

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

José Uetanabara Júnior

Independent Researcher, miklotovx@gmail.com

Artificial Intelligence (AI) has gained prominence in recent years, with widespread adoption raising challenges related to the auditability of AI-based systems. Explainable Artificial Intelligence (XAI) addresses this issue through post-hoc methods that provide interpretation. However, the integration of XAI methods into architectural representations and compliance-oriented documentation remains largely unstructured. At the same time, the European Union's AI Act (Regulation 2024/1689) demands documentation requirements for high-risk AI systems without prescribing a standardized format. As a result, compliance material is often complex to produce and maintain, and may not accurately reflect the system implementation. To address this gap, this work proposes a UML architectural framework for AI systems incorporating post-hoc XAI, focusing on the structural representation of compliance-relevant items required by Annex IV. The framework defines a minimal set of Unified Modeling Language (UML) stereotypes, tagged values, and relationships, based on an architectural contract emerging from object-oriented (OO) Python implementations. As an additional contribution, this work introduces the UMLOOModeler, a tool that generates UML class diagrams from these implementations using a conservative extraction strategy, ensuring consistency between the implementation and architectural representations. The framework is illustrated through heterogeneous AI configurations and a partial example of technical documentation, supporting traceability, auditability, and documentation consistency.

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 regulatory documentation, enabling authorities, auditors, and authorized users to assess compliance, reliability, and accountability [1].

The regulation does not prescribe a standardized format for regulatory documentation, allowing organizations to define how compliance materials are structured and presented, provided that all mandatory items specified in the annexes are addressed. In the absence of an established format, organizations often rely on fragmented information to produce extensive textual reports or model cards for compliance purposes [2]. While these artifacts may describe system behavior and performance, they not directly provide a structured architectural representation of their components. As a result, the documentation may not faithfully reflect how the system operates in practice. Furthermore, producing and maintaining technical material becomes a complex, time-consuming, and error-prone process.

One of the key items addressed by the Annex IV of the EU AI Act is the need to support interpretability of AI system outputs [1]. While a wide range of post-hoc XAI methods have been extensively studied in the literature, their integration into compliance-oriented

material remains largely unstructured [3, 4]. In practice, these methods are often implemented as separate analytical components without explicit architectural representation. As a result, the relationships between data sources, trained models, and explanations frequently remain implicit, limiting structural traceability and auditability, as observed in compliance assessments [5].

To support this form of integration, architectural modeling languages such as the Unified Modeling Language (UML) are widely used for system documentation and structural representation [6, 7]. However, standard UML does not natively incorporate concepts specific to post-hoc XAI, nor does it explicitly address the documentation obligations introduced by the regulation. Therefore, it does not, by default, provide constructs for modeling explainability within a compliance-oriented context.

To address this gap, this work introduces a UML architectural framework for AI systems incorporating post-hoc XAI, focusing on the structural representation of compliance-relevant items required by Annex IV. It defines a minimal architectural contract connecting data sources, model definitions, trained models, and explainability methods, allowing these elements to be explicitly represented in standardized UML class diagrams. The framework is model-agnostic, data-agnostic, and explainer-agnostic, and supports systems developed following OO design principles, where structural elements can be explicitly identified and structured.

The main contribution of this work is the formalization of a minimal architectural contract that enables the generation of regenerable, traceable, and auditable technical documentation for AI systems incorporating post-hoc XAI methods within a compliance-oriented context. In addition, an initial supporting tool, the UMLOOModeler, is introduced to assist in deriving faithful UML diagrams directly from OO Python source code. By generating architectural representations directly from the implementation, the

resulting UML model can serve as a structural source of truth: if any compliance-relevant element, relationship, or attribute is not represented in the model, it does not exist in the documented system pipeline.

Rather than replacing mandatory textual documentation, the framework and tool are intended to complement existing compliance processes. They aim to support structural consistency, traceability, and auditability of the elements defined in the architectural contract. These elements are associated with several regulatory documentation requirements established under Annex IV. By representing them through explicit modeling, this work introduces an architecture-centric perspective that enables auditability by design, as auditors can validate compliance-relevant items against a formal model representation rather than relying solely on narrative descriptions. This perspective aligns with model-driven engineering practices that treat system models as compliance assets rather than merely descriptive reports [8]. The framework is not restricted to a specific application domain and is expected to be applicable to a broad range of AI systems.

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 increasingly complex. As examples, we can mention Deep Neural Networks, Convolutional Neural Networks (CNN), Recurrent Neural Networks (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, a XAI method is used to interpret the machine learning model and provide the appropriate interpretation [3, 4, 9].

2.2 Local and Model-Agnostic Explanation Methods

Local model-agnostic explainability methods share a central characteristic: they are capable of generating explanations without accessing or modifying the internal structure of the model. 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 model behavior within a small neighborhood [10].
- Shapley Additive Explanations (SHAP) generates explanations by applying cooperative game theory to assign each feature a Shapley value, representing its average marginal contribution to the prediction [11].

2.3 EU AI Act Overview

The European Union’s AI Act establishes the first comprehensive regulatory framework for AI 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 items of technical documentation for high-risk AI systems, while Annex I is considered as a contextual reference for system classification. Such material must enable regulatory authorities, auditors, and authorized users to understand how it operates, assess its reliability, and verify compliance with legal requirements.

In summary, the key items typically include:

- 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.

2.4 Model-Driven Engineering and Architecture-Centric Documentation

Model-Driven Engineering (MDE) is a software development paradigm in which models are treated as primary engineering assets rather than secondary documentation [8]. In this paradigm, system models assume a central role in the engineering process, serving as structured references that reduce discrepancies between specification and implementation, thereby enabling systematic traceability across the software lifecycle.

From an architecture-centric perspective, this paradigm supports the use of structural models as central representations of system organization. When architectural models are explicitly defined and aligned with implementation components, they can serve not only as design documentation but also as compliance-relevant assets. In such contexts, structural relationships, component responsibilities, and declared properties become verifiable elements embedded within the model itself, enabling more consistent traceability and facilitating auditing processes.

3 DOCUMENTATION CHALLENGES IN MODERN AI SYSTEMS

Before introducing the proposed UML architectural framework, it is necessary to examine how AI systems

are currently documented. Organizations typically rely on textual reports, model cards, spreadsheets, and other heterogeneous artifacts to describe data sources, model definitions, trained models, and explainability components [2]. Although these materials may provide relevant information about specific aspects of the system, they often lack structural integration with the actual implementation and architectural design. As a result, documentation and system components tend to evolve independently, creating inconsistencies, reducing traceability, and increasing maintenance effort. Without a structural model that can be systematically regenerated from the implementation, documentation becomes a narrative description rather than a verifiable representation.

In the context of high-risk AI systems subject to Annex IV requirements, technical documentation cannot afford to be inaccurate or structurally inconsistent, as this may undermine transparency and complicate accountability in regulated environments [1]. Therefore, there is a need for a structured modeling framework capable of generating compliance material that faithfully reflects the operational architecture of the AI system.

4 A UML ARCHITECTURAL FRAMEWORK FOR MODELING AI SYSTEMS UNDER THE EU AI ACT

To address the structural challenges identified in the previous section, this chapter introduces a UML architectural framework designed to represent compliance-relevant elements of AI systems required by Annex IV of the EU AI Act [1]. To demonstrate applicability, an illustrative breast cancer diagnosis system is used.

4.1 Illustrative Breast Cancer Diagnosis System

Figure 1 presents an illustrative breast cancer diagnosis system used to instantiate the framework. The system is modeled at a high level of abstraction as a modular neural network organized into three packages, namely `ClinicalModule`, `ImageModule`, and

`GeneticModule`, each responsible for processing a distinct data modality. The shared `DataEntry` interface represents the entry point for the data consumed by these modules. The relationships between the interface and the packages are included solely for conceptual clarity and do not represent execution flow or system integration logic.

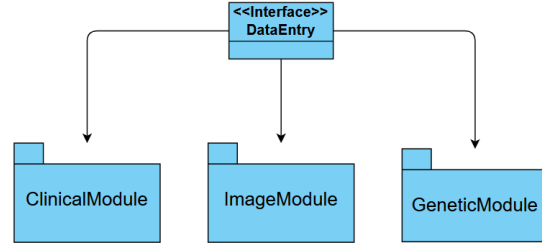


Figure 1: High-level representation of the illustrative system.

Of the three modules presented, the `ClinicalModule` was implemented using the Wisconsin Breast Cancer dataset available in the scikit-learn library [12]. The `ImageModule` was implemented using the Breast Cancer Histopathology image dataset [13], while the `GeneticModule` was implemented using genomic expression from The Cancer Genome Atlas [14].

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 framework to demonstrate how heterogeneous data modalities, learning models, and post-hoc explanation methods can be documented using UML.

4.2 Detailed UML Modeling

This work starts from the concrete structure of OO source code implemented in Python within the Jupyter Notebook environment [15]. The implementation strategy adopted in this work emerged from the incremental organization of the code during development, when the following responsibilities were identified and isolated into explicit software constructs: data access, model definition, custom classifier, trained

model and explainability. This organization made explicit a minimal architectural contract required for the framework to operate as intended and is shown in Figure 2.

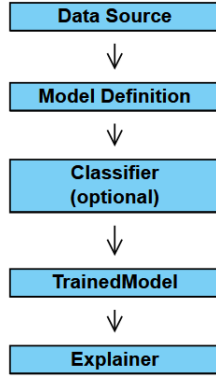


Figure 2: Core implementation responsibilities structured in the code, making explicit the minimal architectural contract.

With the architectural contract defined, a mapping to UML was performed through the introduction of specific stereotypes, enabling its representation within a class diagram. The UML class diagram was selected due to its suitability for representing static structure and explicit relationships among architectural components [6, 7]. All stereotypes used in the diagram are defined as extensions of the UML Class metaclass and are summarized in Table 1.

Multiple implementations were developed following this architectural contract, each corresponding to a different local model-agnostic explanation method. The complete source code used in the illustrative examples is publicly available at: github.com/miklotovx/UMLOOModeler.

Figure 3 presents the UML modeling derived from the illustrative system used to instantiate the framework. Rather than focusing on a single module, the diagram integrates structures from each module to demonstrate how heterogeneous AI system components can be represented. The diagram was manually constructed based on the class diagrams generated by the UMLOOModeler tool, which is discussed in more detail

in Chapter 6, with modeling decisions explicitly guided by the corresponding source code. The attributes, methods, and parameters shown in the diagram accurately reflect the entities manipulated in the implementation. Non-essential details were intentionally omitted to improve readability.

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 `ClinicalDatabase`, `ImageDatabase`, and `GeneticDatabase` classes represent the data sources used by the illustrative system. The <<DataSource>> stereotype was used to indicate data input. 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.

The `ModelA_MLP` and `ModelB_RF` classes represent model definition elements 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 a trainable model.

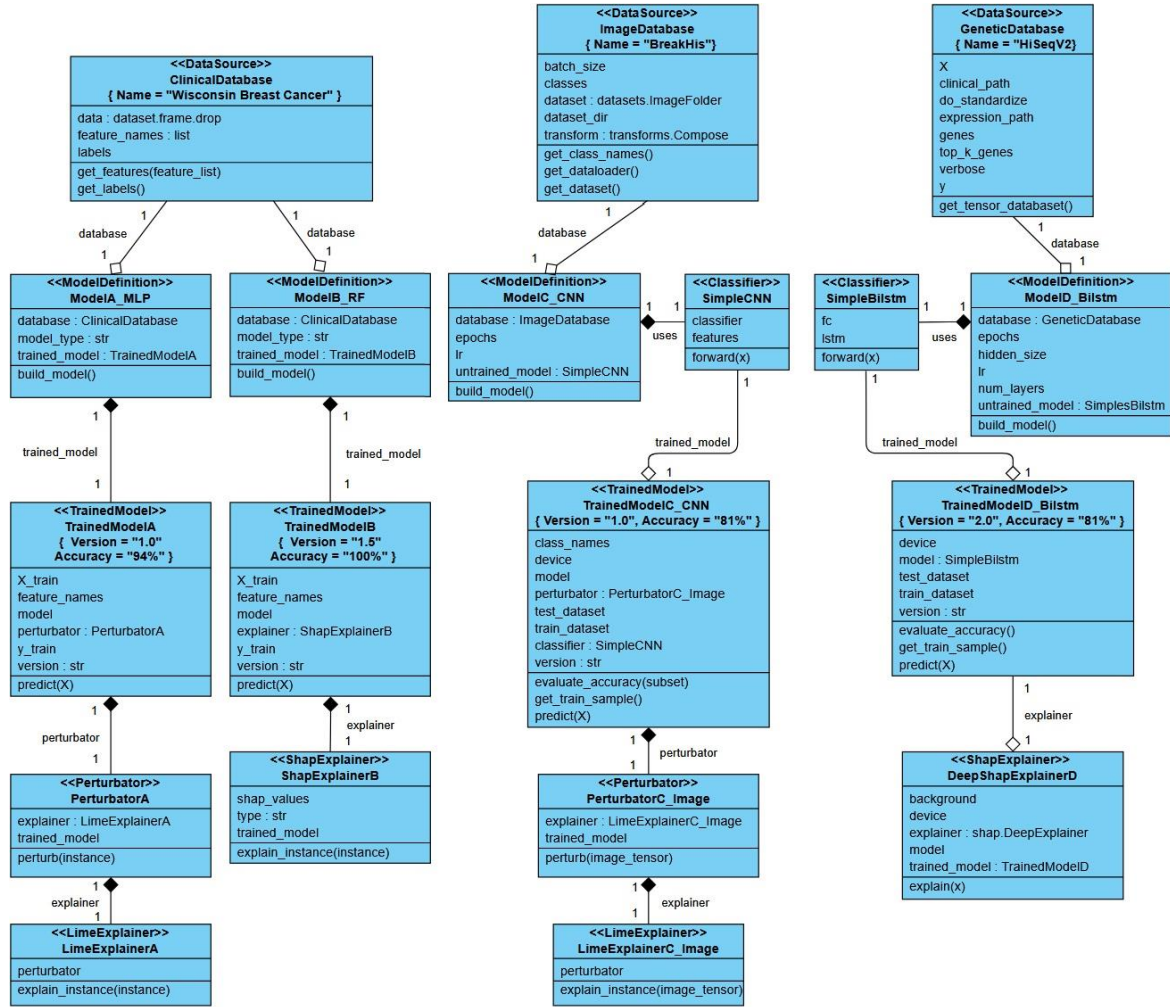


Figure 3: Detailed UML modeling of the illustrative system.

The `TrainedModelA` and `TrainedModelB` classes represent the trained MLP model and the trained RF model, respectively. These classes store the trained classifier and the training data. 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 LIME,

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. The `<<ShapExplainer>>` stereotype was applied to this class to represent the explanations.

The `ModelC_CNN` class represents a model definition element responsible for instantiating and training a convolutional neural network using the 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 custom 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 and, as with the previously described trained models, includes tagged values used to record versioning and performance information. The `<<TrainedModel>>` stereotype was applied to indicate that this class corresponds to a trained model. It is explained using LIME, following the same explanatory pattern adopted for `TrainedModelA`, in which the `PerturbatorC_Image` and `LimeExplainerC_Image` classes are assigned the

`<<Perturbator>>` and `<<LimeExplainer>>` stereotypes, respectively.

The `ModelD_Bilstm` class represents a model definition element 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 and, 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. It is explained using the DeepSHAP method, which extends the SHAP framework to deep learning models by combining backpropagation-based attribution with Shapley value principles.

The choice of UML relationships in the diagram reflects how explainability methods are integrated into the implementation, rather than being predefined by the framework. 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 implementation strategies are reflected through appropriate UML relationships.

For instance, in the implementation using the `GeneticDatabase`, where a `Bilstm` model is explained with DeepSHAP, the explainer is instantiated externally and receives the trained model as input, which results in an inversion of the aggregation relationship. This design choice 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 traceability, auditability, 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 Section 5.3.

4.3 Application of the UML Framework

To demonstrate the application of the framework, a partial example of technical documentation for a simplified AI system was produced. This compliance material illustrates how the framework supports several items of Annex IV of the EU AI Act, with a detailed discussion of this coverage provided in Section 5.1. For this purpose, an OO Python implementation based on the architectural contract introduced earlier was used. The code (`clinical_shap.py`) and the PDF (`clinical_shap_example_documentation.pdf`) containing the example regulatory documentation are available in the same repository mentioned previously.

The `clinical_shap.py` implementation performs a binary classification task based on the Wisconsin Breast Cancer dataset using a Random Forest model and incorporates a post-hoc explainability method based on SHAP. In the code, the architectural contract can be observed through the OO structure, which explicitly represents the main architectural elements of the system.

From this implementation, a UML class diagram containing these architectural elements and their relationships was generated using the UMLOOModeler tool. Based on this diagram, the defined stereotypes and tagged values were manually applied to instantiate the architectural contract described earlier. This step associates each structural element of the UML model with its corresponding architectural role within the framework.

The generated UML diagram was then incorporated into Section 1 (Figure 1) of the example technical document together with a textual description of the system's general characteristics. This allows the complete system structure and its architectural flow to be inspected at the very beginning of the technical documentation, providing immediate architectural visibility. The remaining sections of the document refer to Figure 1, accompanied by additional textual descriptions. This results from the diagram serving as the structural source of truth for the compliance material, representing a UML model faithfully derived from the system source code, where structural relationships and responsibilities are explicitly represented. Consequently, if a component, attribute, or relationship does not appear in the diagram, it does not exist in the implemented system and therefore should not be described in the regulatory documentation. This ensures consistency between implementation and documentation while providing clear structural traceability and auditability by design across the documented system elements.

The use of a UML model in the example document as the structural source of truth also enables the regeneration of the technical documentation. To illustrate this concept, Section 2 intentionally contains an extended textual description compared to the other sections. This description accurately reflects the current version of the system (version 1.5). However, if the system is updated, part or all of this text may need to be rewritten to remain consistent with the implementation. Depending on the magnitude of the update, additional sections of the document may also require modification. To mitigate this issue, Figure 1 should be used as the central reference for the system architecture, allowing the textual load to be reduced. Thus, when the system evolves, updating the diagram is typically sufficient, requiring only minor and localized updates. This architecture-centric approach supports the regeneration of the compliance material and may facilitate the maintenance over time.

It is acknowledged that the framework does not cover all items required for full compliance with the EU AI Act. Some, such as risk management, governance, and organizational procedures, remain outside the scope of this work. The framework focuses primarily on the structural representation of AI systems and their explainability methods, aiming to support transparency, traceability, and auditability at the architectural level. A broader discussion is presented in the next chapter.

5 DISCUSSION

As previously presented, the proposed UML architectural framework was instantiated in an illustrative breast cancer AI system, and a partial example of technical documentation was produced covering several items specified in Annex IV of the EU AI Act. While these examples demonstrate its applicability in practice, certain points require further discussion. This chapter presents a mapping between modeling elements and regulatory documentation requirements, the model-agnostic characteristics across data, models, and explainability methods, the trade-offs between abstraction and traceability in compliance-oriented modeling, the role of tagged values as compliance-related metadata, and the potential benefits for AI system auditing and documentation.

5.1 Framework Coverage of Annex IV Documentation Requirements

The first item to be discussed concerns the coverage provided by the framework with respect to the regulatory documentation requirements of Annex IV. While this work aims to support structural traceability and auditability for AI systems, it does not address all items demanded by the regulation. To clarify its scope and limitations, this subsection focuses specifically on

the requirements defined in Annex IV, which establishes the mandatory items for high-risk AI systems [1].

As demonstrated in the example technical documentation presented in Section 4.3, the UML diagram generated from the implementation adhering to the architectural contract allows several items required for regulatory documentation to be represented structurally. Table 2 summarizes the main regulatory items and indicates how each item is structurally supported by the framework in this example.

Other regulatory requirements intentionally remain outside the modeling scope of this work. This decision is motivated by abstraction and usability concerns: representing such requirements in UML would increase diagram complexity without proportional benefits for auditors and software engineers. Consequently, they are more appropriately addressed through traditional textual material, reinforcing the role of the UML framework as a complementary mechanism rather than a replacement for regulatory documentation [16].

5.2 Model-Agnosticity Across Data, Models, and Explainers

The second item to be discussed concerns the model-agnosticity. As previously demonstrated in Section 4.2, the architectural contract introduced by the framework allows different AI systems to be documented using the same structural representation, provided that this contract is respected. This includes systems operating over distinct data modalities, such as tabular, image, and sequential data, employing different learning models, including both black-box and white-box configurations, and integrating different post-hoc explainability methods. While the framework is not intended to cover all possible scenarios, the architectural contract was designed to provide sufficient generality to document heterogeneous AI systems while preserving structural consistency and traceability.

Table 2: Coverage of EU AI Act regulatory documentation requirements by the 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 <code><<DataSource>></code> , <code><<ModelDefinition>></code> , <code><<TrainedModel>></code> , and <code><<ShapExplainer>></code> .
3. Training, validation, and testing data	Datasets used, their origin, and quality	Tagged values <code>{Name}</code> in <code><<DataSource>></code> enabling dataset identification.
4. Models and parameters	Model types, versions, and relevant parameters	Tagged values <code>{Version}</code> in <code><<TrainedModel>></code> ; model type implicit in class name.
5. Logging and traceability	Recording and audit of decisions	Logging is not covered. Structural traceability enabled through UML stereotypes and relationships.
6. Metrics and testing	Performance indicators, robustness, and fairness	Tagged value <code>{Accuracy}</code> in <code><<TrainedModel>></code> .
7. Explainability	Methods, limitations, and explanations	Stereotype <code><<ShapExplainer>></code> .
8. Risk management	Risk identification and mitigation	Not covered.
9. Governance and quality	Internal processes, roles, responsibilities	Not covered.

5.3 Abstraction and Traceability

The third item to be discussed concerns abstraction and traceability. The EU AI Act requires that technical documentation support traceability, requiring that model versions, training datasets, and performance metrics be individually identifiable [1]. For this reason, in the illustrative system instantiated using the framework, generic abstractions that could introduce many-to-many relationships were deliberately avoided in order to show how this regulatory requirement can be satisfied in practice. 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. In the instantiated

examples, this repetition remains manageable due to the limited scale of the system. However, in larger AI systems incorporating 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. Attempts to reduce redundancy through abstraction may complicate auditing processes by obscuring the explicit relationships between datasets, models, and explainers. 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.

5.4 Tagged Values as Compliance Points

The fourth item to be discussed concerns the use of tagged values. In the example technical documentation presented in Section 4.3, two tagged values, `Version` and `Accuracy`, were associated with `TrainedModelB` to explicitly record the trained model's version and its performance. However, in other AI systems different metrics may be more appropriate, such as sensitivity, F1-score, or other domain-specific indicators. For this reason, rather than relying on a fixed or predefined vocabulary, the framework does not impose additional restrictions on the use of tagged values beyond those already provided by UML. This allows software engineers to define name-value pairs as needed to document compliance-related metadata directly in the model.

It should be noted that excessive use of tags may overload diagrams and reduce their readability. For this reason, they should be applied focusing on information that is directly relevant to compliance. All remaining details should be documented in the accompanying textual material.

5.5 Potential benefits for Post-Hoc Auditing and System Documentation

The fifth item to be discussed concerns the potential benefits of the framework for AI system documentation and compliance-oriented analysis. Because the EU AI Act does not prescribe a standardized format for compliance material, organizations may structure and present it in different ways, provided that the mandatory items specified in the regulation are addressed [1]. In practice, this flexibility often leads to documentation composed of multiple artifacts, including extensive textual reports, model cards, or runtime-oriented descriptions [2].

Based on the illustrative system presented in Section 4.2 and the example technical documentation discussed in Section 4.3, it is possible to identify several potential benefits in qualitative terms. In particular, the framework can support the preparation, maintenance,

and inspection of AI system compliance material through architectural modeling.

The first potential benefit concerns the use of the UML diagram as a structural source of truth for documentation. The diagram is derived from the implementation adhering to the architectural contract defined by the framework, ensuring consistency with the source code. This allows architectural elements of the system to be systematically mapped and described in the technical documentation associated with the items specified in Annex IV of the EU AI Act. As a result, inconsistencies between implementation and documentation can be reduced, since missing or outdated components become visible at the diagram level.

The second potential benefit concerns the structural visibility of the system architecture. In the example technical documentation presented in Section 4.3, the UML diagram is placed at the beginning of the document, allowing the overall structure of the AI system, as well as relevant metadata associated with its components, to be inspected before the detailed textual descriptions are introduced. By explicitly representing architectural elements, the diagram provides a compact overview of the system structure, facilitating the understanding of how it operates and how its main components interact.

The third potential benefit concerns the ability to regenerate and maintain system compliance material. The architecture-centric approach employed by the framework supports the partial regeneration and maintenance of the regulatory documentation over time. When updates occur in the implementation, the UML diagram can be regenerated and used as the primary reference for updates. Instead of requiring extensive textual revisions across multiple sections, the diagram can be updated first, followed by targeted textual adjustments when necessary. This process can help organizations maintain consistency between implementation components and compliance material as the system evolves. From a broader perspective, this

reflects an architecture-centric and model-driven approach, in which the UML model operates as a central artifact from which technical documentation can be systematically organized and partially regenerated.

The fourth potential benefit concerns auditability. The use of the framework promotes auditability by design, since the architectural representation of the AI system, including its XAI components, can be inspected directly through the UML diagram. Rather than relying solely on narrative descriptions, auditors can validate system structure against the architectural diagram.

Structural traceability also becomes more explicit in this representation. Undocumented trained models, missing explainability methods, and unclear associations between system components can be detected through diagram inspection. The UML model provides lightweight support for both human analysis and technical documentation by explicitly representing system components, their relationships, and relevant properties within the diagram. By making potential gaps visible at the diagram level, architectural modeling may reduce the effort required to evaluate completeness and coherence during compliance reviews [20], while strengthening structural transparency and consistency across the compliance material required by the EU AI Act [1].

6 UMLOOMODELER: A STRICT 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 for auditing and regulatory documentation [1].

Several UML generation tools exist 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 structural relationships. While such tools may be suitable for general software engineering use, 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, ensuring that they are directly traceable to the source code and free from interpretative inference [1].

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 structural constructs explicitly present in the code, such as class definitions, attributes, methods, inheritance relations, and explicit associations, aggregations, or compositions. No additional generalizations or semantic assumptions are introduced during diagram generation beyond what can be directly evidenced from the source code.

As a result of this conservative strategy, the level of automation achieved by the UMLOOModeler is intentionally conditioned by the explicit architectural structure present in the implementation. 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.

Figure 4 presents this principle using 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 3 in order to illustrate the application of the proposed UML architectural framework. The source code (`image_lime.py`) is available in the repository referenced in Section 4.2.

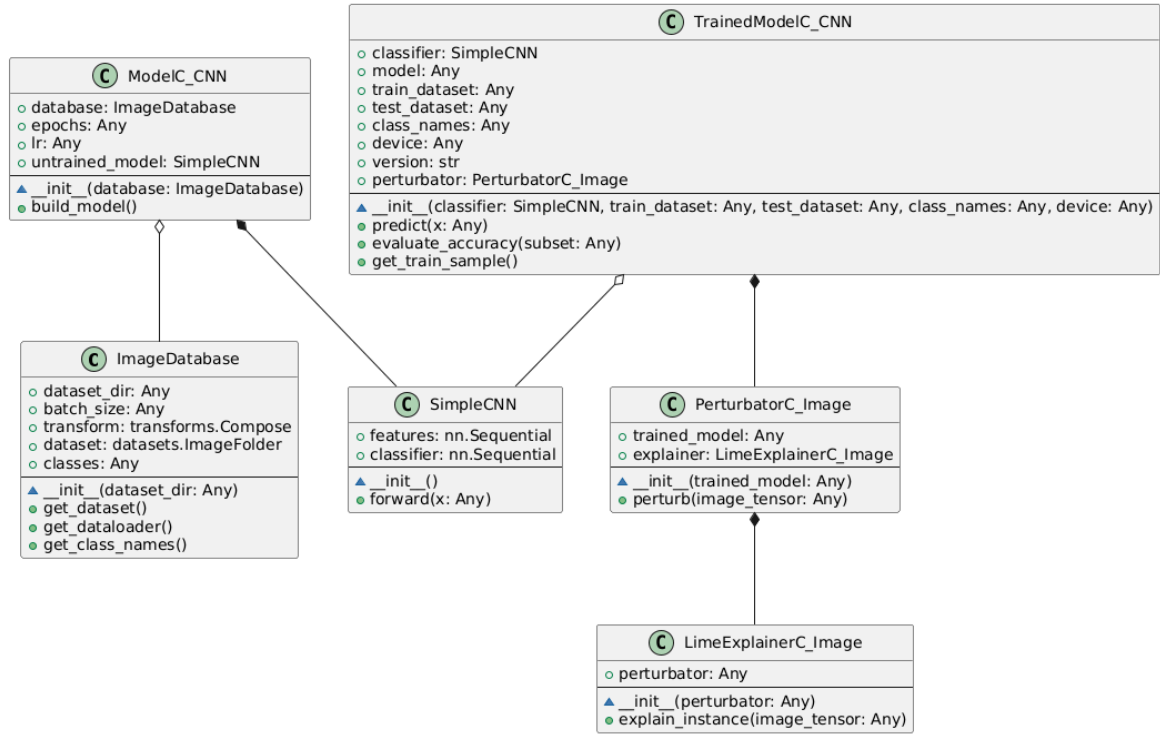


Figure 4: UML class diagram automatically generated by the UMLOOModeler for the ImageModule example.

The UMLOOModeler tool is publicly available as a web-based application, allowing users to generate UML class diagrams directly from Python implementations. The UMLOOModeler can be accessed at <https://umlloomodeler.streamlit.app/>.

7 RELATED WORK

Navarro *et al.* proposed a conceptual metamodel providing a visual representation of explainability in AI systems. The proposal uses three main blocks: explanation, technique, and model, and employs abstract classes, enumerations, and compositions to represent the elements involved in the explanatory process [17].

The metamodel focuses on a high-level abstraction of explanatory techniques, providing 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 their concrete implementation. This is consistent with its focus on conceptual modeling rather than on the description of implemented systems.

Gonçalves *et al.* proposed a modular framework for achieving explainability and regulatory compliance in high-risk AI systems. The framework integrates XAI techniques, compliance-by-design principles, and MLOps practices through a dual-flow architecture,

generating runtime evidence artifacts to support auditability under the GDPR and the EU AI Act [18].

While their framework contributes to operational compliance through the generation of runtime evidence, its reliance on dynamic artifacts may increase the burden of managing and analyzing compliance material over time.

Lucaj *et al.* proposed a set of technical documentation templates aimed at supporting compliance by providing structured guidelines for organizing required information in high-risk AI systems. The approach defines three separate templates (data, model, and application) with standardized sections aligned with Annex IV requirements, facilitating the preparation of compliance material by different stakeholders [19].

The proposed templates are directly aligned with regulatory needs, offering clear guidance for structuring technical documentation. However, they remain centered on textual artifacts, which require continuous updates as the system evolves. This may lead to inconsistencies between the implemented system and the documented material, particularly in dynamic development environments where changes occur frequently.

While these works provide relevant contributions to AI system documentation, they do not establish a direct structural link between the implemented system and the produced documentation, leading to the problems discussed in Section 3 and Section 5.5. For this reason, this work adopts an architecture-centric perspective to address these limitations.

8 CONCLUSION AND FUTURE WORK

The introduction of the EU AI Act imposes extensive regulatory documentation requirements for high-risk AI systems, aiming to support the assessment of compliance, reliability, and accountability [1]. The regulation does not prescribe a standardized format, allowing organizations to define how compliance material is structured and presented, provided that all

mandatory items specified in the annexes are addressed. This flexibility often leads organizations to rely on fragmented information to produce extensive textual reports or model cards [2]. While these artifacts may describe system behavior and performance, they do not provide a structured architectural representation of system components, and therefore may not faithfully reflect how the system operates in practice.

One of the key requirements of the regulation is the need to support interpretability of system outputs [1]. However, despite the widespread adoption of post-hoc XAI methods, organizations still lack concrete engineering mechanisms to systematically document how these methods are structurally integrated into AI systems in a manner that satisfies traceability and auditability requirements. As a result, compliance efforts often rely on manual processes and inconsistent practices, leading to increased costs, inefficiencies, and legal uncertainty [3]. This gap reflects the absence of engineering-oriented solutions capable of explicitly representing explainability within the system architecture under regulatory constraints.

To address this gap, this work proposed a UML architectural framework for AI systems incorporating post-hoc XAI, focusing on the structural representation of compliance-relevant items required by Annex IV [1]. By relying on an explicit architectural contract and adopting explicit 1-to-1 relationships, the framework establishes a structural foundation that supports the consistent and traceable representation of relationships between data, models, and explainability methods within standard software engineering constructs [7].

An additional contribution of this work is the introduction of the UMLOOModeler, a tool that supports the automated generation of UML diagrams from OO Python source code. The tool follows a conservative extraction strategy, ensuring that all extracted constructs correspond directly to elements explicitly present in the implementation, without interpretative inference. Poorly structured implementations are therefore directly reflected in the

resulting architectural representation and consequently in the documentation. Elements not present in the implementation are neither represented in the UML model nor propagated to the documentation, preventing the inclusion of inferred or non-existing information. This guarantees that the UML diagram and the produced technical documentation faithfully reflect the implemented system.

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 constructs derived from the system itself. When the implementation adheres to the architectural contract defined by the framework, these elements can be consistently identified and validated. In this sense, the framework and tool jointly contribute to an audit-oriented view of AI systems, in which UML diagrams act as complementary evidence alongside textual and procedural compliance materials. This design choice reflects a fundamental requirement in compliance-oriented contexts, where architectural representations must be provided from verifiable evidence rather than interpretative inference.

In Section 4.3, an example technical documentation demonstrates how an architectural model derived from the framework can be used to structure AI system documentation. The resulting representation enables the identification and organization of information required to address several items defined in Annex IV [1]. In this context, the use of a UML diagram as a central architectural reference supports the reduction of redundant textual descriptions and facilitates the identification of missing or inconsistent information. From this perspective, the UML model acts as a structural source of truth, allowing technical documentation to be partially regenerated and more easily maintained as the system evolves through localized textual updates, reducing the need for extensive revisions.

Despite its contributions, the framework presents several limitations. First, it does not aim to cover all items defined in the regulation [1], particularly those related to organizational processes, risk management, and human oversight, which fall outside the scope of UML architectural modeling. This reflects a deliberate design decision to preserve usability, avoiding excessive diagram complexity that would arise from representing requirements not directly tied to system structure. Second, the applicability of the framework depends on the system adhering to the minimal architectural contract, which may not be satisfied by all existing implementations. Third, the illustrative technical documentation presented in Section 4.3 is based on a simplified AI system, and therefore does not capture the full complexity of real-world deployments. Fourth, strict traceability requirements limit the use of abstraction. Redundancy may be introduced to preserve explicit 1-to-1 relationships between elements, which can lead to increased diagram complexity in large-scale systems. Finally, the 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 framework in real-world AI systems. Additionally, its applicability to systems implemented using procedural or weakly object-oriented Python code will be evaluated, aiming to assess the extent to which meaningful architectural traceability can be achieved with minimal refactoring effort. Furthermore, quantitative metrics will be explored to evaluate the impact on documentation-related effort. Such analyses aim to assess whether the framework 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 items derived from Annex IV of the EU AI Act [1]. From a longer-term perspective, this line of work may contribute to future research on architectural patterns for documenting complex AI systems.

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] MITCHELL, M.; WU, S.; ZALDIVAR, A.; BARNES, P.; VASSERMAN, L.; HUTCHINSON, B.; SPITZER, E.; RAGHU, M.; GEBRU, T. Model Cards for Model Reporting. In: Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19). New York, NY, USA: ACM, 2019. p. 220–229. <https://doi.org/10.1145/3287560.3287596>
- [3] 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>
- [4] 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>
- [5] SCANTAMBURLO, T.; FALCARIN, P.; VENERI, A.; FABRIS, A.; GALLESE, C.; BILLA, V.; ROTOLO, F.; MARCUZZI, F. Software Systems Compliance with the AI Act: Lessons Learned from an International Challenge. In: Proceedings of the 2nd International Workshop on Responsible AI Engineering (RAIE '24). Lisbon, Portugal: ACM, 2024. p. 1–8. <https://doi.org/10.1145/3643691.3648589>
- [6] 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>
- [7] OBJECT MANAGEMENT GROUP (OMG). OMG Unified Modeling Language (UML), Version 2.5. 2015. Available at: <https://www.omg.org/spec/UML/2.5>
- [8] BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-Driven Software Engineering in Practice. 2nd ed. Morgan & Claypool Publishers, 2017.
- [9] MOLNAR, C. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. 2022. Available at: <https://christophm.github.io/interpretable-ml-book/>
- [10] 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>
- [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] 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>
- [13] 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>
- [14] 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>
- [15] 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>
- [16] 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:2505.20311v3, 2026. Available at: <https://arxiv.org/abs/2505.20311v3>
- [17] 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
- [18] GONÇALVES, D.; CORREIA, A. XAI-Compliance-by-Design: A Modular Framework for GDPR- and AI Act-Aligned Decision Transparency in High-Risk AI Systems. Journal of Cybersecurity and Privacy, v. 6, 2026, Art. 43.
- [19] LUCAJ, L.; LOOSLEY, A.; JONSSON, H.; GASSER, U.; VAN DER SMAGT, P. TechOps: Technical documentation Templates for the AI Act. arXiv preprint arXiv:2508.08804, 2025. Available at: <https://arxiv.org/abs/2508.08804>
- [20] CEDERBLADH, J.; CICCETTI, 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>