RegComply: A Knowledge Graph Pipeline for Compliance Gap Analysis Against EU Regulations

Thierry Noundou Njike[1], Abiola Paterne Chokki[1] and Xavier Devroey[2]

[1]CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication), Charleroi, Belgium
[2]NADI, University of Namur, Namur, Belgium

Abstract

Software producing organizations in the European Union (EU) must show compliance with regulations whose obligations are scattered across long, frequently updated legal texts, and for which manual assessment rarely traces a decision back to its source article. We present RegComply, a pipeline that extracts obligations from a regulation's articles and annexes into a knowledge graph built on the EU's own European Legislation Identifier (ELI) ontology, retrieves the obligations applicable to a use case with traceability to their source, generates verification requirements of four complementary types, and produces a prioritized compliance gap report. An expert feedback loop turns each correction's justification into a reusable rule. In an initial internal evaluation on DMWay, a CETIC middleware product, against the Cyber Resilience Act (CRA), two engineers judged most extracted obligations and justifications correct, and feedback reduced correction effort in a second round.

Keywords

regulatory compliance, knowledge graph, retrieval-augmented generation, requirements traceability, Cyber Resilience Act

1. Introduction

Software producing organizations must increasingly show compliance with European regulations. The Cyber Resilience Act (CRA) is a recent example. It sets mandatory cybersecurity requirements for every product with digital elements placed on the EU market, from design to end of life [1]. The CRA spans 71 articles and 8 annexes, with obligations for secure design, vulnerability handling, and incident reporting. Reporting duties have applied since 11 September 2026 and the remaining obligations apply from 11 December 2027, with fines of up to 15 million euros or 2.5% of worldwide annual turnover [1]. Compliance officers and engineers currently read such texts manually and map them to product or process requirements by hand. This approach does not scale. It is slow, hard to repeat, and hard to justify to an auditor. For one product, our case study required checking 35 candidate obligations and 108 verification requirements (Section 4).

A compliance decision needs a clear justification: an assessor should be able to trace it back to the exact obligation that motivated it. Both the regulation and the software product it applies to change over time, so an assessment done once becomes outdated and must be run again after every release or amendment. An approach that keeps requirements linked to their regulatory source, and that can be re-run when either the regulation or the product changes, would reduce this burden.

This paper presents RegComply, a pipeline that turns a regulation into a knowledge graph and uses that graph to support compliance verification. The pipeline has three steps. First, it builds a knowledge graph from a regulation by extracting the obligations in its articles and annexes. Second, it retrieves the obligations applicable to a use case and derives verification requirements for each one, that is, concrete checks such as a test case, a review, an inspection, or a walkthrough, that let an engineer confirm whether the obligation is met. Third, it produces a compliance gap report and a prioritized remediation plan, keeping every recommendation traceable to a specific obligation and its source article or annex.

BENEVOL 2026: 25th Belgium-Netherlands Software Evolution Workshop, November 26–27, 2026, Madrid, Spain
thierry.noundounjike@cetic.be (T. N. Njike); abiola.chokki@cetic.be (A. P. Chokki); xavier.devroey@unamur.be (X. Devroey)
0009-0007-6369-2955 (T. N. Njike); 0000-0003-4500-2141 (A. P. Chokki); 0000-0002-0831-7606 (X. Devroey)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

This paper makes three contributions: a method to build a knowledge graph of a regulation's obligations on the EU's own ELI ontology, so the pipeline is designed to apply across EU regulations (evaluated here on the CRA); a method that generates verification requirements of four complementary types per obligation, with a feedback loop that turns an expert's correction into a reusable rule; and a compliance gap report whose remediation plan is prioritized directly from the graph rather than from a manually maintained control library. Following Design Science Research [2], we design and evaluate this pipeline through an industrial case study: DMWay, a CETIC middleware product, assessed against the CRA by two engineers.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the RegComply pipeline. Section 4 reports the evaluation. Section 5 concludes and outlines future work.

2. Background and Related Work

Automated compliance gap analysis. Guo et al. present ComplianceNLP, an end-to-end system that ingests regulatory documents (SEC, MiFID II, Basel III) into a knowledge graph and vector store and reports compliance gaps against institutional policies [3], the closest work to RegComply: both build a graph from regulatory text and produce a gap report from it. Saha and Shukla propose PROPARAG, which retrieves policy evidence per control and assesses coverage against NIST SP 800-53 [4]. Similar coverage-assessment systems target NIST 800-53, CSA-CCM, and CIS 20 controls [5], or represent regulatory information as an eventic graph for retrieval-augmented checking [6]. All start from a catalog or framework that is already curated or pre-decomposed into controls; none build their graph from a regulation's own prose articles and annexes the way RegComply does, and none turn an expert's correction into a rule that improves later assessments.

Extracting obligations from regulatory text. Breaux and Antón extracted rights and obligations from the HIPAA Privacy Rule [7]; Abualhaija et al. survey how regulations can be turned into machine-analyzable representations and how compliance verification can be automated [8]; and Xue et al. used large language models to auto-formalize regulatory requirements into compliance test cases [9]. To our knowledge, none of these studies derives a product-specific gap report from a regulation's own text; RegComply carries extracted obligations through retrieval, verification requirement generation across four types [10, 11], and gap analysis.

Legal knowledge graphs. The EU publishes legislation using ELI, an ontology giving legal acts and their parts a stable identifier and relationships [12]. Garza et al. built PrivComp-KG, a knowledge graph that stores privacy policies together with the regulations they must satisfy [13]. RegComply reuses a small subset of ELI terms (Section 3) rather than a bespoke schema, so its graph model is designed to apply across EU regulations rather than one domain.

Gaps and contributions. The literature above leaves three gaps. First, gap-analysis systems target pre-decomposed control catalogs or a bespoke schema for specific frameworks; none ground their ontology in the EU's own publishing structure to generalize across EU regulations, which RegComply addresses by building on ELI. Second, none turn an expert's correction into a rule that improves later retrieval or requirement generation, which RegComply addresses with its feedback loop. Third, these systems report gaps per obligation or control rather than a sequenced plan; RegComply additionally derives a prioritized remediation order directly from the traceable graph, by deadline and number of open requirements, rather than a flat list or a manually maintained control library.

RegComply does not check whether a piece of evidence satisfies a requirement once implemented. This is a distinct problem, studied for example by Amaral et al. for GDPR data processing agreements [14], which we leave to future work (Section 5).

3. The RegComply Pipeline

RegComply processes a regulation and a use case in three steps: knowledge graph construction, knowledge graph exploitation, and compliance gap reporting. Figure 1 gives an overview of the three

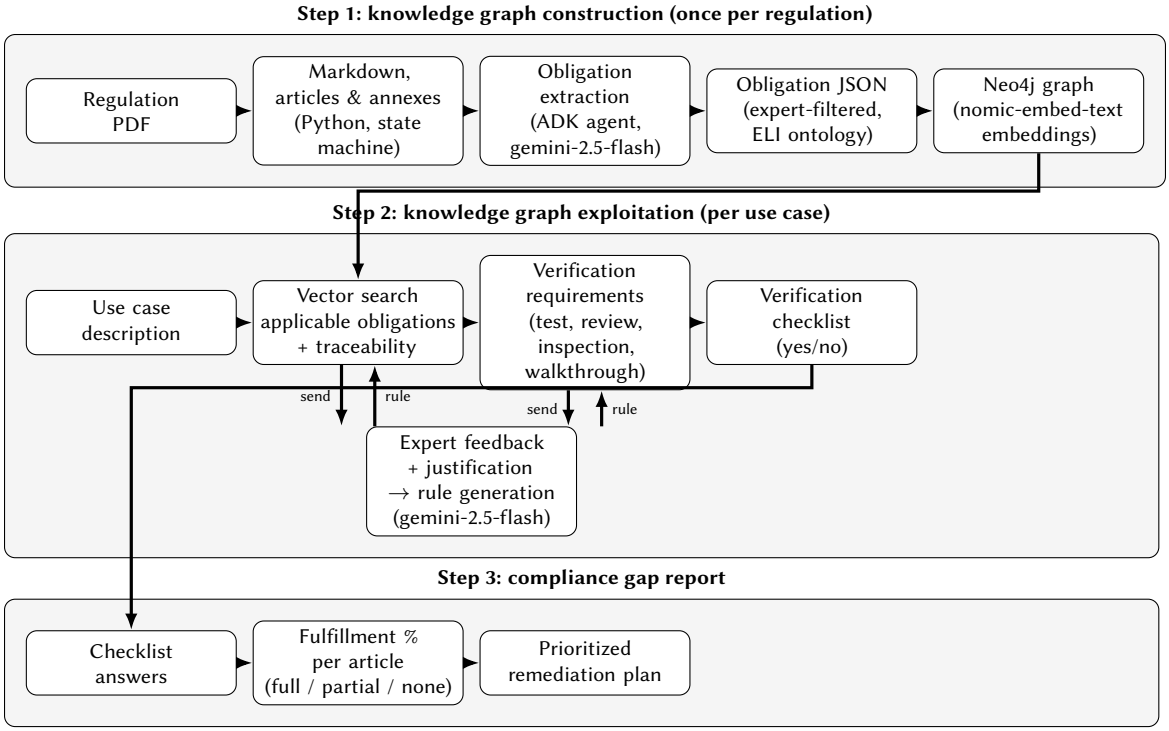


Figure 1: The RegComply pipeline: knowledge graph construction (Step 1), knowledge graph exploitation on a use case with a two-way expert feedback loop (Step 2), and compliance gap reporting (Step 3). In Step 2, obligations and requirements are sent to the expert for review ("send"), and the resulting rule flows back into retrieval and requirement generation ("rule").

steps and the artifacts they produce; the technology used at each step is noted in the figure and detailed below. Implementation details, including the document structuring script, the ontology schema, the Cypher queries used for retrieval, prompts, model configuration, generated JSON, and evaluation data, are available in the project repository.[1]

Step 1: Knowledge Graph Construction. This step runs once per regulation. It has three stages.

Document structuring. A custom Python script converts the official PDF of a regulation into Markdown. The script uses a state machine to recognize the hierarchy of the text: chapters, articles, annexes, paragraphs, and points. Its implementation (`cra_parser.py`) has eight states: preamble, recitals, transition, chapter, article and annex headings, and article and annex bodies. Line patterns such as "Article N" or "ANNEX" followed by a Roman numeral trigger the transitions, and PDF headers, footers and footnotes are dropped. This preserves the numbering and nesting of the original text, which a generic PDF-to-text conversion typically loses, and it identifies the articles and annexes that make up the regulation. This deterministic step is kept separate from the LLM-based stages that follow, so the legal structure does not depend on model behavior. The parser is specific to the layout of the CRA, the GDPR and similar EU regulations (chapters, articles and annexes), so a regulation with another layout would need new line patterns.

Obligation extraction and ontology mapping. Articles and annexes are not themselves the atomic unit RegComply reasons over: each typically bundles several distinct obligations. A Large Language Model (LLM) agent, built with Google's Agent Development Kit (ADK) and backed by gemini-2.5-flash, reads each article and annex and extracts its obligations. Each article and each annex is converted into a JSON record holding the nodes and relationships of the obligations it contains. A domain expert then reviews this JSON once per regulation and removes obligations that are incorrect or not relevant, so only the obligations judged important and correct are kept. The ontology is a simplified version of ELI [12], adjusted to keep only the elements needed for compliance retrieval. A Regulation has

[1] https://github.com/cetic/Wings-RegCompl

Articles and Annexes; each Article or Annex has the Obligations it contains; and each Obligation has a description, a number, and, when applicable, a deadline. Concretely, articles, annexes and obligations are instances of `eli:LegalResourceSubdivision` linked by `eli:is_part_of`, an obligation has an `eli:description` and an `eli:number`, and we add `hasDeadline`, which ELI does not provide. Each obligation node also stores a vector embedding of its text, computed with the nomic-embed-text embedding model [15], to support semantic search in Step 2.

Ingestion. The nodes and relationships are ingested into a Neo4j graph database [16]. The result is a queryable representation of the regulation that keeps the legal structure and supports both graph traversal and vector search.

Step 2: Knowledge Graph Exploitation. This step applies the knowledge graph to a specific use case. It starts from a short description of the use case: the product or process under assessment and its relevant characteristics.

Applicable obligation retrieval. The use case description is embedded with nomic-embed-text and compared against obligation embeddings through vector search. Every retrieved obligation is linked back to its node in the graph, which lets RegComply justify each selection with the exact obligation text and its source article or annex. This traceability distinguishes RegComply from approaches that return a similarity score without a link back to the source.

Verification requirement generation. For each applicable obligation, RegComply proposes a list of verification requirements. An LLM agent, built with ADK and backed by gemini-2.5-flash, reads the obligation and, based on the use case, proposes verifiable requirements. Each requirement is tagged with one of four verification types [10, 11]: test case, review, inspection, or walkthrough. An obligation may need several requirements of different types.

User feedback. A domain expert reviews the retrieved obligations and the proposed requirements, and can add, update, or remove a suggested obligation, giving a short justification for the action. gemini-2.5-flash uses this justification to derive a rule for future assessments of the same regulation and use case. As shown in Figure 1, the loop is bidirectional: it draws on and feeds back into both retrieval and requirement generation, reducing repeated mistakes across assessments.

Verification checklist. All verification requirements for a use case are grouped into a checklist. An engineer answers yes or no for each item and can attach a supporting evidence file. RegComply stores this evidence but does not check whether it supports the answer; verifying evidence correctness is future work (Section 5).

Step 3: Compliance Gap Report. The answers collected in Step 2 feed a compliance gap report. The report gives the percentage of verification requirements fulfilled, and lists the requirements that remain open. Each article is classified as fully, partially, or not yet compliant, based on the state of the requirements attached to its obligations; this fulfillment rate reflects requirement completion, not a legal determination of compliance.

The report also prioritizes the remaining requirements. Two criteria drive this order. The first is the deadline attached to the obligation in the regulation, when one exists. The second is the number of requirements still needed to close an obligation. A requirement linked to an obligation with two requirements already implemented out of three is prioritized over a requirement linked to an obligation with none implemented out of three, because fewer requirements remain to complete it. This heuristic directs effort where it closes the largest number of open obligations first. The count treats all requirements as equally costly, which is a simplification.

Illustrative Example. Article 14 of the CRA requires manufacturers to report an actively exploited vulnerability to ENISA within 24 hours of becoming aware of it. RegComply extracts this as an obligation with a description, a number, and a 24-hour deadline, linked to Article 14. Assuming DMWay, the CETIC IoT middleware used in our case study, is assessed in the role of manufacturer of a product with digital elements, vector search retrieves this obligation as applicable, citing Article 14 as justification. RegComply then proposes a review requirement (confirm that an ENISA reporting step exists in the incident response procedure) and a test-case requirement (simulate a vulnerability report and confirm it reaches ENISA within 24 hours). If the engineer marks the review requirement as implemented but the test case as not yet implemented, the obligation is classified as partially compliant, and it is prioritized

ahead of obligations with no deadline and more remaining requirements.

4. Evaluation

We evaluated RegComply on DMWay, a CETIC middleware product that maps and manages data exchanges between heterogeneous Internet of Things (IoT) devices and remote data management systems. DMWay processes data over a network connection, which places it in scope of the CRA. We processed the full CRA text, 71 articles and 8 annexes, from which RegComply extracted 59 obligation instances, and used the resulting graph to assess DMWay. The first two authors, both CETIC engineers familiar with DMWay's architecture, reviewed RegComply's output against the official CRA text and their own domain judgment, following the demonstration and evaluation activities of the pilot case study. The review covered only RegComply's output and involved no legal or compliance expert, so recall was not assessed. The evaluation checks five aspects of the pipeline.

Correct extracted obligations (88%, 52/59 obligations). RegComply's LLM agent extracted 59 obligation instances from the CRA. Comparing each one against the official text of its source article or annex, the engineers judged 52 to be genuine, distinct obligations and 7 not to be. This is a precision measure, since missed obligations cannot appear when only the 59 extracted instances are reviewed. The expert review step in Step 1 removes such extra instances before they reach the graph.

Applicability and justification correctness (91%, 32/35 obligations). Among the 52 correctly extracted obligations, RegComply retrieved 35 as applicable to DMWay. For each, we checked whether it is genuinely applicable and whether the cited passage correctly supports that, since an obligation that is not applicable cannot have a correct justification. 32 of the 35 were judged correct. Applicable obligations that RegComply did not retrieve were not assessed.

Verification requirement generation correctness (90%, 97/108 requirements). For each of the 35 retrieved obligations, RegComply proposed one or more verification requirements across the four types, 108 in total. We checked whether each proposed requirement was an adequate check for its obligation; 97 were judged correct and 11 were not.

Feedback integration (11/108 → 3/108 corrections). The 11 requirements flagged as incorrect above are the corrections the engineers made in a first assessment round. We checked whether feedback from that round reduces the corrections needed on the same 108 requirements in a second round. Feedback cut corrections from 11 to 3. Both rounds used the same product version and use case, so this shows whether the agent repeats its mistakes, not whether corrections transfer to a changed product.

Prioritization consistency (5/5). We checked whether the five most prioritized requirements in the gap report either have a deadline or, once implemented, would fully validate their article or annex, consistent with the heuristic defined in Section 3.

5. Conclusion and Future Work

This paper presented RegComply, a knowledge graph pipeline for compliance verification against European regulations. It extracts a regulation's obligations into a graph built on a simplified version of the EU's own ELI ontology, retrieves obligations applicable to a use case, generates verification requirements of four complementary types, and turns an expert's correction into a reusable rule. It produces a compliance gap report prioritized directly from the graph. Two engineers evaluated this design on DMWay, a CETIC product, against the CRA, comparing RegComply's extracted obligations and justifications to their own assessment. In this initial internal assessment, 52/59 extracted obligations and 32/35 retrieved obligations with their justifications were judged correct, 97 of the 108 generated verification requirements were judged adequate, and feedback reduced the corrections needed in a second round. Because the graph is built once per regulation, a new product release needs only a new use case description and a new run of Steps 2 and 3, although we have not yet tested this on a changed product.

Limitations. The evaluation covers one product and one regulation, assessed by two engineers from the team that built the pipeline, without a legal or compliance expert and without an independently built reference set to check for missed obligations, so recall is unknown. The second feedback round reused the same product version and 108 requirements, so part of the improvement may reflect familiarity. The prioritization check confirms the top-ranked items follow the defined heuristic, not that ordering by deadline and number of open requirements is more effective than alternative prioritization strategies. Also, updating the graph after a regulatory change currently requires rebuilding it rather than an incremental update. The evaluation used a single hosted model, gemini-2.5-flash. The code supports open-weight models, but they were not run. RegComply was also not compared with a single-prompt LLM baseline, so its advantage over one remains to be shown.

Future work. We plan to compare RegComply with an independent manual assessment by compliance and legal experts (time, missed obligations), a single-prompt LLM baseline, and an ontology-free variant. We also plan to validate RegComply on other products and regulations with independent assessors, including changed product versions, extend it to reason over multiple regulations at once and flag overlapping obligations, update the graph incrementally as a regulation or product changes, read the Formex XML published by the Publications Office [17], run open-weight models, verify attached evidence, and explore severity-based prioritization heuristics.

Acknowledgments

This research was supported by WINGS - Phase 4 Project (Grant Agreement No. 9114).

Declaration on Generative AI

During the preparation of this work, the authors used Claude in order to: grammar and spelling check, paraphrase and reword. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] European Parliament and Council of the European Union, Regulation (EU) 2024/2847 of 23 October 2024 on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) No 168/2013 and (EU) 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act), Official Journal of the European Union, OJ L, 2024/2847, 20.11.2024, 2024. http://data.europa.eu/eli/reg/2024/2847/oj.

[2] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Quarterly 28 (2004) 75–105.

[3] D. Guo, J. Wu, S. M. Yiu, ComplianceNLP: Knowledge-graph-augmented RAG for multi-framework regulatory gap detection, 2026. ArXiv:2604.23585.

[4] B. Saha, S. K. Shukla, An automated framework for cybersecurity policy compliance assessment against security control standards, 2026. ArXiv:2605.07515.

[5] R. Bar-Haim, L. Eden, Y. Kantor, V. Agarwal, M. Devereux, N. Gupta, A. Kumar, M. Orbach, M. Zan, Towards automated assessment of organizational cybersecurity posture in cloud, in: Proceedings of the 6th Joint International Conference on Data Science & Management of Data (CODS-COMAD 2023), 2023, pp. 167–175. doi:10.1145/3570991.3571008.

[6] J. Sun, Z. Luo, Y. Li, A compliance checking framework based on retrieval augmented generation, in: Proceedings of the 31st International Conference on Computational Linguistics (COLING 2025), 2025, pp. 2603–2615. https://aclanthology.org/2025.coling-main.178/.

[7] T. D. Breaux, A. I. Antón, Analyzing regulatory rules for privacy and security requirements, IEEE Transactions on Software Engineering 34 (2008) 5–20. doi:10.1109/TSE.2007.70746.

[8] S. Abualhaija, M. Ceci, L. Briand, Legal requirements analysis, 2023. ArXiv:2311.13871.

[9] Z. Xue, X. Chen, M. Zhang, Explicating tacit regulatory knowledge from LLMs to auto-formalize requirements for compliance test case generation, 2026. ArXiv:2601.09762.
[10] IEEE, IEEE Standard for System, Software, and Hardware Verification and Validation, 2017. IEEE Std 1012-2016.
[11] IEEE, IEEE Standard for Software Reviews and Audits, 2008. IEEE Std 1028-2008.
[12] Publications Office of the European Union, ELI Ontology, version 1.5, 2024. https://data.europa.eu/eli/ontology.
[13] L. Garza, L. Elluri, A. Kotal, A. Piplai, D. Gupta, A. Joshi, PrivComp-KG: Leveraging knowledge graph and large language models for privacy policy compliance verification, 2024. ArXiv:2404.19744.
[14] O. Amaral, M. I. Azeem, S. Abualhaija, L. C. Briand, NLP-based automated compliance checking of data processing agreements against GDPR, IEEE Transactions on Software Engineering 49 (2023) 4282–4303. doi:10.1109/TSE.2023.3288901.
[15] Z. Nussbaum, J. X. Morris, B. Duderstadt, A. Mulyar, Nomic embed: Training a reproducible long context text embedder, 2024. ArXiv:2402.01613, accepted to Transactions on Machine Learning Research.
[16] J. Webber, A programmatic introduction to Neo4j, in: Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH '12), 2012, pp. 217–218. doi:10.1145/2384716.2384777.
[17] Publications Office of the European Union, Formex: Formalized Exchange of Electronic Publications, n.d. https://op.europa.eu/en/web/eu-vocabularies/formex.