modeled as packages: `ClinicalModule`, `ImageModule`, and `GeneticModule`.

The `ClinicalModule` represents a component designed to process tabular clinical data, the `ImageModule` corresponds to an image-based classification component, and the `GeneticModule` is designed to process sequential genetic data. Figure 1 provides a high-level contextual view of these modules and their shared data entry point, represented by the `DataEntry` interface. It is important to note that the interface-package connections shown in Figure 1 are illustrative and included solely to provide contextual clarity, rather than to represent execution flow or system integration logic.

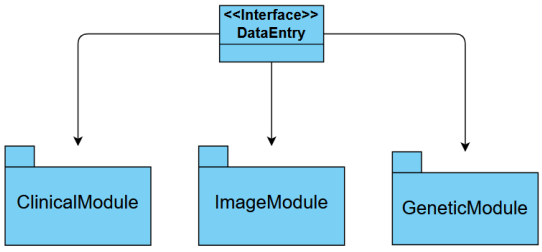

Figure 1: High-level representation of the illustrative system.

Of the three modules presented, the `ClinicalModule` was implemented using the Wisconsin Breast Cancer dataset available in the scikit-learn library [13]. The `ImageModule` was implemented using the Breast Cancer Histopathology image dataset [14], while the `GeneticModule` was implemented using genomic expression from The Cancer Genome Atlas [15].

It is important to emphasize that the illustrative system presented in this section does not represent a reference architecture or a recommended system design. Instead, it constitutes a concrete instantiation of the proposed framework, selected exclusively to demonstrate how heterogeneous data modalities, learning models, and post-hoc explanation methods can be documented within a UML representation.

3.2 Detailed UML Modeling

Unlike approaches that rely on conceptual metamodels, this work starts from the concrete structure of OO source code implemented in Python within the Jupyter Notebook environment [16]. The implementation strategy adopted in this work emerged from the incremental organization of the code during development, in which responsibilities such as data access, model definition, training, and explainability were progressively isolated into explicit software constructs. This organization made explicit a minimal architectural structure required for the framework to operate as intended, namely the clear separation between data sources, model definition components, trained models, and post-hoc explainability mechanisms. These architectural roles could then be directly mapped to UML elements and stereotypes. Multiple implementations were developed following this architectural structure, each corresponding to a distinct local model-agnostic explanation method. As a result, traceability across explanatory processes is preserved, and the resulting documentation remains closely aligned with the actual implementation, thereby supporting system auditability [17]. The complete source code used in the illustrative examples is publicly available at: `github.com/miklotovx/UMLOOModeler`.

Figure 2 presents the UML modeling derived from the illustrative system used to instantiate the proposed framework. Rather than focusing on a single module, the diagram includes selected elements from each module in order to demonstrate how post-hoc explainability mechanisms can be consistently represented across heterogeneous data modalities. The diagram was manually constructed based on the class diagrams generated by the UMLOOModeler tool, with modeling decisions explicitly guided by the corresponding source code.

The attributes, methods, and parameters shown in the diagram accurately reflect the elements manipulated in the implementation. Non-essential elements were intentionally omitted to improve readability. The UML class diagram was selected because it enables the direct representation of models, explainers, data sources, and their relationships. All stereotypes used in the diagram are extensions of the UML Class metaclass and are summarized in Table 1.

The `ClinicalDatabase`, `ImageDatabase`, and `GeneticDatabase` classes represent the data sources used by the illustrative system. The `<<DataSource>>` stereotype was used to indicate data input elements. In addition, a tagged value `Name` was introduced to explicitly identify each dataset. For example, the dataset associated with the `ClinicalDatabase` originates from the Wisconsin Breast Cancer dataset.

Table 1: Stereotype definition

Stereotype	Extended metaclass	Description
`<<DataSource>>`	Class	Source of data.
`<<ModelDefinition>>`	Class	Training orchestrator.
`<<TrainedModel>>`	Class	Trained model.
`<<Perturbator>>`	Class	Generates perturbations.
`<<LimeExplainer>>`	Class	LIME explanation.
`<<ShapExplainer>>`	Class	SHAP Explanation.
`<<Classifier>>`	Class	Classification Model.

The `ModelA_MLP` and `ModelB_RF` classes represent model definition components responsible for instantiating and training a learning algorithm using the tabular data available in the `ClinicalDatabase`. The `MLP` and `RF` identifiers were deliberately included in the class names to indicate the learning algorithms used during training were the MLPClassifier and the Random Forest Classifier, respectively. For these classes, the `<<ModelDefinition>>` stereotype was applied to indicate their role as trainable model.

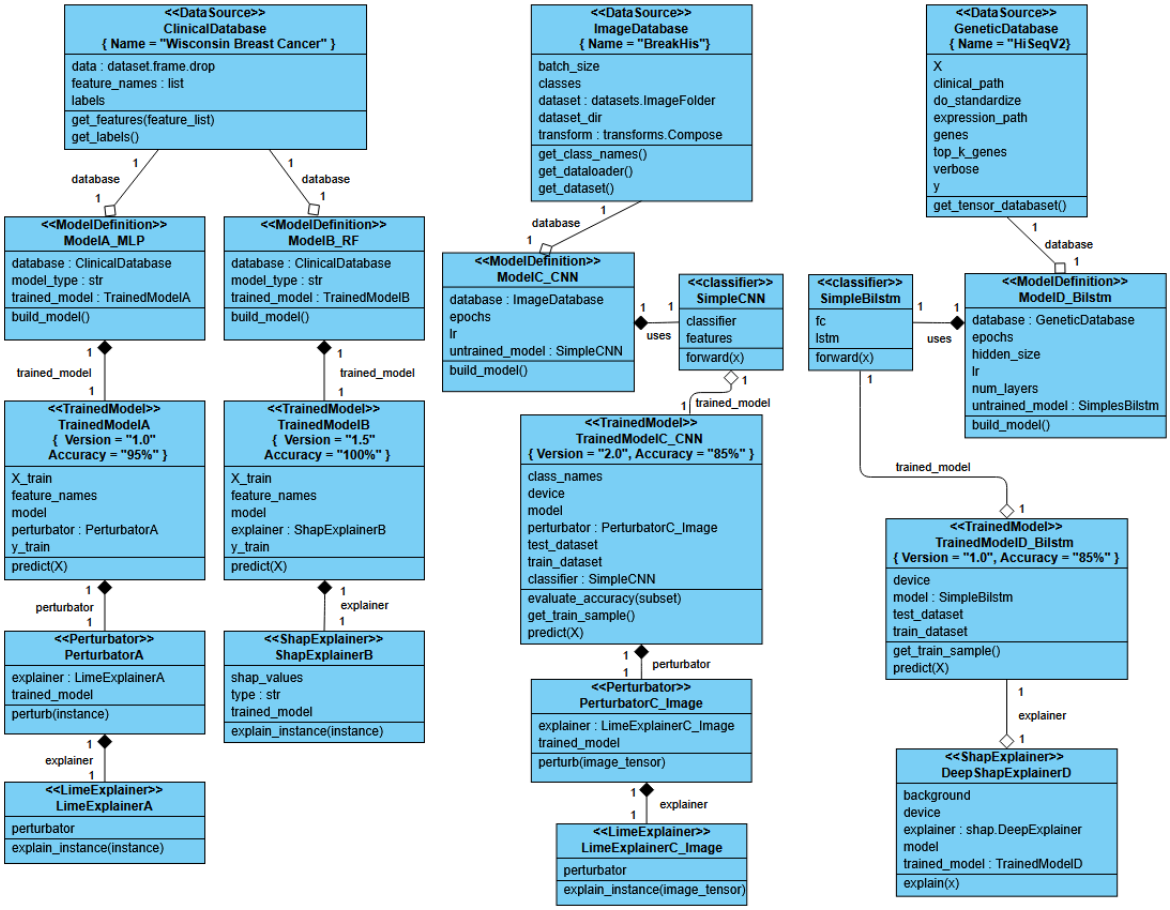

Figure 2: Detailed UML modeling of the illustrative system.

The TrainedModelA and TrainedModelB classes represent the trained MLP model and the trained RF model, respectively. These classes store the trained classifier, the training data, and the names of the input variables. Two tagged values, Version and Accuracy, were added to explicitly record the trained model's version and its performance. The <<TrainedModel>> stereotype was used to indicate that these classes correspond to trained models.

Each trained model is explained by a distinct method. The TrainedModelA class is explained using the LIME technique, which employs a perturbation-based strategy. For this purpose, it is associated with the PerturbatorA class, which generates data variations for local analysis. The PerturbatorA class receives a reference to the trained model and instantiates the LimeExplainerA class to generate the final explanation. The <<Perturbator>> and <<LimeExplainer>> stereotypes were applied to these classes, respectively.

The TrainedModelB class maintains a direct association with the ShapExplainerB class. Unlike LIME, this method does not rely on a perturbator component, as explanations are derived directly from the trained model. SHAP explains individual predictions by assigning contribution values to input features based on Shapley values. The <<ShapExplainer>> stereotype was applied to this class to represent the explanations.

The ModelC_CNN class represents a model definition component responsible for instantiating and training a convolutional neural network using image data available in the ImageDatabase. The CNN identifier was deliberately included in the class name to indicate that the learning algorithm employed during training is a CNN. In this case, the training workflow explicitly instantiates a custom classifier architecture, represented by the SimpleCNN class, which is responsible for producing discrete predictions from image data. This modeling choice reflects the fact that the CNN architecture is implemented as a personalized classifier rather than as a predefined model from a machine learning library. For this reason, the SimpleCNN class received the <<Classifier>> stereotype.

The TrainedModelC_CNN class represents the trained CNN model. Its structure and behavior are consistent with the trained models previously described, and it includes the same tagged values used to record versioning and performance information. The <<TrainedModel>> stereotype was applied to indicate that this class corresponds to a trained model.

TrainedModelC_CNN is explained using the LIME technique, following the same explanatory pattern adopted for TrainedModelA. Accordingly, the PerturbatorC_Image and LimeExplainerC_Image classes received the <<Perturbator>> and <<LimeExplainer>> stereotypes, respectively.

The ModelD_Bilstm class represents a model definition component responsible for instantiating and training a RNN using sequential data available in the GeneticDatabase. The Bilstm identifier was deliberately included in the class name to indicate the learning algorithm employed during training. As in the image-based case, the training workflow instantiates a custom classifier architecture, represented by the SimpleBilstm class, which received the <<Classifier>> stereotype.

The TrainedModelD class represents the trained Bilstm model. As with the other trained models, tagged values are used to record versioning and performance

information. The `<<TrainedModel>>` stereotype was applied to indicate that this class corresponds to a trained model.

`TrainedModelD` is explained using the DeepSHAP technique, which extends the SHAP framework to deep learning models by combining backpropagation-based attribution with Shapley value principles. In this case, the explainer operates directly over the trained model through introspection, without relying on an explicit perturbation component, mirroring the explanation pattern adopted for SHAP-based models and contrasting with the perturbation-based LIME workflow.

The choice of UML relationships in the diagram reflects how explainability mechanisms are integrated into the implementation, rather than the explainability technique itself. Depending on the architectural design adopted in the source code, post-hoc explanation methods may be instantiated as internal components, injected as external dependencies, invoked procedurally, or encapsulated within dedicated wrapper classes. These different implementation strategies are represented using appropriate UML relationships, such as composition, aggregation, inheritance, or direct association. This approach ensures that the UML model faithfully mirrors the actual structure of the system, allowing the diagram to capture not only structural relationships, but also the architectural semantics induced by the chosen implementation of explainability workflows.

All relationships in the diagram were explicitly defined with 1-to-1 multiplicity in order to preserve clarity and avoid ambiguity in the representation of post-hoc explainability workflows. A detailed discussion of the rationale behind this modeling choice, including its implications for auditability and risk interpretation, is provided in Chapter 4.

3.3 Application of the UML-based Framework

To demonstrate how the proposed UML-based framework supports post-hoc auditing activities, this section presents an illustrative audit scenario derived from the documentation artifacts generated for the example system. The objective is to show how the UML model can be used as a structured entry point for technical investigation, regulatory inspection, and explainability assessment.

From an audit perspective, the UML class diagram produced using the proposed framework serves as a consolidated representation of the automated decision-making pipeline. As illustrated in Figure 2 and detailed in Section 3.2, the diagram explicitly captures the flow from data sources to trained models and their associated explainability mechanisms. For instance, image data provided by the `ImageDatabase` is processed by a trainable CNN model definition component, instantiated through a personalized classifier, and later explained using the LIME technique after training.

By explicitly modeling datasets, model definition components, trained models, and their associated explainability mechanisms through stereotypes, tagged values, and relationships, the UML artifacts enable the verification of whether the documented system configuration corresponds to the deployed one and whether explainability mechanisms are consistently applied. Each trained model is directly associated with its corresponding post-hoc explainer, either through perturbation-based workflows, as in the case of LIME, or through direct feature attribution methods, as in the case of SHAP, reducing ambiguity in the interpretation of explainability pipelines and supporting systematic regulatory inspection [17].

It is acknowledged that the proposed framework does not cover all aspects required for full compliance with the EU AI Act. Certain elements, such as risk management, governance, and organizational procedures, remain outside the scope of the UML modeling presented in this section. Broader considerations, including the trade-off between abstraction and traceability, together with a systematic mapping of the proposed framework to the

documentation requirements of the EU AI Act, are addressed in the next chapter.

4 DISCUSSION

As previously presented, the proposed UML-based framework aims to complement technical documentation by supporting both the structural description and the auditing of post-hoc XAI systems. While the illustrative instantiations demonstrate its applicability across different data modalities, models, and explainability techniques, several conceptual and methodological aspects need further discussion. This chapter examines the coverage of the proposed framework with respect to the documentation requirements defined in Annex IV of the EU AI Act, its model-agnostic characteristics across data, models, and explainability mechanisms, the trade-offs between abstraction and traceability, the role of tagged values as compliance points, and the practical implications of the framework for post-hoc auditing and audit-by-design [1].

4.1 Framework Coverage of Annex IV Documentation Requirements

The first item to be discussed concerns the coverage of the proposed UML-based framework with respect to documentation requirements. While the framework aims to support structural transparency and traceability for post-hoc XAI systems, it does not address all elements demanded by the EU AI Act. To clarify its scope and limitations, this subsection focuses specifically on the requirements defined in Annex IV, which establishes the mandatory elements for high-risk AI systems [1].

Table 2 summarizes the nine main regulatory elements defined in Annex IV and indicates whether they are addressed by the proposed framework. The use of UML stereotypes and tagged values allows essential elements to be visualized directly in UML diagrams.

Table 2: Coverage of EU AI Act Documentation Requirements by the Proposed UML Framework

Item	EU AI Act Requirement	Coverage
1. Purpose and intended use	Description of the system's objective and application context	Not covered
2. Technical architecture	Overview of system structure and main components	Stereotypes <<DataSource>>, <<ModelDefinition>>, <<TrainedModel>>, <<Perturbator>>, <<LimeExplainer>>, <<ShapExplainer>>, <<Classifier>>
3. Training, validation, and testing data	Datasets used, their origin, and quality	Tagged values {Name} in <<DataSource>>
4. Models and parameters	Model types, versions, and relevant parameters	Tagged values {Name, Version} in <<TrainedModel>>; model type implicit in class name
5. Logging and traceability	Recording and audit of decisions	Structural traceability enabled through UML stereotypes and relationships
6. Metrics and testing	Performance indicators, robustness, and fairness	Partially covered; Tagged value {Accuracy} in <<TrainedModel>>
7. Explainability	Methods, limitations, interpretability	Explicit represented via <<Perturbator>>, <<LimeExplainer>>, and <<ShapExplainer>> stereotypes
8. Risk management	Risk identification and mitigation	Not covered
9. Governance and quality	Internal processes, roles, responsibilities	Not covered

Other regulatory requirements and more detailed items intentionally remain outside the modeling scope of this work. This decision is primarily motivated by concerns related to abstraction and usability: modeling highly granular or organizational aspects at the UML level would increase diagram complexity without providing proportional benefits for developers or auditors. Consequently, such elements are more appropriately addressed through traditional textual material, reinforcing the role of the proposed UML framework as a complementary artifact rather than a replacement for regulatory documentation [2, 4].

4.2 Model-Agnosticity Across Data, Models, and Explainers

The second item that must be discussed concerns the model-agnostic nature of the proposed framework. As previously illustrated, provided that the minimal architectural contract defined by the framework is respected, the approach is not tied to a specific data modality, learning paradigm, or post-hoc explainability technique. Instead, it provides a common architectural representation capable of documenting heterogeneous XAI pipelines while preserving structural traceability.

In this work, model-agnosticity is understood as a architectural property that manifests across three complementary dimensions: data, models, and explainability mechanisms. As demonstrated in the illustrative examples, the same UML-based framework is able to represent systems operating over tabular, image, and sequential data, employing different learning models, including black-box and white-box configurations, and integrating distinct post-hoc explanation techniques such as LIME, SHAP, and DeepSHAP.

Although it is not possible to claim that the framework can be instantiated in all conceivable scenarios, it is expected to provide sufficient generality to represent diverse XAI system configurations. The proposed framework does not aim for exhaustive coverage of all possible configurations, but rather to demonstrate that heterogeneous XAI systems can be consistently documented without compromising clarity, usability, or auditability.

4.3 Abstraction and Traceability

The third item that must be discussed concerns abstraction and traceability. The EU AI Act makes it explicit that technical documentation must support traceability, requiring each model version, training dataset, and performance metric to be individually identifiable [1]. For this reason, in the illustrative system instantiated using the proposed framework, architectural abstraction was deliberately avoided in order to show how this regulatory requirement can be satisfied in practice. Beyond traceability, abstraction also raises concerns regarding the integrity of explanations. Post-hoc explainability depends on the consistency between the model, the data, and the local neighborhood considered during explanation generation. Introducing generic abstractions may compromise this consistency when differences exist in data sources, preprocessing steps, or internal model behavior. Consequently, all relationships in the UML model were defined with explicit 1-to-1 multiplicities to preserve full traceability between elements.

The primary drawback of minimizing abstraction is the resulting repetition of code and architectural elements. In the instantiated example, this repetition remains manageable due to the limited scale of the system. However, in larger AI systems comprising

dozens or hundreds of models, both the source code and the corresponding UML diagrams may become highly redundant and difficult to maintain. This issue is not merely technical, but reflects a regulatory constraint imposed by the EU AI Act, which prioritizes explicit traceability over architectural conciseness. Attempts to reduce redundancy through abstraction may conflict with auditability requirements and could be challenged by regulatory authorities [1, 17]. For this reason, continued dialogue between developers and regulators is essential to establish acceptable modeling practices. In the long term, compliance-oriented frameworks that balance regulatory rigor with software engineering principles would benefit both stakeholders.

4.4 Tagged Values as Compliance Points

The fourth item that must be discussed concerns the use of tagged values. In the illustrative system, they are employed as a native UML extension mechanism to represent selected metadata associated with data sources and trained models. Rather than relying on a fixed or predefined vocabulary, the proposed framework does not impose additional restrictions on the use of tagged values beyond those provided by UML. This allows software engineers to define name-value pairs as needed to document selected compliance-related metadata directly in the UML diagram, while preserving architectural traceability and alignment with regulatory expectations.

At the same time, excessive use of tags may overload diagrams and reduce their readability. For this reason, they should be applied selectively, focusing on information that is directly relevant to compliance and auditing. All remaining details should be documented in the textual artifacts, following the same rationale adopted for stereotypes.

4.5 Practical Implications for Post-Hoc Auditing and System Documentation

The fifth item concerns the framework itself. The practical implications of this work should be understood primarily in qualitative terms. Given that the EU AI Act allows technical documentation to be prepared in flexible formats, it is not possible to objectively assess whether the framework reduces overall effort or volume, as the final outcome depends on organizational practices and reporting choices [1].

Instead, the framework contributes by supporting a more structured and consistent representation of explainability-related elements in compliance-oriented artifacts. From a post-hoc auditing perspective, the UML diagrams provide a structured entry point for technical inspection, enabling auditors to identify models, data sources, and associated explainability mechanisms through explicit architectural relationships, without requiring direct access to the implementation [17].

From a software engineering perspective, the framework supports the explicit representation of architectural and explainability-related decisions, contributing to internal consistency between implementation artifacts and compliance-oriented materials.

4.6 Audit by Design

The sixth item concerns the audit-by-design perspective. Beyond their descriptive role, the UML artifacts can be interpreted as a visual mechanism for checking traceability and structural consistency in post-hoc XAI documentation. By making architectural relationships explicit, the diagrams support the identification of missing or misaligned explainability elements at a structural level [17].

In practice, undocumented trained models, absent explainability mechanisms, or unclear associations between components can be detected directly through diagram inspection. Rather than aiming at formal verification, the UML model provides lightweight support for human analysis, enabling auditors and software engineers to assess whether explainability-related elements are consistently represented and whether relevant architectural relationships are explicitly captured. By making potential gaps visible at

the diagram level, architectural modeling reduces the effort required to evaluate completeness and coherence during compliance reviews [17].

5 UMLOOMODELER: A RIGID UML PARSER

The UMLOOModeler was developed to support the automated generation of UML class diagrams that remain strictly faithful to the analyzed source code. The primary design goal of the tool is to preserve structural accuracy, which is a critical requirement in post-hoc auditing and compliance-oriented documentation [17].

Several UML generation tools are currently available, offering different levels of automation and abstraction. However, some of these tools either require manual configuration, provide extensive general-purpose modeling functionality that exceeds what is typically required for architectural documentation in compliance settings, or rely on heuristic inferences to reconstruct architectural relationships. While such approaches may be suitable for exploratory software analysis, inferred relationships and implicit assumptions are problematic in regulatory and audit contexts. In the context of high-risk AI systems, architectural diagrams intended to support auditing activities must accurately reflect the system as implemented, making resulting artifacts directly traceable to the source code and free from interpretative inference [1 ,17].

To address these concerns, the UMLOOModeler adopts a conservative extraction strategy. It processes OO Python source code by parsing abstract syntax tree structures and extracting only elements that are explicitly present in the code, such as class definitions, attributes, methods, inheritance relations, and explicit associations or aggregations. No additional generalizations or semantic assumptions are introduced during diagram generation beyond what can be directly evidenced from the source code.

As a consequence of this conservative strategy, the level of automation achieved by the UMLOOModeler is intentionally conditioned by the explicit architectural structure present in the source code. When OO constructs and relationships are clearly defined, the generated UML diagram provides a faithful and expressive representation of the system architecture. In contrast, when architectural responsibilities are weakly structured or implicitly encoded, the resulting diagram reflects these limitations without attempting to compensate through inference or reconstruction. This behavior is a deliberate design choice, as preserving fidelity to the implementation is essential for auditability and compliance-oriented documentation [17].

Figure 3 illustrates this principle by presenting the UML diagram automatically generated for the `ImageModule` example. It does not include the defined stereotypes or tagged values, which were intentionally added in the manually constructed model shown in Figure 2 in order to illustrate the application of the proposed framework. The source code corresponding to this example is available in the repository mentioned previously.

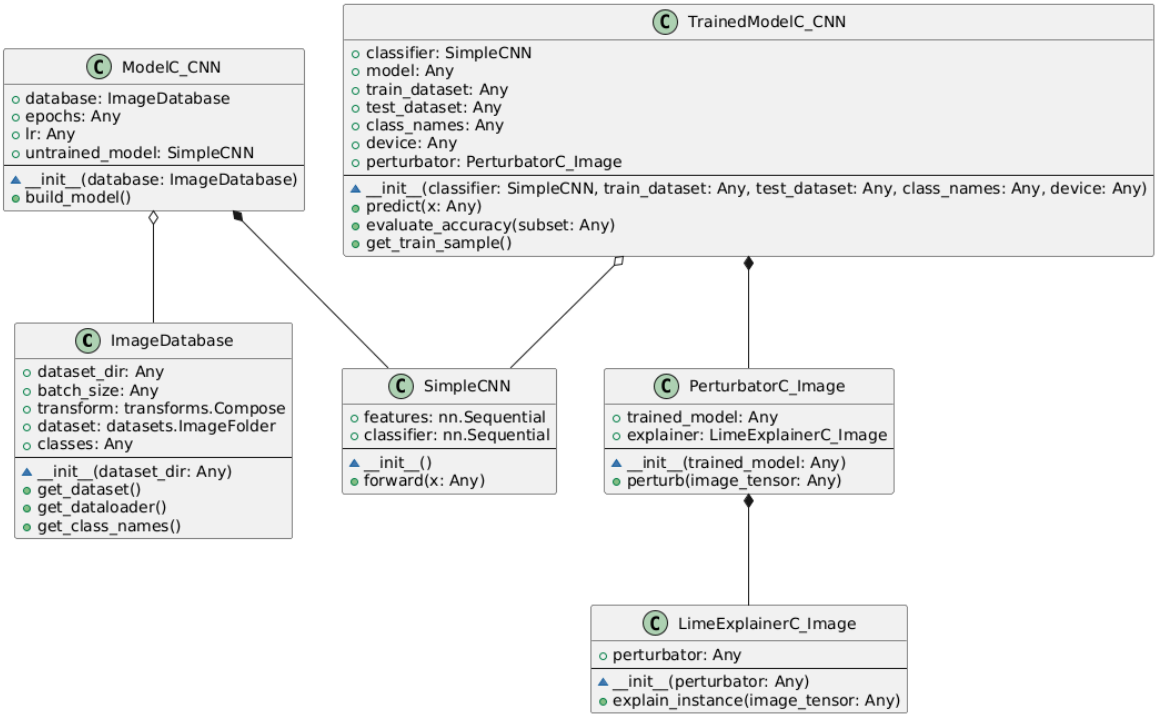

Figure 3: UML class diagram automatically generated by the UMLOOModeler for the ImageModule example.

The UMLOOModeler tool is publicly available as a web-based application, allowing users to generate UML class diagrams directly from Python source code. The current implementation can be accessed at https://umloomodeler.streamlit.app/ .

6 RELATED WORK

Navarro proposed a conceptual metamodel providing a visual representation of explainability in AI systems. His proposal makes use of three main blocks - explanation, technique, and model - and also employs abstract classes, enumerations, and compositions to represent the elements involved in the explanatory process [18].

The proposal is conceptually valid and focuses on high-level abstraction of explanatory techniques. In this way, it provides visual support for the categorization of techniques, formats, and types of explanation. However, it does not include the representation of the internal architecture of systems or the actual execution flow. This is justified by the fact that the work is centered on conceptual modeling, rather than the description of specific implementations.

While Navarro offers a conceptual framework for the visual representation of XAI, this work adopts a different focus by addressing the modeling of concrete system implementations. The proposed framework operates at a lower level of abstraction, emphasizing

architectural traceability and auditability rather than conceptual categorization.

7 CONCLUSION AND FUTURE WORK

The growing deployment of AI systems in regulated and high-risk domains has intensified the demand for transparency, accountability, and traceable decision-making processes. In particular, the approval of the EU AI Act introduces extensive documentation and auditability obligations for high-risk AI systems, imposing significant operational and organizational burdens on companies. Despite the widespread adoption of XAI techniques, current practices remain largely disconnected from software engineering artifacts, offering explanations primarily at the algorithmic level without providing structured, system-level architectural representations capable of supporting regulatory compliance activities.

In practice, organizations currently lack concrete mechanisms to systematically document how explainability components are integrated into AI systems in a manner that satisfies traceability, auditability, and architectural transparency requirements. As a result, compliance efforts tend to rely on fragmented textual documentation, manual reporting processes, and inconsistent interpretations of regulatory requirements texts, leading to increased costs, inefficiencies, and legal uncertainty. This gap is not merely theoretical but reflects a practical absence of engineering-oriented solutions capable of operationalizing XAI within the broader software development lifecycle under regulatory constraints.

To address this problem, this work proposes a UML-based framework for modeling post-hoc XAI systems, explicitly aimed at supporting structured representation, architectural traceability, and compliance-oriented analysis. Rather than treating explainability as an isolated analytical output, the framework models XAI components as explicit architectural elements, allowing explanation mechanisms, models, and data dependencies to be represented within standardized software modeling artifacts. In doing so, the framework establishes a structured link between XAI and software engineering practices, supporting regulatory compliance activities through explicit architectural representations.

The proposed framework extends UML through a minimal set of stereotypes and tagged values that capture key XAI concepts while preserving model readability and architectural fidelity. Rather than focusing on the internal logic of explainability techniques, the framework emphasizes the structural relationships between data, predictive models, and explainers. This design choice enables the representation of heterogeneous XAI pipelines, including different model architectures and explanation mechanisms, in a unified and consistent manner. As demonstrated throughout the paper, the framework is applicable across different data modalities and explanation strategies, reinforcing its model-agnostic and extensible nature.

An additional contribution of this work is the introduction of the UMLOOModeler, a tool that supports the automated derivation of UML diagrams from OO Python source code. By automating the extraction of architectural information, the UMLOOModeler reduces manual modeling effort while ensuring strict alignment between implementation and UML artifacts. This automation directly supports traceability requirements by enabling auditors and software engineers to inspect explanation flows, model responsibilities, and explicit architectural relationships based on verifiable artifacts derived from the system itself. In this sense, the framework and tool jointly contribute to an audit-oriented view of XAI systems, in which UML diagrams act as complementary evidence alongside textual and procedural compliance materials.

Although illustrative examples were employed to demonstrate the applicability of the proposed framework, their role is limited to contextualizing how the framework can be instantiated in practice. While it is not possible to claim that it can be applied to all conceivable AI system configurations, its design is

intended to provide sufficient generality to accommodate heterogeneous models, explanation mechanisms, and data modalities.

Despite its contributions, the proposed framework presents several limitations. First, the framework does not aim to cover all documentation requirements specified by the EU AI Act, especially those related to organizational processes, risk management, and human oversight, which remain outside the scope of UML-based modeling. Second, the applicability of the framework assumes that the system implementation adheres to the minimal architectural contract defined by the proposed stereotypes and relationships. Systems that violate this contract may not be faithfully represented. Third, strict traceability requirements may restrict the use of abstraction mechanisms in large-scale systems, potentially leading to diagram redundancy. Finally, the proposed framework has not yet been evaluated in real-world compliance or auditing scenarios, and its practical effectiveness in such contexts remains to be empirically assessed.

Future work will focus on further validating and extending the proposed framework. First, future investigations will examine whether the framework can support the generation of compliance-oriented documentation for existing AI systems implemented using procedural or weakly OO Python code, aiming to assess the extent to which meaningful architectural traceability can be achieved with minimal refactoring effort.

Second, quantitative metrics will be explored to evaluate the impact of the framework on documentation-related effort, including indicators such as time reduction, manual workload decrease, and overall cost associated with producing compliance-ready technical documentation. Such analyses aim to assess whether the proposed approach can effectively reduce the organizational burden of compliance activities.

Finally, future developments will concentrate on extending the UMLOOModeler to provide deeper analytical support, including the identification of XAI-related architectural contracts, the assisted annotation of extracted models with framework-defined stereotypes and tagged values, and preliminary analyses of compliance-related aspects derived from Annex IV of the EU AI Act. From a longer-term perspective, this line of work may serve as an initial step toward architectural patterns for documenting complex AI systems, including large-scale foundation models and LLM-based applications.

REFERENCES

[1] EUROPEAN UNION. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (AI Act). Official Journal of the European Union, L 1689, 12 July 2024, p. 1–147. Available at: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689

[2] EUROPEAN COMMISSION. Assessment List for Trustworthy Artificial Intelligence (ALTAI) for Policy Makers. High-Level Expert Group on Artificial Intelligence, 2020/2025 update. Available at: https://digital-strategy.ec.europa.eu/en/library/assessment-list-trustworthy-artificial-intelligence-altai-self-assessment

[3] MUSTAC, T.; HENSE, P. SoS AI: An AI System of Systems Approach for EU AI Act Conformity. SSRN Electronic Journal, 2025. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5280073

[4] STAMPERNAS, S.; LAMBRINOUDAKIS, C. A Framework for Compliance with Regulation (EU) 2024/1689 for Small and Medium-Sized Enterprises. Journal of Cybersecurity and Privacy, 2025. https://doi.org/10.3390/jcp5030040

[5] BARREDO ARRIETA, A. et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, v. 58, p. 82–115, 2020. https://doi.org/10.1016/j.inffus.2019.12.012

[6] SAARELA, M.; PODGORELEC, V. Recent Applications of Explainable AI (XAI): A Systematic Literature Review. *Applied Sciences*, 14(19):8884, 2024. https://doi.org/10.3390/app14198884

[7] SELIC, B. A systematic approach to domain-specific language design using UML. In: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), 2007. p. 2–9. https://doi.org/10.1109/ISORC.2007.10

[8] OBJECT MANAGEMENT GROUP (OMG). OMG Unified Modeling Language (UML), Version 2.5. 2015. Available at: https://www.omg.org/spec/UML/2.5

[9] RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. "Why should I trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, 2016. p. 1135–1144. https://doi.org/10.1145/2939672.2939778

[10] MOLNAR, C. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. 2022. Available at: https://christophm.github.io/interpretable-ml-book/

[11] LUNDBERG, S. M.; LEE, S.-I. A Unified Approach to Interpreting Model Predictions. In: Advances in Neural Information Processing Systems (NeurIPS 2017), vol. 30. Available at: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[12] HUMMEL, A. et al. The EU AI Act, Stakeholder Needs, and Explainable AI: Aligning Regulatory Compliance in a Clinical Decision Support System. arXiv preprint arXiv:2505.20311, 2025. Available at: https://arxiv.org/abs/2505.20311

[13] PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 2011. Available at: https://jmlr.org/papers/v12/pedregosa11a.html

[14] SPANHOL, F. A.; OLIVEIRA, L. S.; PETITJEAN, C.; HEUTINCK, L. Breast cancer histopathological image classification using convolutional neural networks. In: 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, 2016, p. 2560–2567. https://doi.org/10.1109/IJCNN.2016.7727519

[15] WEINSTEIN, J. N. et al. The Cancer Genome Atlas Pan-Cancer analysis project. Nature Genetics, v. 45, n. 10, p. 1113–1120, 2013. https://doi.org/10.1038/ng.2764

[16] KLUYVER, T. et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. In: Positioning and Power in Academic Publishing: Players, Agents and Agendas. IOS Press, 2016. https://doi.org/10.3233/978-1-61499-649-1-87

[17] CEDERBLADH, J.; CICCHETTI, A.; SURYADEVARA, J. Early Validation and Verification of System Behaviour in Model-based Systems Engineering: A Systematic Literature Review. ACM Transactions on Software Engineering and Methodology (TOSEM), v. 33, n. 3, 2024, p. 1–67. https://doi.org/10.1145/3631976

[18] NAVARRO, A.; LAVALLE, A.; MATÉ, A.; TRUJILLO, J. A modeling approach for designing explainable Artificial Intelligence. In: ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, CEUR Workshop Proceedings. Available at: https://ceur-ws.org/Vol-3618/forum_paper_24.pdf