MLOps-Driven Automation of Regulatory Documentation for AI-Based Medical Software

Fabrizio Rosmarino*, Giulio Mallardi, Luigi Quaranta and Filippo Lanubile

Dept. of Computer Science, University of Bari Aldo Moro, Bari, Italy

Abstract

In recent years, the increasing integration of AI within clinical software has led to the emergence of Software as a Medical Device (SaMD), a category of systems subject to strict regulatory oversight. A major challenge for developers in this domain is the continuous production of regulatory documentation, which remains largely manual and disconnected from development pipelines. This paper proposes a strategy to automate documentation generation by embedding MLOps principles—such as traceability, reproducibility, and continuous integration—into the development workflow. Applied to a representative healthcare AI project, the approach produced consistent, audit-ready artefacts with minimal manual effort, demonstrating its potential to narrow the gap between rapid innovation and regulatory compliance.

Keywords

MLOps, Software as a Medical Device, Regulatory Compliance, Documentation Automation, CI/CD

1. Introduction

Artificial Intelligence (AI) is increasingly embedded in clinical software, giving rise to a new class of regulated systems known as Software as a Medical Device (SaMD) [1]. In Europe, SaMD is regulated under frameworks such as EU MDR 2017/745 [2], which introduce strict compliance obligations across the product lifecycle. AI-based SaMD poses unique challenges due to its dynamic nature and need for rigorous validation to ensure clinical safety and effectiveness. Development processes must comply with stringent international standards covering quality management, risk analysis, and software lifecycle control. Central to this compliance is the continuous production of technical documentation—often a manual, error-prone task disconnected from development workflows [3].

From an MLOps perspective, the lack of integration between development pipelines and regulatory processes represents a major gap. Unlike traditional DevOps, MLOps for regulated AI systems must support not only traceability and reproducibility, but also generate artifacts required for audit and certification purposes. In this context, automation of documentation becomes a key enabler for scaling and operationalizing AI-based SaMD development.

2. Background and Related Work

2.1. MLOps in Regulated Software Development

MLOps (Machine Learning Operations) is an extension of DevOps tailored to the unique challenges of machine learning systems. It focuses on automating and managing the entire ML lifecycle—from data ingestion and model training to deployment and monitoring [4]. While MLOps is well-established in commercial environments, its adoption in regulated sectors, such as medical software, remains limited. In these contexts, workflows must meet strict requirements for traceability, reproducibility, and auditability—characteristics that are not always guaranteed by standard MLOps pipelines [5].

MLOps@ECAI 2025: 1st Workshop on Machine Learning Operations, Bologna, Italy

*Corresponding author.

fabrizio.rosmarino@uniba.it (F. Rosmarino); giulio.mallardi@uniba.it (G. Mallardi); luigi.quaranta@uniba.it (L. Quaranta); filippo.lanubile@uniba.it (F. Lanubile)

0009-0009-8068-2008 (F. Rosmarino); 0000-0001-5847-117X (G. Mallardi); 0000-0002-9221-0739 (L. Quaranta); 0000-0003-3373-7589 (F. Lanubile)

Common tools like GitHub Actions, Docker, and MLflow support automation and versioning but are not designed to produce the regulatory artefacts required by standards such as MDR 2017/745, ISO 13485, or IEC 62304. As a result, documentation and compliance tasks are often handled separately, leading to inefficiencies and increased risk. This limitation has been explicitly highlighted by Granlund et al., who studied the development of certified medical software and noted that CI/CD pipelines must be augmented with regulatory-aware processes to ensure audit-ready artefact [6]. Bridging this gap by integrating regulatory documentation into automated ML pipelines is essential for enabling scalable, compliant development of AI-based medical software.

2.2. Regulatory Documentation for SaMD

Software as a Medical Device (SaMD) is defined by the International Medical Device Regulators Forum (IMDRF) as software intended to be used for medical purposes without being part of a physical medical device [7]. In the European Union, SaMD falls under the Medical Device Regulation (MDR) 2017/745, which mandates strict requirements for technical documentation, risk analysis, and software lifecycle control. Developers must demonstrate compliance with standards such as ISO 13485 for quality management [8], ISO 14971 for risk management [9], and IEC 62304 for software lifecycle processes [10]. These frameworks require the production of structured, auditable documents, including software architecture, verification plans, known anomalies, and traceability matrices. Of particular importance is the handling of third-party dependencies, referred to as Software of Unknown Provenance (SOUP). According to IEC 62304, each SOUP component must be identified, its version tracked, and its risks assessed and mitigated. This creates a significant burden for developers using modern open-source libraries, which often change rapidly and have incomplete or misaligned documentation. To support compliance, regulatory bodies and initiatives such as OpenRegulatory[1] provide templates for key documents, including the Software List and SOUP List. These templates can serve as practical guides, but they still rely on manual compilation, which can introduce inconsistencies and hinder reproducibility.

2.3. Automated Documentation Generation

Although documentation has long been acknowledged as vital in safety-critical domains, relatively few efforts have focused on automating this process in the context of regulated machine learning systems. Traditional DevOps tools, such as Sphinx and Doxygen, can generate documentation from annotated code; however, they do not support regulatory requirements defined in standards like ISO 13485 or IEC 62304. MLOps platforms such as MLflow, Kubeflow, and Metaflow emphasize pipeline orchestration, experiment tracking, and model versioning [4], but lack capabilities for generating the structured, auditable documentation required for compliance. These platforms prioritize internal traceability over regulatory transparency. OpenRegulatory provides documentation templates aligned with MDR and ISO standards, and these are widely adopted by European startups developing SaMD. However, these templates are intended for manual use with word processors or spreadsheets, offering no support for integration with CI/CD workflows or automated validation. Some recent academic efforts have proposed partial automation of documentation, particularly in the context of software safety cases [11]. Notably, Mallardi et al. [12] introduced a framework for responsible AI certification that incorporates structured documentation into MLOps pipelines to address MDR-related compliance needs. However, the approach remains general-purpose and does not yet offer tooling for specific artefact generation. To our knowledge, no publicly available systems currently provide end-to-end automation of MDR-compliant documentation directly from machine learning development pipelines.

3. Proposed Approach

This section describes the architecture and design rationale of our system, which automates the generation of documentation artefacts required under the EU MDR 2017/745 regulatory framework. The

[1]https://openregulatory.com/

solution is designed to integrate seamlessly into MLOps pipelines and focuses on two key documents: the Software List and the SOUP (Software of Unknown Provenance) List. The system is implemented in Python, emphasizing modularity, portability, and maintainability. Its design supports CI/CD workflows (e.g., GitHub Actions) and enables incremental development through distinct but interconnected stages, which are detailed in the following subsections.

3.1. Analysis of Requirements and Normative Alignment

The first phase involved identifying documentation requirements based on the European standards outlined in Section 2. From this analysis, we identified a subset of documents that are (i) critical to certification and (ii) potentially automatable. Notably, we focused on the Software List and the Software of Unknown Provenance (SOUP) List, which are essential for demonstrating traceability, component identification, and risk mitigation strategies.

3.2. System Architecture and Design Criteria

The system is designed to operate on well-structured software repositories that follow best practices in software engineering. It is implemented in Python and relies on common tools such as *GitHub*, *PyPI*, and virtual environments. The architectural core is a documentation orchestrator—a component that orchestrates metadata extraction and document rendering via a set of scheduled or event-driven tasks. Among the key design principles:

- **Scalability**: the system can be extended to include additional document types, enabling future expansion to cover a broader range of regulatory artefacts.
- **Maintainability**: the documentation logic is modular and separated from the templates, allowing for independent updates and customisation.
- **Portability**: the system is compatible with standard CI/CD infrastructures (e.g., GitHub Actions), ensuring ease of integration across projects.
- **Auditability**: all automated processes are logged, and versioned outputs are maintained to support traceability and external audit requirements.

3.3. Automatic Data Extraction Mechanisms

The data collection pipeline includes:

- **Static Source Tree Inspection**: used to collect filenames, directory structures, software versions, and file paths within the project repository.
- **Dependency scanning**: performed by parsing files such as `requirements.txt`, `pyproject.toml`, and environment descriptors to extract third-party packages and their versions.
- **PyPI Metadata Retrieval**: each external package is queried on the Python Package Index to retrieve metadata such as license type, maintainer identity, version history, and repository URL. This step uses the `requests` library to interface with PyPI.
- **Git inspection**: Using Git commands, the system extracts repository metadata such as commit history, authorship, timestamps, and change frequency for each file. This allows for detailed temporal tracking and helps identify deprecated or inactive components.
- **Standard library discrimination**: the `sysconfig` and `pkgutil` modules are used to differentiate built-in Python modules from third-party dependencies.

All extracted data are structured and stored in JSON format, supporting persistence across runs and enabling reproducibility and audit readiness. Fields that cannot be resolved automatically, such as risk classification, are left as placeholders for manual input by domain experts.

3.4. Template-Based Workflow for Automated Document Generation

We employed the Jinja2[2] templating engine to define customizable and regulatory-aligned document formats. For each document type (e.g., Software List, SOUP List), we created a structured template with placeholders populated dynamically from the collected metadata. This approach allows automatic updates whenever the source code changes, with minimal manual intervention. Furthermore, unresolved or ambiguous fields (e.g., risk classification) are flagged for manual input. The documentation generation process is fully integrated into the CI/CD pipeline and is triggered automatically upon each pull request to the repository. The flow, shown in Figure 1, consists of five main stages:

- **Pull Request**: When a developer opens a pull request on GitHub, a workflow configured via GitHub Actions is executed. This ensures that documentation is updated synchronously with code modifications.
- **Data Analysis**: The system extracts relevant metadata from the repository. This includes static code analysis, dependency extraction from configuration files, and querying the Python Package Index (PyPI) to retrieve licensing, versioning, and maintainer information.
- **Documents Generation**: The extracted metadata is passed to the templating component to generate structured documents. This stage produces two regulatory artefacts: the Software List and the SOUP List. Any unresolved or unverifiable fields are flagged for expert review.
- **Store Documents**: The generated documentation files are stored in a dedicated directory within the repository. These documents, along with auxiliary files such as historical software state snapshots and error reports, are committed and pushed to the pull request branch.
- **Final Output**: The result is a version-controlled, machine-generated set of documentation files, ready for inclusion in the technical file required by regulatory audits and certification procedures.

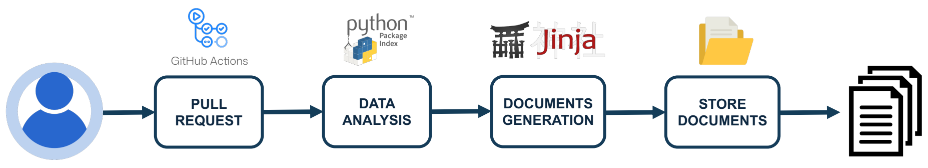

Figure 1: Overview of the automated documentation workflow.

3.5. Validation on a Real-World Project

We tested the system on CT-COVID,[3] a public Python-based SaMD project for COVID-19 detection using CT scans. The repository follows a clear structure, defines its dependencies explicitly, and includes a CI/CD setup, making it a suitable case study for our evaluation. The documentation tool was integrated into the project using GitHub Actions. Each pull request triggered automatic generation of the required documentation artefacts in Markdown format. The output captured detailed metadata for over 30 external packages, including names, versions, licenses, and maintainer information. To evaluate the results, we compared the generated documentation with the information available in the repository. The output demonstrated strong consistency, with most fields being filled in automatically. Manual input was only needed for specific elements, such as risk classification or component rationale, which naturally require expert judgment. This validation shows that the system can reliably extract and organize regulatory metadata with minimal manual effort and can be adapted to support other documents and use cases.

4. Conclusion and Future Work

This work presented an approach to automating regulatory documentation for AI-based medical software by embedding document generation into MLOps workflows. Implemented with widely

[2]https://jinja.palletsprojects.com/en
[3]https://github.com/se4ai2122-cs-uniba/CT-COVID

adopted tools such as Git, PyPI, and Jinja2, the system dynamically produces MDR-aligned documents that remain consistent with code changes and version-controlled within the repository. Validation on a real-world SaMD project demonstrated the ability to extract key metadata—such as dependencies, version history, and authorship—and populate documentation templates with high completeness. At the same time, expert oversight remains necessary for fields such as clinical validation. These results highlight the potential to reduce the manual burden of compliance while ensuring reproducibility and audit readiness. Although currently limited to Python-based projects, the approach can be extended to other document types and compliance frameworks. Overall, this study represents a first step towards integrating regulatory documentation into continuous MLOps pipelines, bridging the gap between rapid AI innovation and the stringent demands of medical software certification.

Acknowledgments

This research was co-funded by the Complementary National Plan PNC-I.1 (“DARE”, PNC0000002, CUP: B53C22006420001), the NRRP Initiative - Next Generation EU (“FAIR”, PE00000013, CUP: H97G22000210007), and (“SERICS”, PE0000014, CUP: H93C22000620001).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT to check grammar and spelling.

References

[1] A. Sengupta, F. N. Ismail, Software as a medical device: Design and compliance, in: 2025 17th International Conference on Computer and Automation Engineering (ICCAE), 2025, pp. 321–325.

[2] L. Svempe, Regulation and Its Impact on Innovation in Healthcare: SAMD Case, SOCRATES Rīga Stradiņš University Faculty of Law Electronic Scientific Journal of Law 1 (2022) 43–52.

[3] I. Y. Chen, E. Pierson, S. Rose, S. Joshi, K. Ferryman, M. Ghassemi, Ethical Machine Learning in Healthcare, Annual Review of Biomedical Data Science 4 (2021) 123–144.

[4] D. Kreuzberger, N. Kühl, S. Hirschl, Machine Learning Operations (MLOps): Overview, Definition, and Architecture, IEEE Access 11 (2023) 31866–31879.

[5] S. Garg, P. Pundir, G. Rathee, P. Gupta, S. Garg, S. Ahlawat, On continuous integration / continuous delivery for automated deployment of machine learning models using mlops, in: 2021 IEEE Fourth Int. Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2021, pp. 25–28.

[6] T. Granlund, V. Stirbu, T. Mikkonen, Towards regulatory-compliant mlops: Oravizio’s journey from a machine learning experiment to a deployed certified medical product, SN Computer Science 2 (2021) 342. doi:10.1007/s42979-021-00726-1.

[7] R. Hermon, P. Williams, V. McCauley, Software as a Medical Device (SaMD): Useful or Useless Term?, in: 54th Hawaii Int. Conference on System Sciences, HICSS, ScholarSpace, 2021, pp. 1–10.

[8] N. M. Jadhav, R. S. Shendge, Iso 13485:2016 – The Gateway of Global or Regional Harmonization for Medical Device Regulations, in: Int. J. of Pharmaceutical Quality Assurance, 2024, pp. 502–511.

[9] K. Yang, Risk Management in Medical Devices: An application of ISO 14971, in: 2024 IEEE Int. Symposium on Product Compliance Engineering (ISPCE), 2024, pp. 1–3.

[10] T. Laukkarinen, K. Kuusinen, T. Mikkonen, Devops in Regulated Software Development: Case Medical Devices, in: 2017 IEEE/ACM 39th Int. Conf. on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER), 2017, pp. 15–18.

[11] T. Lueddemann, H. Bebert, J. Schiebl, T. C. Lueth, Dynamic generation of technical documentation for medical devices, in: Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO), 2014, pp. 2043–2048.

[12] G. Mallardi, L. Quaranta, F. Calefato, F. Lanubile, Towards ensuring responsible ai for medical device certification, in: 2025 IEEE/ACM International Workshop on Responsible AI Engineering (RAIE), 2025, pp. 29–32. doi:10.1109/RAIE66699.2025.00009.