

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITY OF MODENA AND REGGIO EMILIA

"Enzo Ferrari" Department of Engineering

Master's Degree in Artificial Intelligence Engineering (LM-32)

Leveraging Large Language Models and Retrieval-Augmented Generation for Automated Policy Comparison in Legal Compliance

Supervisor:
Prof. Simone Calderara
Co-supervisor:
Prof. Filippo Miti
Prof. Davide Chiuchiù

Candidate:
Federico Lo Presti
Student ID 207442

ACADEMIC YEAR 2025/2026

Abstract

Leveraging Large Language Models and Retrieval-Augmented Generation for Automated Policy Comparison in Legal Compliance

The increasing volume and complexity of regulatory frameworks require enterprises to continuously align internal policies with evolving external legal obligations. This process is traditionally manual, costly, and difficult to scale, particularly under tight regulatory deadlines and in multi-jurisdictional contexts such as the European Union.

This thesis investigates the use of Retrieval-Augmented Generation (RAG) to support the automation of regulatory compliance analysis. The problem is structured into three core tasks: extracting obligations from regulatory texts, detecting conflicts between these obligations and internal corporate policies, and generating harmonised policy revisions. The framework is extended with gap analysis to identify missing coverage.

To address these tasks, a modular RAG-based pipeline is designed, integrating semantic retrieval, large language model reasoning, and structured prompt design. Multiple architectural variants are proposed to explore trade-offs between cost, accuracy, and scalability, including obligation-first retrieval strategies and Natural Language Inference (NLI)-based filtering.

The system is evaluated on Italian corporate policies and EU regulatory documents, highlighting challenges related to legal language ambiguity and retrieval limitations. Results show that structuring the process around explicit obligations stabilises comparison cost under increased document fragmentation, improves verdict traceability, and enables missing-obligation analysis while preserving conflict-detection performance on the controlled test set.

Overall, the thesis demonstrates how RAG-based systems can effectively support key compliance activities, reducing manual effort while enabling controlled cost–accuracy trade-offs.

Keywords: Regulatory Compliance, Retrieval-Augmented Generation, Legal NLP, Conflict Detection, Obligation Extraction, Policy Harmonisation

Contents

Abstract ii

1 Introduction 1

1.1 Context and Motivation . . . 1

1.2 Problem Statement . . . 2

1.3 Research Objectives and Contributions . . . 3

1.4 Thesis Outline . . . 5

2 Background and State of the Art 6

2.1 Natural Language Processing for Legal Texts . . . 6

2.1.1 Structural Characteristics . . . 6

2.1.2 Document Segmentation Strategies . . . 7

2.2 Large Language Models . . . 9

2.2.1 Scaling Laws and In-Context Learning . . . 9

2.2.2 Instruction Tuning and Alignment . . . 9

2.2.3 Structured Output Generation . . . 9

2.3 Retrieval-Augmented Generation . . . 10

2.3.1 Motivation . . . 10

2.3.2 Architecture . . . 10

2.3.3 From Naive to Advanced and Modular RAG . . . 11

2.3.4 Embedding Models . . . 13

2.3.5 Vector Databases . . . 13

2.3.6 Hybrid Retrieval . . . 14

2.3.7 Failure Modes of RAG Systems . . . 14

2.4 Natural Language Inference and Cross-Encoders . . . 17

2.4.1 Bi-Encoders vs. Cross-Encoders . . . 18

2.4.2 Cross-Encoder NLI for Contradiction Detection . . . 18

2.4.3 Confidence Calibration and Threshold Tuning . . . 18
2.5 Agentic AI Architectures . . . 19
2.5.1 From Prompt Chains to Autonomous Agents . . . 19
2.5.2 Tool Use and Function Calling . . . 20
2.5.3 Challenges and Limitations . . . 20
2.6 Related Work . . . 21
2.6.1 Regulatory NLP Systems . . . 21
2.6.2 RAG for Legal and Compliance Applications . . . 21
2.6.3 Conflict Detection Between Normative Texts . . . 21
2.6.4 Agentic Approaches to Document Processing . . . 22
2.6.5 Positioning of This Work . . . 22

3 System Architecture **23**
3.1 Design Goals and Architectural Drivers . . . 23
3.2 High-Level Design and Data Flow . . . 24
3.2.1 Layered View . . . 24
3.2.2 The `Architecture` Coordinator . . . 24
3.2.3 Data Flow . . . 26
3.2.4 Concurrency and Execution Model . . . 27
3.3 Domain Model . . . 27
3.4 Document Ingestion and Preprocessing . . . 28
3.4.1 Normalisation . . . 29
3.4.2 Structure-Aware Segmentation . . . 29
3.4.3 Length Normalisation . . . 30
3.4.4 Asymmetric Treatment of the Two Collections . . . 31
3.5 Vector Store . . . 32
3.5.1 Embedding Space . . . 32
3.5.2 Index and Persistence . . . 32
3.6 Retrieval Layer . . . 33
3.6.1 Dense, Sparse and Hybrid Retrieval . . . 33

3.6.2 Fusion Strategy . . . 34
3.6.3 Cross-Collection Top-k Retrieval . . . 35
3.6.4 Transposing the Index Map . . . 36
3.6.5 Cross-Encoder NLI Filtering . . . 37
3.7 LLM Integration . . . 38
3.7.1 Provider Abstraction and Interaction Modes . . . 39
3.7.2 Output Contracts and Their Enforcement . . . 39
3.7.3 Cost and Token Accounting . . . 40
3.8 Prompt Design . . . 41
3.8.1 Prompts as External Artefacts . . . 41
3.8.2 Prompt Contracts . . . 41
3.8.3 Constraining the Model . . . 43
3.9 Extension Points and Summary . . . 44

4 Pipeline Architectures: A Comparative Design 45
4.1 Architecture Overview: Common Stages . . . 45
4.2 Tunable Parameters and Their Motivation . . . 46
4.3 BASE: Full Chunk-to-Chunk Comparison . . . 47
4.4 OBLIGATION_SIM: Obligation-First Retrieval . . . 48
4.4.1 Why Obligations Are the Retrieval Unit . . . 49
4.5 OBLIGATION_SIM_MIS: Missing-Obligation Detection . . . 51
4.6 SMALL_STRONG: CrossEncoder NLI Pre-Filtering . . . 52
4.7 Trade-off Analysis . . . 53
4.7.1 Axes of Divergence . . . 53
4.7.2 Article-level vs. Obligation-level Comparison . . . 54
4.7.3 Filtering: CrossEncoder vs. No Pre-filter vs. ID Tracking . . . 54
4.7.4 Retrieval Depth and Coverage . . . 55
4.7.5 Cost and Scalability . . . 55
4.7.6 Traceability and Downstream Use . . . 56
4.7.7 Summary of Expected Trade-offs . . . 56

5 Agentic Rewrite Pipeline **58**

5.1 Motivation . . . 58

5.2 EU-Style Amendment Processing . . . 59

5.3 Agent Decision Loop . . . 59

5.3.1 Phase 1: Decision / Parse . . . 60

5.3.2 Phase 2: Retrieve and Apply . . . 60

5.3.3 Output model . . . 61

5.3.4 Contrast with the simple pipeline . . . 61

5.4 Article Lookup and Output Parsing . . . 62

5.4.1 Base document preparation . . . 62

5.4.2 Multi-strategy matching . . . 62

5.4.3 Apply prompt constraints . . . 62

5.4.4 Error handling . . . 63

6 Implementation Details **64**

6.1 Technology Stack and Project Structure . . . 64

6.1.1 Core dependencies . . . 64

6.1.2 Project structure . . . 64

6.2 Configuration and Environment Management . . . 65

6.2.1 Centralised settings . . . 65

6.2.2 Environment variables . . . 65

6.3 Streamlit Frontend . . . 66

6.3.1 Sidebar: model and document selection . . . 66

6.3.2 Main panel: analysis and results . . . 66

6.3.3 Cost and performance dashboard . . . 67

6.3.4 Styling . . . 68

6.4 Telemetry and Observability . . . 69

6.4.1 Architecture . . . 69

6.4.2 CSV schema . . . 69

6.4.3 Public API . . . 69

6.4.4 Metrics collected per pipeline run 69
6.4.5 File management 70

7 Experimental Evaluation **73**
7.1 Dataset Description 73
7.2 Evaluation Protocol and Metrics 74
7.3 Quantitative Results 76
7.3.1 LLM Backend: GPT-5.3 vs. GPT-4o-mini 76
7.3.2 Comparison of the Unfiltered Pipeline Variants 77
7.3.3 Document Granularity: the Stressed Setup 77
7.4 Qualitative Analysis 78
7.4.1 Worked Example: from Obligations to Verdict 78
7.4.2 Verdict Quality Across Conflict Categories 82
7.5 Ablation Studies 82

8 Discussion **84**
8.1 Key Findings 84
8.2 Limitations 86
8.3 Ethical and Legal Considerations 87

9 Conclusions and Future Work **91**
9.1 Summary of Contributions 91
9.2 Future Directions 92

References **94**

List of Figures

2.1 Example of the hierarchical structure of a regulatory document. 7

2.2 Comparison of structure-aware, fixed-size, and hybrid chunking strategies applied to the same sample legal article. 8

2.3 Hybrid retrieval architecture combining vector search and BM25 keyword search through rank-based fusion. 15

3.1 High-level architecture of the RAG-based compliance comparison system. 26

4.1 Obligation-based pipeline structure. The obligation variants add an LLM extraction stage before retrieval; the extracted obligations replace article chunks as retrieval and comparison units, while OBLIGATION_SIM_MIS additionally derives unresolved obligations from the comparison output. 49

4.2 Structure of the SMALL_STRONG variant. Retrieved external–internal pairs pass through a local NLI CrossEncoder, and only pairs classified as contradictions are forwarded to the LLM comparison stage. The diagram shows the intended filtering path; its quantitative cost–recall effect is not evaluated in this thesis. 52

6.1 Main Streamlit interface for selecting the pipeline and policy documents, starting the conflict analysis, and inspecting the resulting summary and extracted obligations. 67

6.2 Detail view of a detected conflict, showing the model-generated rationale and the expandable source passages used to inspect the external and internal evidence. . . . 68

6.3 Cost and performance dashboard reporting total execution time, token consumption, and cumulative estimated cost for the completed analysis. 68

List of Code

3.1 Whitespace normalisation for PDF pages, taken from `retriever.py` 29
3.2 The `Article` Pydantic model, taken from `models.py` 29
3.3 Hybrid ensemble retriever (vector + BM25), taken from `retriever.py` 34
3.4 Cross-Encoder NLI filter keeping only contradictions 38
3.5 Italian-language prompt used for obligation extraction. 42

1. Introduction

1.1 Context and Motivation

The regulatory landscape faced by modern enterprises is characterised by an ever-increasing volume and complexity of normative instruments. Organisations operating across multiple jurisdictions must continuously align their internal policies with a heterogeneous body of external regulations that undergo frequent revisions [1]. The European Union alone produces a large volume of legislative acts each year, including regulations, directives, and decisions, many of which introduce new obligations or amend existing ones that cascade into corporate governance frameworks. For instance, recent data indicates that over 2,300 legal acts were adopted in 2025 alone, highlighting the continuous evolution and complexity of the European regulatory landscape [43].

Traditionally, the task of regulatory alignment is performed manually by legal and compliance teams. Domain experts must read newly published regulations, identify the obligations they contain, retrieve relevant sections of existing internal policies, assess whether conflicts or gaps exist, and finally draft harmonised text that reconciles corporate operational needs with legal requirements. A single amending regulation may affect dozens of internal policy articles, and manually matching its provisions to internal policies makes comprehensive coverage impractical within tight regulatory deadlines.

Recent advances in Natural Language Processing (NLP), particularly through Retrieval-Augmented Generation (RAG), provide a promising pathway for automating regulatory compliance workflows. RAG architectures combine the parametric knowledge of large language models (LLMs) with non-parametric external memory accessed via retrieval mechanisms, enabling the integration of up-to-date and domain-specific information during inference [33, 20].

This hybrid approach addresses key limitations of standalone LLMs, such as hallucinations and static knowledge, by grounding generated outputs in retrieved evidence while preserving strong reasoning and generation capabilities [20, 68]. When augmented with vector databases and embedding-based retrieval, RAG pipelines enable semantic search over large corpora and support context-aware reasoning.

As a result, these systems can approximate several core cognitive steps traditionally carried out by human compliance analysts, including document retrieval, contextual interpretation, and structured information extraction [26].

This thesis is motivated by the opportunity to bridge the gap between state-of-the-art AI capabilities and the operational needs of corporate compliance functions, proposing a system[1] that automates conflict detection, obligation extraction, and policy harmonisation through multiple architectural variants with explicit cost-accuracy trade-offs.

1.2 Problem Statement

Given a corpus of internal corporate policies $\mathcal{P}_{int} = \{p_1, p_2, \ldots, p_n\}$ stored in a vector database, and a newly published external regulatory document d_{ext}, the central problem addressed by this thesis can be decomposed into three sub-problems:

1. **Obligation Extraction.** Automatically identify and classify the legally binding obligations contained in d_{ext}, distinguishing between *regulatory requirements* (general rules or inderogable limits) and *organisational obligations* (specific measures, procedures, or structures that an internal policy must implement).

   Example. Consider the provision: "Personal data shall be processed lawfully, fairly, and transparently." This constitutes a *regulatory requirement*, as it defines a general principle. By contrast, a clause such as "The company must implement a procedure for handling data subject access requests within 30 days" represents an *organisational obligation*, as it prescribes a concrete internal measure.

2. **Conflict Detection.** For each extracted obligation $o_i \in d_{ext}$[2], retrieve a subset $\mathcal{S}_i \subseteq \mathcal{P}_{int}$ of the most semantically relevant policy fragments, and determine whether the internal policy text is *consistent* with, or *in conflict* with, the external obligation. A conflict arises when the internal policy, considered within its self-contained context, subordinates, restricts, or contradicts the rights, protections, or duties mandated by the regulation.

[1] Evaluated on Italian-language corporate policies and EU-sourced regulatory documents, a language pair that amplifies retrieval difficulty given the relative scarcity of Italian legal text in standard pre-training corpora.

3. **Policy Harmonisation.** When a conflict is detected, generate a revised policy text that reconciles the regulatory requirement with the operational intent of the original internal provision, preserving legal conformity without disrupting organisational coherence.

A secondary problem extends this framework to *gap analysis*: identifying obligations in d_{ext} that are not addressed by *any* segment of $\mathcal{P}_{int}$, i.e., missing coverage rather than active contradiction.

The problem is non-trivial for several reasons [7]. First, legal language is inherently ambiguous and context-dependent; the same provision may be conflicting or compatible depending on the broader normative framework. Second, the absence of explicit treatment in an internal policy does not necessarily constitute a conflict, as the relevant discipline may reside in other parts of the policy not retrieved by the system. Third, the combinatorial space of obligation-to-policy pairings grows multiplicatively with document size, demanding architectures that control computational cost (LLM token consumption) while maintaining detection recall.

1.3 Research Objectives and Contributions

This thesis aims to advance the automation of regulatory compliance verification through Retrieval-Augmented Generation (RAG) techniques, and makes the following contributions:

1. **Design and implementation of a modular RAG-based compliance pipeline.** The thesis introduces a complete architecture for regulatory compliance verification, integrating document ingestion, chunking, embedding, hybrid retrieval, LLM-based comparison, and harmonised text generation [33, 20].

2. **Development and comparative analysis of multiple architectural variants.** Four pipeline configurations are designed to explore trade-offs between token cost, retrieval quality, and conflict-detection accuracy:

[2] An *obligation* is a normative unit that expresses a duty, requirement, or prohibition imposed on a subject by a regulatory provision. As detailed in sub-problem 1 above, two subtypes are distinguished: *regulatory requirements* (general rules or inderogable limits) and *organisational obligations* (specific measures or procedures that an internal policy must implement). A fuller treatment of obligation classification in the legal NLP literature is provided in Section 2.1.

- BASE: direct chunk-to-chunk comparison;
- OBLIGATION_SIM: an obligation-first architecture reducing dependence on the size of $\mathcal{P}_{int}$;
- OBLIGATION_SIM_MIS: an extension supporting missing-obligation detection (gap analysis);
- SMALL_STRONG: a CrossEncoder-filtered architecture using NLI as a low-cost pre-filter [4, 65, 58].

3. **Introduction of an obligation-first retrieval strategy.** The proposed OBLIGATION_SIM architecture formalises obligation extraction as an intermediate reasoning layer, shifting comparison from raw chunks to structured normative units and improving scalability.

4. **NLI-based filtering for efficient contradiction detection.** The SMALL_STRONG variant integrates a CrossEncoder NLI model (`nli-MiniLM2-L6-H768`) to pre-filter candidate conflicts. It is designed to reduce LLM calls, with the resulting cost–recall trade-off left for controlled quantitative validation.

5. **Extension to missing-obligation analysis.** The OBLIGATION_SIM_MIS variant identifies regulatory obligations not addressed by internal policies, enabling gap analysis beyond explicit conflict detection.

6. **Evaluation framework under realistic constraints.** The three unfiltered pipeline variants are evaluated across accuracy, latency, and cost dimensions, varying the model backend and document granularity. Due to the lack of public benchmarks for Italian corporate compliance, evaluation relies on a manually annotated ground truth. The NLI pre-filter is described architecturally but is not included in the controlled quantitative comparison.

7. **Agentic pipeline for amendment-document heterogeneity.** The thesis implements an architectural extension toward automated policy rewriting, introducing a decision–retrieval–application loop whose role is to absorb the format heterogeneity of EU-style amendment instructions, including references and operation phrasings that a regex-based parser cannot reliably handle. The agentic loop is a coverage mechanism, not a cost or latency optimisation, and the quality of its generated amendments is outside the scope of the present evaluation [67].

8. **Prompt design methodology for legal compliance tasks.** The system defines structured prompt contracts for obligation extraction, comparison, verdict unification, and harmonisation, with explicit output schemas.

9. **Telemetry and observability framework.** A reproducible logging system tracks token usage, latency, and cost, enabling systematic comparison across architectures.

1.4 Thesis Outline

The remainder of this thesis is organised as follows.

Chapter 2 reviews the theoretical foundations and related work in legal NLP, Transformer-based language models, Retrieval-Augmented Generation, vector databases, Natural Language Inference, and agentic AI systems.

Chapter 3 presents the overall system architecture, including document ingestion, chunking, vector storage, hybrid retrieval, LLM integration, and prompt design.

Chapter 4 describes the four proposed pipeline variants, analyses their computational characteristics, and discusses the trade-offs they introduce.

Chapter 5 introduces the agentic rewrite pipeline for automated policy amendment and details its decision, retrieval, and text-application stages.

Chapter 6 reports the main implementation choices, including the software stack, project organisation, configuration management, frontend design, and telemetry subsystem.

Chapter 7 presents the experimental methodology and results, combining quantitative metrics with qualitative case studies across architectures and model configurations.

Chapter 8 discusses the main findings, the limitations of the proposed approach, and the broader implications of AI-assisted legal compliance analysis.

Chapter 9 concludes the thesis by summarising the contributions and outlining directions for future work.

2. Background and State of the Art

This chapter presents the theoretical foundations and technological landscape underpinning the proposed system. We begin with the specific challenges of applying Natural Language Processing to legal texts, then introduce Large Language Models and Retrieval-Augmented Generation as core building blocks. We subsequently discuss Natural Language Inference via Cross-Encoders and the emerging paradigm of Agentic AI. Finally, we survey related work at the intersection of regulatory compliance and AI-assisted document analysis.

2.1 Natural Language Processing for Legal Texts

Legal language presents distinct challenges for Natural Language Processing (NLP) systems. Unlike general-domain text, regulatory and legislative documents exhibit rigid structural organisation, domain-specific terminology, long and syntactically complex sentences, and frequent cross-references [2]. These characteristics make tasks such as segmentation, retrieval, and semantic comparison significantly more difficult than in standard NLP settings [13, 8].

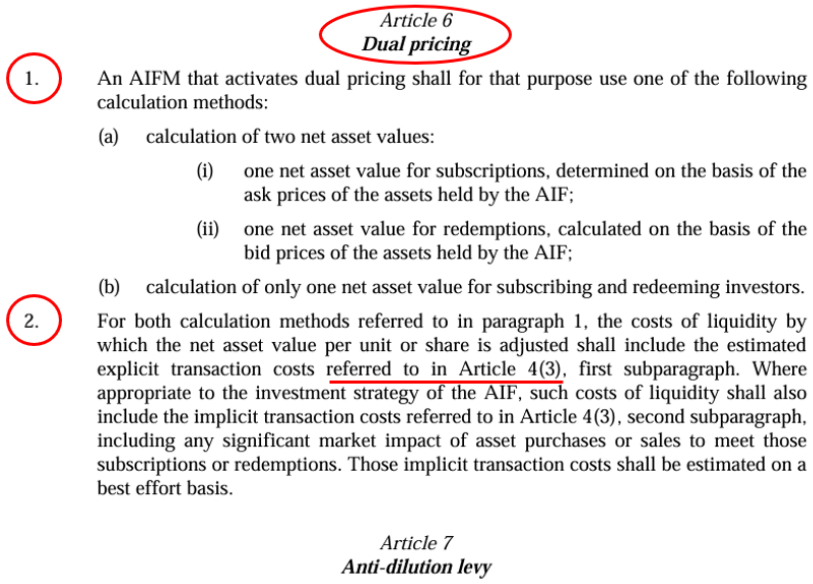
A critical aspect is the ambiguity and interpretive flexibility of legal texts [56]. Normative statements are often phrased to allow flexibility in application, requiring systems not only to extract information but also to preserve subtle distinctions between obligations, permissions, and prohibitions. Consequently, downstream tasks such as compliance checking demand both structural awareness and careful semantic interpretation.

2.1.1 Structural Characteristics

Legal documents follow hierarchical structures composed of titles, chapters, articles, and paragraphs [2]. Each level carries semantic meaning and contributes to the interpretation of the text. In addition, legal provisions frequently include explicit references to other parts of the document or external sources (e.g., “pursuant to Article 12(3)”), creating dependencies that cannot be captured through flat text representations alone.

These properties imply that preprocessing steps must preserve the logical organisation of the

document as much as possible. Flattening the structure into arbitrary text segments risks losing essential contextual information, negatively affecting both retrieval and reasoning stages.



Article 6
Dual pricing

1. An AIFM that activates dual pricing shall for that purpose use one of the following calculation methods:

(a) calculation of two net asset values:

(i) one net asset value for subscriptions, determined on the basis of the ask prices of the assets held by the AIF;

(ii) one net asset value for redemptions, calculated on the basis of the bid prices of the assets held by the AIF;

(b) calculation of only one net asset value for subscribing and redeeming investors.

2. For both calculation methods referred to in paragraph 1, the costs of liquidity by which the net asset value per unit or share is adjusted shall include the estimated explicit transaction costs referred to in Article 4(3), first subparagraph. Where appropriate to the investment strategy of the AIF, such costs of liquidity shall also include the implicit transaction costs referred to in Article 4(3), second subparagraph, including any significant market impact of asset purchases or sales to meet those subscriptions or redemptions. Those implicit transaction costs shall be estimated on a best effort basis.

Article 7
Anti-dilution levy

Figure 2.1: Example of the hierarchical structure of a regulatory document.

2.1.2 Document Segmentation Strategies

Document segmentation is a fundamental step in legal NLP pipelines, particularly in Retrieval-Augmented Generation (RAG) systems. Since LLMs operate within a bounded context window, long documents must be divided into smaller units while preserving semantic coherence [20].

Several segmentation strategies are commonly adopted:

- **Structure-aware segmentation**: Splits the document based on explicit legal markers such as article or paragraph boundaries. This approach preserves the normative structure and produces semantically meaningful units aligned with legal interpretation.
- **Fixed-size chunking**: Divides text into segments of predefined length (e.g., fixed number of characters or tokens) with optional overlap. While simple and efficient, this approach may break logical units and reduce interpretability.

- **Hybrid approaches**: Combine structural segmentation with length constraints to balance semantic coherence and computational limits. For instance, long articles may be recursively split, while short fragments are merged to avoid loss of context.

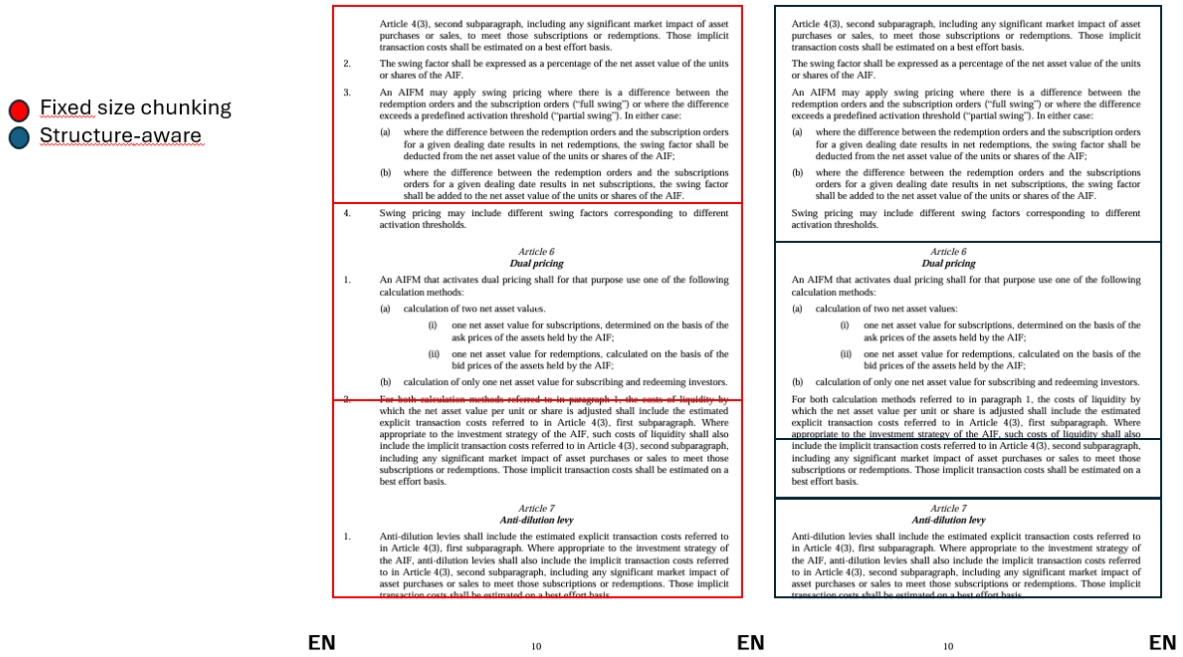


Figure 2.2: Comparison of structure-aware, fixed-size, and hybrid chunking strategies applied to the same sample legal article.

In practice, segmentation must also address edge cases such as very short or very long sections. Short fragments may lack sufficient context for reliable interpretation and are often merged with adjacent segments, while overly long passages can exceed model limits and are therefore split into smaller overlapping chunks.

The choice of segmentation strategy has a direct impact on system performance [29]. Poor segmentation can lead to incomplete retrieval, diluted context, and ultimately degraded model accuracy. For this reason, segmentation is not merely a preprocessing step but a critical design component of legal RAG pipelines.

This work adopts a hybrid, structure-aware segmentation strategy tailored to regulatory documents, as described in Chapter 3. In particular, article-level splitting is combined with adaptive merging and recursive subdivision to ensure that each segment remains both semantically meaningful and computationally tractable.

2.2 Large Language Models

Large Language Models (LLMs) are transformer-based neural networks pre-trained on vast text corpora using self-supervised objectives, primarily next-token prediction [59, 5].

2.2.1 Scaling Laws and In-Context Learning

Research has demonstrated predictable scaling relationships between model size, dataset size, compute budget, and downstream performance [25]. Larger models exhibit increasingly capable *in-context learning*—the ability to perform new tasks by conditioning on examples provided in the prompt without parameter updates [5].

Techniques such as chain-of-thought prompting [63] further enhance reasoning capabilities by instructing models to produce intermediate reasoning steps. This is particularly relevant for legal analysis, where multi-step logical reasoning over normative statements is required.

2.2.2 Instruction Tuning and Alignment

Raw pre-trained models are further refined through:

1. **Supervised Fine-Tuning (SFT)**: Training on curated instruction-response pairs to improve instruction-following behaviour.

2. **Reinforcement Learning from Human Feedback (RLHF)**: Aligning outputs with human preferences via reward modelling, reducing harmful or unhelpful generations [42].

These alignment steps yield models capable of following complex, multi-step instructions—a prerequisite for legal document analysis tasks where prompts encode domain-specific comparison criteria and output format constraints.

2.2.3 Structured Output Generation

Modern LLMs support constrained decoding modes that guarantee outputs conform to a specified schema (e.g., JSON Schema). This capability is critical for systems that programmatically consume LLM outputs: rather than parsing free-form text with fragile heuristics, the model is constrained to

emit valid structured data (e.g., Pydantic-compatible JSON objects with typed fields). This ensures downstream reliability in production pipelines.

2.3 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) combines parametric knowledge (stored in model weights) with non-parametric knowledge (retrieved from an external corpus) to produce grounded, up-to-date responses [33].

2.3.1 Motivation

LLMs suffer from knowledge cutoff dates, hallucination (generating plausible but unsupported claims), and inability to cite sources reliably [68]. RAG addresses these limitations by conditioning generation on retrieved evidence, reducing hallucination rates and enabling verifiable attribution to source documents [52]. For regulatory compliance applications, grounding is particularly important: every claim about a policy's content must be traceable to the actual text.

2.3.2 Architecture

A standard RAG pipeline comprises three stages:

1. **Indexing**: Documents are segmented into chunks, encoded into dense vector representations via an embedding model, and stored in a vector database alongside metadata.

2. **Retrieval**: Given a query, the system retrieves the top-k most similar chunks using approximate nearest-neighbor (ANN) search over the embedding space.

3. **Generation**: Retrieved chunks are injected into the LLM prompt as context, and the model generates a response grounded in the provided evidence.

The quality of a RAG system depends on the alignment between the retrieval and generation stages: retrieved context must be relevant, sufficient, and not so voluminous as to exceed the model's effective context window or dilute attention over critical passages.

2.3.3 From Naive to Advanced and Modular RAG

RAG should not be understood as a single fixed architecture, but as a family of designs that differ in how they prepare queries, select evidence, construct context, and coordinate generation. The literature commonly distinguishes between naive, advanced, and modular RAG systems [20]. This progression reflects an increasing degree of control over the information flow between the external corpus and the language model, rather than a change in the fundamental objective introduced by Lewis et al. [33]: supplementing parametric knowledge with evidence retrieved at inference time.

Naive RAG follows a linear retrieve-then-generate sequence. Documents are divided into chunks and embedded during indexing; at query time, the query is encoded in the same vector space, the top-k nearest chunks are selected, and their text is inserted directly into the generation prompt. This design is simple, computationally efficient, and suitable for establishing a baseline. However, it generally relies on a single representation of the user's query, a fixed retrieval depth, and limited validation of the returned evidence. A broad or ambiguous query may therefore retrieve passages that are topically related but insufficient to answer the actual question. Likewise, the generator receives the selected context without an intermediate mechanism for checking whether the passages are mutually consistent, complete, or authoritative.

Advanced RAG introduces optimisations before, during, and after retrieval. Pre-retrieval techniques improve either the indexed corpus or the query. Corpus-side operations include structure-aware segmentation, metadata enrichment, removal of duplicate material, and adaptation of the embedding model to the target domain. Query-side operations include rewriting, expansion, and decomposition. Query rewriting converts a broad or conversational request into a formulation that better matches the language of the indexed documents. Query expansion introduces synonyms, related expressions, or alternative formulations to increase recall, while query decomposition separates a complex request into smaller searches whose results can later be combined. These operations are valuable in legal retrieval, where the same normative concept may be expressed through different terminology. They must nevertheless preserve distinctions such as permission, obligation, and prohibition, since an expansion that changes deontic force may improve lexical coverage while corrupting the legal meaning of the query.

Retrieval-time optimisation focuses on candidate selection and ordering. A common pattern

uses a fast bi-encoder or lexical retriever to produce a broad candidate set, followed by a more precise reranker that evaluates each query-document pair jointly. The first stage protects scalability, while the second improves the position of passages that are genuinely relevant but difficult to distinguish through vector similarity alone. Hybrid retrieval provides another form of optimisation by combining semantic and lexical evidence. Metadata filters can further restrict the search to a jurisdiction, document type, effective date, or organisational unit. These mechanisms expose an explicit precision-recall trade-off: a larger candidate pool reduces the probability of excluding relevant evidence, but also increases processing cost and the amount of potentially distracting material passed downstream.

Post-retrieval processing determines how the selected evidence is transformed into model context. Retrieved passages may be deduplicated, reordered, grouped by source, filtered by a relevance threshold, or compressed to retain only the sentences needed for the task. This stage is important because the nominal context-window size of an LLM is not a reason to include every available passage. Excessive context consumes tokens and can weaken the prominence of decisive evidence. In compliance applications, context construction must also retain provenance metadata so that a generated conclusion can be traced to the document, article, and fragment from which its supporting evidence originated.

Iterative RAG extends the process by allowing retrieval to occur more than once. An initial search may reveal a definition, exception, or cross-reference that motivates a second, more specific query. The system can therefore alternate retrieval and reasoning until it has collected sufficient evidence or reached a stopping condition. **Adaptive RAG** makes this behaviour conditional: it may decide whether retrieval is required, which source should be queried, how many candidates should be requested, or whether the available evidence is insufficient. These strategies are well suited to interconnected regulatory documents, but they introduce additional latency and new control decisions whose errors can propagate through subsequent iterations [20].

Modular RAG generalises these developments by treating query processing, retrieval, reranking, context construction, generation, and verification as replaceable modules. Instead of enforcing one linear path, a modular system can route different requests through specialised retrievers or validation stages. For example, exact article references may be directed towards lexical search, conceptual obligations towards dense retrieval, and amendment instructions towards a source-specific

lookup. Modularity also permits individual components to be evaluated or replaced without retraining the entire system, which is particularly useful when regulations, embedding models, or language-model providers change independently.

The architectures proposed in this thesis occupy different positions within this taxonomy. BASE is closest to the naive retrieve-then-generate pattern, although structure-aware chunking and hybrid retrieval already add advanced elements. OBLIGATION_SIM performs query decomposition and transformation before retrieval by converting regulatory articles into atomic obligations. OBLIGATION_SIM_MIS adds post-comparison coverage tracking through obligation identifiers, while SMALL_STRONG inserts an NLI-based filtering module between retrieval and LLM reasoning. The agentic rewrite pipeline adds a conditional decision about whether a source passage must be retrieved before an amendment can be applied. Taken together, the variants illustrate how a common RAG foundation can be progressively specialised through preprocessing, filtering, routing, and state tracking rather than through changes to the language model alone.

2.3.4 Embedding Models

Sentence-transformers [46] produce fixed-size vector representations of text passages by applying mean pooling over token embeddings from a pre-trained transformer. Models such as `all-MiniLM-L6-v2` provide a strong quality-efficiency trade-off: 384-dimensional embeddings and 22 million parameters offer competitive performance on semantic textual similarity benchmarks at relatively low computational cost.

The choice of embedding model directly impacts retrieval quality. Domain-specific models trained on legal corpora can improve recall for specialised terminology, though general-purpose models remain competitive when combined with appropriate chunking and retrieval strategies.

2.3.5 Vector Databases

Vector databases provide persistent storage and efficient similarity search over high-dimensional embeddings. Systems such as ChromaDB [12], Pinecone, FAISS, and Weaviate offer trade-offs between deployment complexity, scalability, and feature richness. Key capabilities include:

- Approximate nearest-neighbor search (HNSW, IVF indices)

- Metadata filtering (restricting search to subsets of the corpus)
- Incremental updates (adding or removing documents without full re-indexing)

2.3.6 Hybrid Retrieval

Pure dense retrieval can miss lexically relevant but semantically distant matches—the *vocabulary mismatch* problem. Hybrid retrieval addresses this by combining:

- **Sparse retrieval** (BM25): A term-frequency-based ranking function effective for exact keyword matching and rare terms [47].
- **Dense retrieval**: Embedding-based similarity capturing semantic meaning regardless of surface form.

Figure 2.3 illustrates the parallel retrieval paths and their convergence into a single fused ranking. The two result lists remain independent until the fusion stage, allowing semantic matches from the dense retriever and exact lexical matches from BM25 to contribute to the final ordering.

Ensemble approaches such as reciprocal rank fusion (RRF) merge ranked lists from both retrievers:

$$\mathrm{RRF}(d) = \sum_{r \in R} \frac{1}{k + r(d)}$$

where R is the set of rankers, $r(d)$ is the rank of document d in ranker r, and k is a smoothing constant. This achieves superior recall compared to either method alone [38].

2.3.7 Failure Modes of RAG Systems

Retrieval augmentation changes the conditions under which an LLM produces an answer, but it does not remove the possibility of error. Instead, it creates a multi-stage system in which failures may originate during document preparation, query construction, retrieval, context assembly, or generation. The final output alone does not necessarily reveal which component was responsible. This distinction is important in compliance analysis because a plausible verdict can conceal the omission of a decisive provision, while an incorrect verdict may be caused by missing evidence rather than deficient reasoning [20, 68].

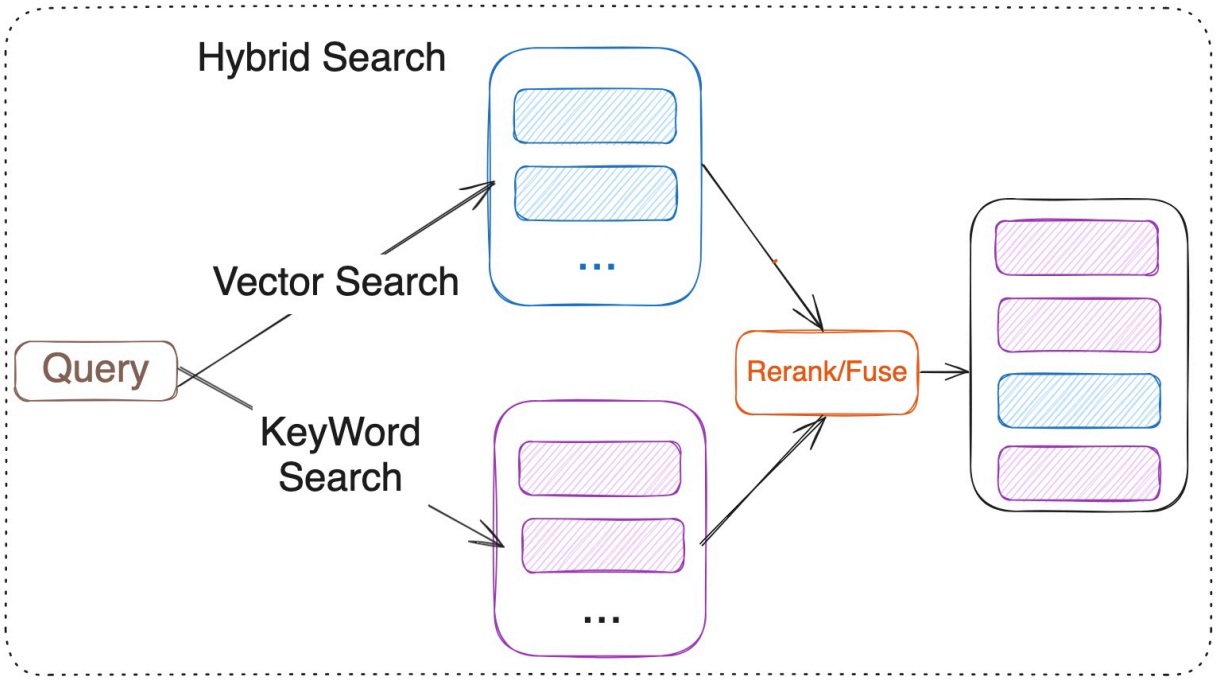


Figure 2.3: Hybrid retrieval architecture combining vector search and BM25 keyword search through rank-based fusion.

A **retrieval failure** occurs when the evidence required for a correct conclusion is not included in the candidate set. The cause may precede the search itself. A relevant clause can be lost or fragmented during document ingestion, an obligation can be omitted during extraction, or a query can use terminology that is poorly aligned with the indexed corpus. The embedding model may also assign low similarity to legally related passages because they differ lexically or because their relationship depends on roles, exceptions, or deontic expressions that are weakly represented in a general-purpose vector space. Approximate nearest-neighbour search and a finite top-k cut-off introduce further possibilities for exclusion. Once the decisive passage is absent from the retrieved set, even a capable generator has no grounded basis on which to incorporate it. In conflict detection, this commonly produces a false negative rather than a visibly malformed answer.

A related but distinct **ranking failure** occurs when the relevant passage is retrieved by one component but receives a position below the final cut-off, or is demoted during fusion with another ranked list. This distinction matters because increasing k may repair a cut-off problem without improving the underlying representation. However, it is not a free solution. A larger k increases

downstream token consumption and may introduce passages that share vocabulary with the query while referring to a different legal subject, time period, or organisational process. Ranking quality must therefore be considered together with retrieval depth and the capacity of subsequent filtering or reranking stages.

Even when relevant evidence is present, **context dilution** can impair generation. If many loosely related passages are concatenated into the prompt, the decisive clause competes for attention with repetitive or contradictory material. This problem is particularly serious when the corpus contains several versions of the same policy or regulation. A semantically relevant but obsolete provision may appear convincing unless version and effective-date metadata are preserved. Duplicate chunks can also give disproportionate prominence to one interpretation simply because it appears several times. Context construction must consequently balance coverage against selectivity, retaining enough surrounding text for interpretation without treating the context window as an unlimited evidence container.

A **chunk-boundary failure** arises when segmentation separates information that must be interpreted jointly. A paragraph may state a general duty while the following paragraph defines an exception, beneficiary, deadline, or scope condition. Retrieving only one side produces an incomplete normative unit even if the selected chunk is individually relevant. Overlap can reduce this risk, but excessive overlap increases index size, duplicate retrieval, and token usage. Structure-aware segmentation provides a stronger defence by respecting article and paragraph boundaries, although it cannot fully resolve cross-references or requirements distributed across distant provisions. The stressed evaluation of Chapter 7 demonstrates this failure mode directly: a conflict divided across chunk boundaries is missed by both article-level and obligation-level variants.

A **generation failure** occurs after adequate evidence has reached the model. The LLM may ignore a retrieved exception, merge statements from different sources, attribute a requirement to the wrong article, or infer a prohibition that the source does not establish. It may also overgeneralise from a narrow provision or treat silence in an internal policy as an explicit contradiction. Structured output constrains the form of the answer, but a syntactically valid verdict can still be semantically incorrect. Retrieval grounding has been shown to reduce hallucination, yet it cannot guarantee that the model interprets or cites the retrieved evidence faithfully [52, 68].

For this reason, **grounding is not equivalent to correctness**. A response is grounded when it

is connected to retrieved material, but the material itself may be irrelevant, incomplete, outdated, or lower in legal authority than another source. Furthermore, a claim may cite a passage that shares its topic without logically supporting its conclusion. In a regulatory setting, correctness also depends on jurisdiction, temporal validity, hierarchy of norms, and the relationship between general rules and specific exceptions. These properties cannot be inferred from semantic similarity alone and require metadata, explicit reasoning instructions, and ultimately human legal review.

Failures can also **propagate across stages**. If obligation extraction omits a requirement, retrieval never searches for it. If extraction produces an over-broad statement, the retriever returns diffuse evidence, which then increases ambiguity in the comparison prompt. A ranking error can place an unrelated internal clause in context, and the generator may convert that accidental similarity into a confident conflict verdict. Finally, an aggregation stage can merge distinct verdicts or incorrectly classify an unresolved obligation as missing coverage. The apparent reliability of the final answer is therefore bounded by the weakest upstream stage.

Mitigation requires stage-level observability rather than reliance on the final verdict alone. Useful controls include preserving source and version metadata, logging extracted queries and ranked candidates, comparing dense and lexical retrieval results, applying reranking or relevance thresholds, limiting duplicate context, and allowing the model to report insufficient evidence instead of forcing a binary conclusion. The present system addresses part of this risk through structure-aware segmentation, hybrid retrieval, obligation identifiers, structured verdicts, and telemetry. Nevertheless, its end-to-end evaluation does not independently measure extraction recall or retrieval recall. These remain explicit limitations and motivate the retrieval-specific evaluation directions discussed in Chapters 8 and 9.

2.4 Natural Language Inference and Cross-Encoders

Natural Language Inference (NLI), also known as Recognizing Textual Entailment (RTE), is the task of determining the logical relationship between a premise and a hypothesis. Given a pair of texts, the model classifies their relationship as one of three categories [4]:

- **Entailment**: The hypothesis is logically implied by the premise.

- **Contradiction**: The hypothesis is incompatible with the premise.
- **Neutral**: The hypothesis is neither implied nor contradicted by the premise.

2.4.1 Bi-Encoders vs. Cross-Encoders

Two architectural paradigms exist for computing text-pair relationships:

- **Bi-encoders** encode each text independently into a fixed-dimensional vector. Similarity is computed via cosine distance, enabling fast retrieval through ANN indices. However, independent encoding prevents fine-grained token-level interactions between the two texts, limiting accuracy on tasks requiring precise semantic comparison.
- **Cross-encoders** concatenate both texts as a single input sequence (separated by a special token) and apply full self-attention across the pair. This produces richer interaction features at the cost of $O(n^2)$ complexity over the combined sequence length, making them unsuitable for large-scale retrieval but ideal as re-rankers or filters [46].

2.4.2 Cross-Encoder NLI for Contradiction Detection

In regulatory compliance, the contradiction label is of primary interest: it signals that an internal policy statement conflicts with an external regulatory requirement. Cross-encoder models fine-tuned on NLI datasets (SNLI [4], MultiNLI [65]) can serve as efficient pre-filters—retaining only text pairs classified as contradictions before invoking more expensive LLM-based analysis.

This three-stage approach (retrieve → filter → compare) exploits the complementary strengths of each component: bi-encoders for fast candidate retrieval, cross-encoders for precise semantic filtering, and LLMs for nuanced verdict generation with natural-language explanations.

2.4.3 Confidence Calibration and Threshold Tuning

Cross-encoder outputs are softmax-normalised logits over the three NLI classes:

$$P(\text{class} \mid x_1, x_2) = \frac{e^{z_{\text{class}}}}{\sum_{c \in \{\text{ent,con,neu}\}} e^{z_c}}$$

Thresholding on the contradiction probability $P(\text{con} \mid x_1, x_2)$ allows tuning the precision-recall trade-off: a lower threshold increases recall (fewer missed conflicts) at the cost of more false positives forwarded to the LLM stage. The optimal threshold depends on the relative cost of missed conflicts versus unnecessary LLM invocations; the present implementation uses a hard class gate, while controlled threshold calibration is left for future evaluation.

2.5 Agentic AI Architectures

Agentic AI refers to systems where one or more LLM-based agents autonomously plan, reason, and execute multi-step workflows by interacting with external tools and data sources [61]. This paradigm extends the capabilities of static prompt chains by introducing dynamic control flow governed by the model's reasoning.

2.5.1 From Prompt Chains to Autonomous Agents

LLM-based systems can be organised along a spectrum of autonomy:

1. **Prompt chaining**: A fixed directed acyclic graph (DAG) of LLM calls with deterministic control flow. Each step's output feeds into the next step's input, but the model makes no routing decisions.

2. **Routing**: The model selects among predefined paths based on input classification, introducing minimal decision-making without iterative reasoning.

3. **ReAct pattern**: Interleaved Reasoning and Acting—the model generates a thought (reasoning trace), selects a tool to invoke, observes the result, and iterates until the task is complete [67].

4. **Plan-and-execute**: The model first generates a complete plan decomposing the task into sub-steps, then executes each step sequentially, revising the plan if intermediate results require adaptation [30].

2.5.2 Tool Use and Function Calling

Agents extend LLM capabilities by invoking external tools (e.g., database queries, API calls, code execution, file operations). The interaction follows a structured protocol:

1. The LLM produces a structured tool-call request (typically JSON conforming to a predefined schema).
2. The runtime validates and executes the tool call.
3. The tool's output is appended to the conversation context.
4. The LLM reasons over the result and decides the next action [49].

This mechanism enables agents to interact with external systems while maintaining the LLM as the central reasoning engine.

2.5.3 Challenges and Limitations

Agentic architectures introduce specific challenges:

- **Error propagation**: Mistakes in early reasoning steps compound through the execution chain.
- **Cost unpredictability**: Dynamic execution paths make token consumption variable and harder to budget.
- **Reliability**: Non-deterministic LLM outputs can lead to inconsistent behaviour across runs.
- **Observability**: Debugging multi-step autonomous workflows requires comprehensive logging of decisions, tool calls, and intermediate states.

These challenges motivate the design of constrained agentic systems with well-defined action spaces, validation steps, and fallback mechanisms, principles applied in our agentic rewrite pipeline (Chapter 5).

2.6 Related Work

The intersection of AI and legal/regulatory compliance has attracted growing research attention. We organise related work along three axes: regulatory NLP systems, RAG for compliance, and conflict detection.

2.6.1 Regulatory NLP Systems

LegalBERT [8] introduced domain-specific pre-training on legal corpora (EU legislation, court cases, contracts), demonstrating consistent improvements over general-purpose models on legal NER, text classification, and question answering tasks. **LexGLUE** [9] provides a benchmark suite for legal NLP, establishing baselines for multi-label classification of EU legislation and enabling standardised comparison across models.

Obligation extraction has been studied in financial regulations [39], with approaches ranging from rule-based deontic logic parsers to fine-tuned sequence labelling models. These works establish that normative statement identification is feasible with current NLP techniques, though accuracy degrades on complex multi-conditional obligations.

2.6.2 RAG for Legal and Compliance Applications

Several works apply RAG to legal question answering. **LegalBench** [21] evaluates LLMs on legal reasoning tasks, revealing that retrieval augmentation improves performance on statute interpretation tasks where precise textual grounding is required.

In the compliance domain, industry solutions (e.g., Thomson Reuters Regulatory Intelligence, Relativity) increasingly embed RAG pipelines to surface relevant regulations given a query or policy draft. However, published academic evaluations of end-to-end compliance RAG systems remain scarce, particularly for cross-policy conflict detection rather than single-document QA.

2.6.3 Conflict Detection Between Normative Texts

More recent work explores LLM-based approaches for regulatory comparison, where large language models are prompted to analyse pairs of policies and produce structured judgements (e.g.,

conflict, alignment, partial overlap) along with supporting evidence. While these approaches show strong empirical performance, they are often evaluated in ad hoc settings rather than standardised benchmarks.

Additionally, they present several limitations, including high computational cost, variability across runs, and the lack of guarantees that all relevant provisions have been systematically compared.

2.6.4 Agentic Approaches to Document Processing

The application of agentic AI to document processing is an emerging area. Works on automated code refactoring [66] and scientific paper writing [34] demonstrate that LLM agents can successfully execute multi-step document transformation tasks when equipped with appropriate tools and constraints. In the legal domain, agentic approaches have been explored for contract drafting and amendment tracking, though published systems remain largely proprietary.

2.6.5 Positioning of This Work

While individual components, such as legal NLP, retrieval-augmented generation (RAG), NLI-based reasoning, and agentic workflows, have been extensively studied in prior work, existing approaches typically focus on a subset of these elements in isolation.

In particular, prior systems often rely either on: (i) direct LLM prompting for policy comparison, which can be computationally expensive and non-deterministic, or (ii) classical NLP techniques, which provide more controllable behaviour but limited reasoning capabilities.

This thesis does not aim to outperform such approaches in isolation. Instead, it explores how these components can be systematically combined within a unified, modular pipeline for regulatory compliance checking.

The main contribution lies in the design and evaluation of multiple pipeline variants, enabling an explicit analysis of trade-offs between cost, accuracy, and coverage. Additionally, an agentic rewrite mechanism is introduced to handle the heterogeneous formats of EU-style amendment documents by absorbing reference and operation-phrasing variability that deterministic parsing cannot cover, rather than to improve cost or latency.

3. System Architecture

This chapter describes the architecture that all pipeline variants share. The presentation is deliberately *structural*: it explains which components exist, what contract each of them exposes and why the boundaries between them were drawn where they are. The variant-specific choices that populate this skeleton are the subject of Chapter 4, while the concrete technology stack, configuration mechanism and telemetry subsystem are described in Chapter 6.

3.1 Design Goals and Architectural Drivers

The system is a Retrieval-Augmented Generation (RAG) pipeline whose task is to compare an external regulatory document with an internal policy knowledge base and to report inconsistencies, contradictions or alignment between the two. Four requirements shaped the architecture, and most of the design decisions discussed in the remainder of this chapter can be traced back to one of them.

Variant pluggability. The central research question of this work is *how* the external document should be represented and *how much* should be filtered before invoking an LLM. The architecture must therefore admit several pipelines that share ingestion, indexing and comparison but differ at two well-defined seams: the unit of retrieval and the filter interposed between retrieval and generation. Everything else is factored into common modules so that a variant is a composition of stages rather than a fork of the codebase.

Cost as a first-class concern. LLM invocation dominates both latency and monetary cost by orders of magnitude with respect to every other stage. The architecture consequently treats the number of LLM calls, and the number of tokens per call, as quantities to be minimised structurally, through batching, transposition of the retrieval relation and cheap local pre-filters, rather than as an afterthought handled by caching.

Traceability. A compliance verdict is only actionable if it can be attributed to a specific normative source. Derived representations therefore keep an explicit back-reference to the artefact they were derived from, and indexed passages carry metadata identifying their position in the

source document, so that a verdict can be resolved back to an article of the external regulation and to a passage of the internal policy.

Containment of stochasticity. The LLM is the only non-deterministic component of an otherwise deterministic pipeline. The architecture confines that stochasticity by prescribing the *shape* of every model response, whether a JSON payload or a fixed set of named fields, so that the surrounding stages consume a predictable structure. As discussed in Section 3.7.2, this contract is stated in the prompt rather than enforced by the API, which bounds how much determinism the architecture can actually claim.

3.2 High-Level Design and Data Flow

3.2.1 Layered View

The system is organised as a sequence of layers, each transforming the output of the previous one into a representation with a strictly narrower scope: from raw bytes, to structured text, to vectors, to candidate pairs, to verdicts. Table 3.1 summarises the decomposition.

Two properties of this decomposition are worth emphasising. First, the layers are connected by *data* contracts rather than by control-flow dependencies: each stage consumes and produces a self-contained value, with typed objects where the data originates in the code and textual structures where it originates in the model. This separation makes it possible to reorder, skip or replace a stage without touching its neighbours. Second, only two layers are marked as variation points. Ingestion, indexing, retrieval and aggregation are identical across all four pipelines of Chapter 4; this is a deliberate constraint, because a comparative study is only meaningful if the variants differ along the axes under investigation and coincide everywhere else.

3.2.2 The `Architecture` Coordinator

At the core of the system lies the `Architecture` class, which plays the role of a *facade* over the layers described above. It owns the resources whose lifetime spans an entire run, including the Azure OpenAI client, the ChromaDB vector store wrapper and the embedding model, as well as the mutable run-level state, namely the accumulated token counts and cost estimates.

Table 3.1: Architectural layers, their contract and their responsibility. A bullet in the last column marks the two *seams*, namely the layers at which the pipeline variants of Chapter 4 are allowed to diverge.

Layer	**Input → Output**	**Responsibility**	**Seam**
Ingestion	PDF → `Article`	Parsing, normalisation, structure-aware segmentation	Not applicable
Representation	`Article` → `Article` or `Obligation`	Optional LLM distillation of the normative content	•
Indexing	Internal text → vectors	Embedding and persistent storage	Not applicable
Retrieval	Query set → index map	Cross-collection top-k retrieval	Not applicable
Filtering	Index map → index map	Pruning of candidate pairs before generation	•
Reasoning	Index map → verdicts	LLM comparison under a prompt-level output contract	Not applicable
Aggregation	Verdicts → report	Unification, gap detection, cost accounting	Not applicable

Centralising these resources is not merely a convenience. Embedding models and database handles are expensive to construct and must be shared across stages that are invoked thousands of times; instantiating them per call would dominate the runtime of the local stages. Cost telemetry, symmetrically, must be observed by every LLM-invoking stage but aggregated once per run, which makes the coordinator the natural place to hold it.

Domain-specific behaviour is not implemented inside the coordinator but delegated to submodules: `llm_invocation`, `document_processing`, `obligation_extraction`, `policy_comparison` and `verdict_handling`, which are bound lazily at method level. The consequence is a class whose surface is a stable vocabulary of pipeline operations, while the implementation of each operation

can evolve, or be substituted by a variant-specific version, independently. Lazy binding additionally keeps the import graph acyclic: submodules refer to the coordinator's state through the instance they receive, not through a module-level import of the coordinator itself.

The overall architecture of the system is illustrated in Figure 3.1, providing a visual representation of the main components and their interactions.

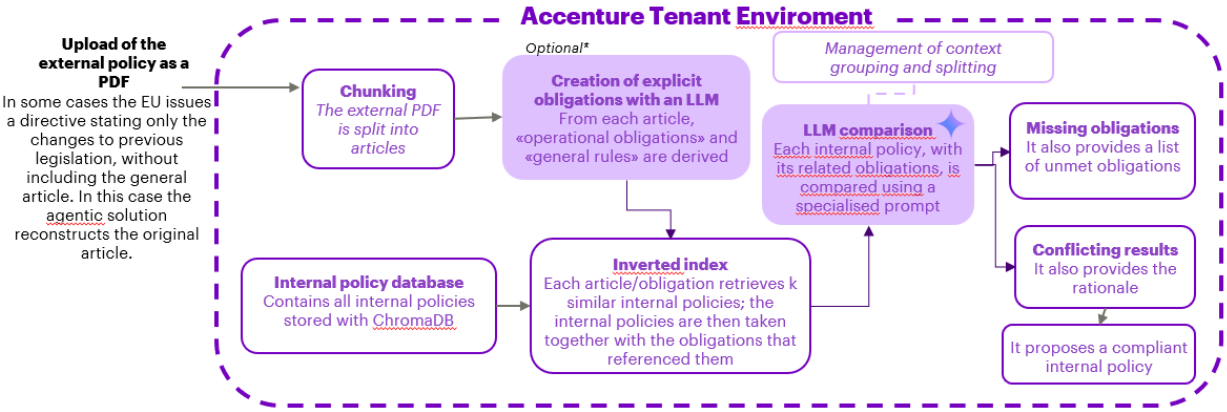


Figure 3.1: High-level architecture of the RAG-based compliance comparison system.

3.2.3 Data Flow

The data flow can be summarised as follows. An external PDF document is loaded and preprocessed into structured textual units, which are optionally transformed into minimal obligations through LLM extraction. These representations are used as queries against a vector database containing internal policy fragments. The retrieved internal passages are paired with the external content and evaluated through LLM-based comparison, which produces a structured verdict together with a possible harmonised policy proposal.

The pipeline is best read as a *funnel*. The Cartesian product $O \times C$ of all external elements and all internal chunks is the theoretical comparison space; retrieval reduces it to $O(nk)$ candidate pairs, the optional filter reduces it further, and transposition finally collapses the surviving pairs into a smaller number of batched LLM calls. Each reduction is applied by the cheapest component capable of applying it, which is the classic *cascade* (or telescoping) arrangement of information-retrieval systems: recall-oriented, inexpensive stages first; precision-oriented, expensive stages last, on a much smaller input.

3.2.4 Concurrency and Execution Model

The workload is overwhelmingly I/O-bound: the wall-clock time of a run is dominated by network round-trips to the LLM endpoint, not by local computation. The system is therefore driven by an `asyncio` event loop, with the extraction and comparison stages issuing their calls concurrently rather than sequentially. Since the provider SDK is used through its synchronous interface, each call is dispatched with `asyncio.to_thread`: the coroutine awaits a worker thread instead of a native awaitable, which yields the same overlapping of network waits without requiring an asynchronous client.

Unbounded concurrency, however, is not an option. Provider endpoints enforce rate limits expressed in requests and tokens per minute, and exceeding them converts latency savings into failures and retries. Parallelism is consequently bounded by an `asyncio.Semaphore`, which acts as a backpressure mechanism: the semaphore caps the number of in-flight requests, so the pipeline self-regulates to the sustainable throughput of the endpoint instead of saturating it. The bound in use was selected empirically and is reported in Section 4.2.

PDF loading is offloaded in the same way, being both blocking and slow. The remaining local components, namely embedding computation and, in SMALL_STRONG, CrossEncoder inference, are executed synchronously on the event loop. This is a deliberate simplification rather than an oversight: these stages run as distinct phases of the pipeline, before the concurrent LLM calls are issued, so blocking the loop there does not delay any pending network operation. It does, however, mean that filtering does not overlap with generation, which is a possible optimisation left open in Chapter 9. Token counts returned by each completed call are accumulated on the coordinator and converted into a cost estimate, so that the run-level totals are assembled incrementally as the pipeline proceeds.

3.3 Domain Model

The stages of the pipeline exchange a small set of typed objects declared with Pydantic [45], alongside plain textual structures such as the index map and the model responses. The typed objects give a name and a shape to the two units on which the whole comparison is defined:

- `Article`: a structural unit of a document, carrying an identifier, its full text and the name of the source document. It is produced by ingestion, entirely from code, and is the unit of retrieval in the article-level variants.

- `Obligation`: an atomic normative statement distilled from an article, carrying a numeric identifier, a type (organisational obligation or general rule), its textual content and a back-reference to the `Article` it originates from. It is the unit of retrieval in the obligation-level variants.

The outcome of a comparison is a *verdict*: a response organised into named fields, produced by the LLM under the format prescribed by its prompt and consumed by the aggregation stage through textual parsing. It is deliberately not listed above, because it never becomes a validated object; the consequences of that choice are examined in Section 3.7.2.

The back-reference carried by `Obligation` deserves emphasis, because it is what reconciles two otherwise conflicting goals. Retrieval and comparison benefit from a compact, denoised representation of the norm; auditing requires the verbatim legal text. Keeping the derived representation *linked to* rather than *substituted for* its source lets the system optimise the former without losing the latter, and is the mechanism through which the traceability requirement of Section 3.1 is met.

It is worth being precise about the role these types play. They document and stabilise the interfaces between stages, and they make the back-reference an explicit field rather than a convention; they do not, however, act as a validation boundary for model output, since the LLM response is decoded and its fields are read before the corresponding objects are populated. The boundary between the two collections is thus typed, while the boundary between the system and the model is not.

3.4 Document Ingestion and Preprocessing

Ingestion converts a PDF, whose only intrinsic structure is the page, into a sequence of units that are meaningful both for a lawyer and for a retriever. This is the first place where the architecture makes a substantive assumption about its domain, and it is worth making that assumption explicit.

3.4.1 Normalisation

The pipeline begins with PDF loading through `PyPDFLoader` [28], which yields one textual unit per page. Page-level text extracted from a PDF is typographically noisy: line breaks reflect the layout of the printed page rather than the syntax of the sentence, and hyphenation, multiple spaces and column artefacts are common.

Normalisation collapses newlines and repeated whitespace into single spaces, as shown in Listing 3.1. The operation is trivial but not cosmetic: a subword tokeniser assigns different token sequences to `"obbligo\n di"` and `"obbligo di"`, so layout noise propagates into the embedding space and, through it, into similarity scores. Normalising before segmentation also prevents layout-induced line breaks from being mistaken for structural boundaries by the regular expressions of the next stage.

Code 3.1: Whitespace normalisation for PDF pages, taken from `retriever.py`

```
return [
  Document(
    page_content=" ".join(d.page_content.replace("\n", " ").split()),
    metadata=d.metadata
  )
  for d in documents
]
```

3.4.2 Structure-Aware Segmentation

External documents are then segmented along legal article boundaries, detected with regular expressions over the canonical article headings of normative texts (e.g. `Article \d+`, `Art. \d+`). Each segment is encapsulated in the `Article` model introduced in Section 3.3.

Code 3.2: The `Article` Pydantic model, taken from `models.py`

```
class Article(BaseModel):
  id: str = Field(..., description="Article identifier, e.g. ART-32")
  content: str = Field(..., description="Full text of the article")
```

```
    source_document: str = Field(..., description="Name of the regulatory document")
```

The alternative, fixed-size sliding windows (the default in most RAG implementations), was rejected for two reasons that are specific to regulatory text. The first is semantic: the article is the unit at which the legislator states a self-contained rule, so an arbitrary window is likely to split a norm from its scope condition, or to merge two unrelated duties into a single vector. The second is referential: legal reasoning and legal auditing operate on article numbers. A window has no name, whereas an article has one, and that identifier is precisely what the traceability requirement of Section 3.1 needs in order to attribute a verdict to a source. Structure-aware segmentation, in other words, is what allows the system to cite its evidence rather than merely quote it.

The price of this choice is a dependency on the typographic regularity of the source document. Where headings deviate from the expected patterns, or where the PDF is a scanned image without a text layer, segmentation degrades; the length-normalisation stage described next is in part a mitigation of exactly this fragility.

3.4.3 Length Normalisation

Article boundaries produce semantically coherent units of highly heterogeneous length: a definition may be a single line, a procedural article several thousand characters. Both extremes are harmful to retrieval, but for opposite reasons.

Very short fragments, such as headers detected as articles, cross-references and one-line definitions, carry too little lexical evidence for their embedding to be discriminative; in a normalised vector space they tend to lie close to many unrelated queries, behaving as attractors that pollute the candidate set. They are therefore merged into the preceding segment, which restores their context without creating a spurious entry in the index. Very long articles suffer from the converse problem: bi-encoders compress a passage into a single fixed-size vector, so as the passage grows the vector converges towards the average of several distinct topics, and any individual requirement it contains becomes progressively harder to retrieve. Passages exceeding the upper threshold are consequently split with a recursive character splitter, with a small overlap between consecutive fragments so that a requirement straddling a cut is still fully present in at least one of them [29]. The concrete values of the two thresholds and of the overlap are reported and motivated in Section 4.2.

One interaction between this stage and the previous one deserves to be stated explicitly. A recursive splitter is designed to try a hierarchy of separators in decreasing order of structural significance, namely paragraph break, line break and space, so that cuts fall at the most meaningful available boundary. Because normalisation has already collapsed every newline into a space, the two structural separators are absent from the text by the time the splitter runs, which makes it fall back to whitespace boundaries. Splits are therefore syntactically safe, in that no word is ever broken, but they are not paragraph-aware. The overlap is what compensates for this: it is the mechanism that prevents a requirement cut at an arbitrary point from being lost, and its role is consequently more central here than in a pipeline that preserves layout. Retaining paragraph markers through normalisation would be a straightforward refinement, and is noted among the possible improvements in Chapter 9.

3.4.4 Asymmetric Treatment of the Two Collections

Internal policy documents are processed with a simpler strategy based on word count rather than on article detection. The asymmetry is intentional and reflects the different role the two collections play.

The external document is the *query* side: it is read once per run, it is the object whose requirements must be enumerated exhaustively, and its identifiers must appear in the final report. Its segmentation therefore optimises for normative completeness and citability. The internal corpus is the *indexed* side: it is embedded once and queried many times, it has no guaranteed normative structure because corporate policies are written as prose, tables and bullet lists rather than as numbered articles. Its fragments only need to be large enough to be understood in isolation and small enough to be retrieved precisely. Applying article-detection heuristics to documents that do not follow that convention would produce arbitrary boundaries with none of the benefits, so a uniform word-count segmentation is both simpler and more robust.

3.5 Vector Store

3.5.1 Embedding Space

Text embeddings are generated with the `sentence-transformers/all-MiniLM-L6-v2` model [46], a six-layer distilled transformer that maps a passage to a 384-dimensional unit-norm vector. Three properties made it the appropriate choice for this architecture. It is a *bi-encoder*, so each internal chunk can be embedded once, offline, and reused across every query; a cross-encoder would require re-encoding the whole corpus for every external element, which is quadratic and therefore incompatible with the retrieval stage. It is small enough to run on CPU, which keeps the local stages free of GPU requirements and lets the embedding cost stay negligible with respect to the LLM cost. And its output dimensionality is low, which matters because both index memory and distance-computation time scale linearly in the number of dimensions.

Because the model normalises its output, all vectors lie on the unit hypersphere. This makes the store's default metric adequate without further configuration: for $\|u\| = \|v\| = 1$ the identity $\|u - v\|^2 = 2 - 2\langle u, v\rangle$ holds, so ordering by Euclidean distance and ordering by cosine similarity produce the same ranking. The retrieval stage can therefore be reasoned about in terms of semantic similarity even though the index computes distances, and no explicit choice of metric is required for the two views to agree.

The choice is not without consequences. A general-purpose sentence encoder is trained on generic web and paraphrase corpora, not on legal Italian; it captures topical similarity well but is comparatively insensitive to the distinctions that carry legal weight, such as *may* versus *must*, addressee of a duty, presence of an exception. This limitation is one of the motivations for the obligation-level representation of Chapter 4, which compensates for the encoder's bluntness by giving it a shorter and semantically purer input, and it is revisited among the limitations discussed in Chapter 8.

3.5.2 Index and Persistence

The embeddings are stored in ChromaDB [12], which indexes them with an approximate nearest-neighbour structure and persists both vectors and payloads on disk. Two architectural consequences

follow.

First, the search is *approximate*. Graph-based ANN indices trade a small probability of missing a true nearest neighbour for a query cost that grows logarithmically, rather than linearly, in the size of the collection. This is the property that makes the system's cost independent of the size of the internal knowledge base: as the corpus grows, retrieval time grows slowly and the number of downstream LLM calls does not grow at all, since it is bounded by k and by the number of external elements. The scalability argument developed in Section 4.7 rests on this fact.

Second, persistence makes the index an artefact with a lifecycle of its own, decoupled from the run. Ingestion of the internal corpus is guarded by an existence check on the collection, so the embedding pass is executed only the first time; every subsequent run, and therefore every pipeline variant, queries the same stored vectors. Any difference observed in Chapter 7 is then attributable to the pipeline and not to a different vector representation of the internal policies.

Each indexed record carries, besides its text and vector, metadata such as the article number, the chunk index and the source identifier. Metadata is not decoration: it is the only channel through which a retrieved vector can be resolved back to a human-readable location in a document, and it therefore closes the traceability loop that segmentation opened.

The database is accessed through a thin wrapper exposing initialisation, insertion and collection statistics. Confining the store behind this interface keeps the vendor-specific API out of the retrieval and comparison modules, so that replacing the backend would affect a single component.

3.6 Retrieval Layer

The retrieval layer is the component that makes the problem tractable at all. Comparing every external element with every internal chunk would require $n \times m$ LLM invocations; retrieval replaces that quadratic product with a sparse relation containing $O(nk)$ candidate pairs, and does so with local computation only. Everything downstream operates on that relation.

3.6.1 Dense, Sparse and Hybrid Retrieval

The component is implemented in the `Retriever` class and supports three strategies:

- **Vector-based retrieval**: ranks documents by embedding similarity [46].
- **Keyword-based retrieval (BM25)**: ranks documents by a term-frequency / inverse-document-frequency score with length normalisation [47].
- **Hybrid retrieval**: combines the two rankings through a weighted ensemble.

The two families fail in complementary ways, which is the reason for combining them. Dense retrieval matches meaning and is therefore robust to paraphrase, which is essential because an internal policy almost never reproduces the wording of the regulation it implements. However, it is weak on rare, out-of-vocabulary tokens whose meaning is conventional rather than distributional: article numbers, statute references, internal acronyms, product names. A lexical retriever behaves in the opposite way: it cannot bridge a paraphrase, but it matches an exact identifier with certainty and its score is inherently explainable in terms of the matched terms. In a regulatory setting both behaviours are needed simultaneously, since cross-references are frequent and so is reformulation.

Hybrid retrieval is consequently the default configuration [20]. For each external input the top-k internal chunks of the fused ranking are returned.

Code 3.3: Hybrid ensemble retriever (vector + BM25), taken from `retriever.py`

```
v = self.db.as_retriever(search_kwargs={"k": search_kwargs})
bm25 = BM25Retriever.from_documents(documents)
self.vector_retriever = EnsembleRetriever(
  retrievers=[v, bm25],
  weights=weights # default [0.7, 0.3]
)
```

3.6.2 Fusion Strategy

Combining two retrievers raises a problem of *commensurability*: a cosine similarity and a BM25 score live on different, unbounded and query-dependent scales, so their values cannot be added directly. The ensemble therefore fuses the two result lists by *rank* rather than by raw score, assigning each document a contribution that decreases with its position in each list and summing

the weighted contributions. Rank-based fusion is scale-invariant by construction and empirically robust, which is why it has become the standard way of merging heterogeneous rankers [20].

The asymmetric weighting $(0.7, 0.3)$ in favour of the dense component encodes the domain assumption stated above: implementation is paraphrase, so semantic matching should dominate, while the lexical component acts as a corrective that promotes documents sharing a rare identifier without being able to override the semantic ranking on its own.

Two implementation details qualify this picture. The sparse retriever is built in memory from the document list rather than from the persistent index, so it is reconstructed per session and its coverage is exactly the set of documents handed to it. Moreover, the depth parameter K discussed throughout Chapter 4 is passed to the dense retriever only: BM25 returns its own default number of candidates, which is smaller. The consequence is that the two lists entering the fusion have different depths, and that the lexical component contributes a short, high-precision list that can promote documents into the fused top-K but cannot, on its own, populate it. This is consistent with the corrective role assigned to it, but it means that increasing K widens the semantic pool without widening the lexical one.

3.6.3 Cross-Collection Top-k Retrieval

The retrieval operation is directional and spans two disjoint collections. The set $O = \{o_1, \ldots, o_n\}$ contains the external obligations (or article-level chunks, depending on the variant), which act exclusively as queries, while $C = \{c_1, \ldots, c_m\}$ contains the searchable internal policy chunks. For each $o_i \in O$, the retriever returns the top-k elements of C according to the hybrid ranking described above; elements of O are never considered as candidates. The dense component implements approximate nearest- neighbour search in the embedding space, but the final order is produced by rank-based fusion with BM25. The complete operation is therefore more precisely described as cross-collection top-k retrieval than as a nearest-neighbour query under a single distance metric.

The result is a directed bipartite relation

$$R \;=\; \{(o, c) \in O \times C \;:\; c \in \mathrm{TopK}_C(o)\},$$

which can be represented equivalently as a sparse binary matrix $A \in \{0, 1\}^{n \times m}$ with $A_{ij} = 1$ iff c_j is among the top-k results for o_i. Each row of A has at most k non-zero entries, while the column

degrees are unconstrained: a single internal chunk may be retrieved by many external elements, or by none at all. The index map produced by the retriever, $M : O \to 2^C$, is precisely the row-wise (adjacency-list) encoding of this bipartite relation.

3.6.4 Transposing the Index Map

The operation applied before comparison is the *transpose* of this structure. The inverted map $M' : C \to 2^O$ is the adjacency-list encoding of $A^\top$, i.e. the column-wise view of the same relation R. No information is added or discarded, since $|R|$ is invariant, but the grouping key changes from the external side to the internal side. Because the comparison stage is invoked once per key, transposing turns $|R|$ potential LLM calls into at most $\min(|R|, m)$ calls, equal to the number of internal chunks with at least one incoming edge, with each call receiving a batch of obligations that share the same internal chunk. The asymmetry $n \gg m$ typical of this setting, with many short obligations compared against a relatively small internal corpus, is what makes the transposed direction strictly cheaper. Algorithm 1 computes $A^\top$ in $O(|R|) = O(nk)$ time and $O(|R|)$ space with a single pass over the map.

Algorithm 1 Transposing the Obligation-to-Chunk Index Map ($A \mapsto A^\top$)

```
1:  procedure InvertIndexMap(M) ▷ M maps each obligation to a list of similar internal chunks
2:      M' ← ∅                          ▷ Empty multi-map: chunk → list of obligations
3:      for all (o, S) ∈ M do           ▷ o: obligation, S: set of similar chunks
4:          for all s ∈ S do
5:              if s ∉ M' then
6:                  M'[s] ← ∅
7:              end if
8:              append o to M'[s]
9:          end for
10:     end for
11:     return M'
12: end procedure
```

The benefit of transposition is not only quantitative. Grouping by internal chunk means that

a single prompt contains one policy passage and *all* the external requirements that were matched to it, so the model evaluates that passage once, in a context where the competing requirements are visible together. In the untransposed direction the same passage would be re-sent in several independent prompts, each time paying its tokens again and each time being judged in isolation, with no guarantee that the resulting verdicts are mutually consistent. Transposition therefore amortises the most expensive part of the prompt, namely the internal text, and simultaneously removes a source of incoherence between verdicts.

The residual case is the symmetric one: an external element matched to several internal chunks still appears in several prompts and may receive divergent verdicts. This is not resolved by transposition but by the aggregation layer, which in every variant reconciles the verdicts concerning the same target through a unification step with worst-case escalation, described in Section 4.3.

3.6.5 Cross-Encoder NLI Filtering

In advanced configurations a Cross-Encoder Natural Language Inference (NLI) model is interposed between retrieval and comparison. The model classifies each external–internal pair as entailment, neutrality or contradiction [4, 65], and only the pairs labelled as contradictory are forwarded to the LLM. The mechanism is used by the SMALL_STRONG variant of Chapter 4.

The architectural rationale is the cascade principle introduced in Section 3.2.3. The retriever is a bi-encoder: it never sees the two texts together, since each is compressed into a vector independently, and it can therefore only estimate *topical* proximity. Whether a policy passage *contradicts* a requirement is a relational property of the pair, not of either text alone, and is thus outside what a bi-encoder can express. A cross-encoder concatenates the two texts and attends across them jointly, which makes it able to represent exactly that relation; the price is that it must be run once per pair, so it cannot be used for search but only for re-ranking a candidate set that something cheaper has already produced. Placing it after retrieval and before generation puts a discriminative model where its cost is bounded by nk and where its output removes work from the only stage that is substantially more expensive than itself.

The design also carries a distinctive risk. Standard re-rankers order candidates and let a threshold decide how many survive; here the filter is a *hard gate* on a discrete class, so a pair is discarded whenever the argmax falls outside the contradiction class, no matter how narrow the margin.

Moreover, the NLI notion of contradiction is not the legal one: a policy that is merely *silent* on a requirement, or that permits what the regulation prescribes, is typically labelled neutral rather than contradictory, even though both cases are compliance-relevant. The gate is therefore precision- and cost-oriented by construction and recall-limited by construction; the empirical magnitude of that limitation is quantified in Chapter 7.

Listing 3.4 shows the filter, which rewrites the index map in place by keeping, for each external element, only the internal chunks that survive the gate; external elements left with no surviving chunk disappear from the map altogether and never reach the LLM.

Code 3.4: Cross-Encoder NLI filter keeping only contradictions

```
model = CrossEncoder("cross-encoder/nli-MiniLM2-L6-H768")
return {
  chunkext: kept
  for chunkext, values in index_map.items()
  if (kept := [
  chunkint for chunkint in values
  if int(torch.argmax(
  torch.tensor(model.predict([[chunkext, chunkint]])[0])
  )) == 0 # Contradiction class
  ])
}
```

3.7 LLM Integration

The LLM is the only non-deterministic component of the pipeline and the only one whose cost is not negligible. The integration layer is designed around those two facts: it isolates the provider behind a uniform interface, it prescribes the shape of the model's output, and it accounts for every token consumed.

3.7.1 Provider Abstraction and Interaction Modes

The system integrates Azure OpenAI as the primary provider, configured through environment variables so that endpoints, deployments and credentials can change between development and deployment without touching the code. Two interaction modes are supported:

- **Responses API**: used for the more capable models, which expose structured output and reasoning-oriented parameters.
- **Chat Completions API**: used for lightweight inference, with conventional parameters such as temperature and token limits.

Supporting both is a requirement rather than a redundancy. The evaluation of Chapter 7 treats the model backend as an experimental variable, comparing a small and a strong model over identical pipelines; since the two generations of models are exposed through different APIs and accept different parameter sets, the comparison is only possible if the difference is absorbed by a single adapter. The rest of the system consequently invokes "the model" through one function and remains unaware of which API served the request.

3.7.2 Output Contracts and Their Enforcement

Every LLM stage is expected to return a response with a prescribed shape: a JSON payload for obligation extraction, a fixed set of named fields for comparison and for the rewriting stages of Chapter 5. Prescribing the shape is what allows the surrounding code to treat a generative stage as a component with a signature, and it is therefore the precondition for the layered decomposition of Section 3.2.1: extraction yields elements that the retriever can query with, comparison yields fields that aggregation can count, compare and escalate.

The enforcement of these contracts, however, is entirely *prompt-level*. The expected format is described in the instructions and the response is recovered by decoding it: `json.loads` for extraction, field-wise parsing for the verdicts, and a regular expression over the `Risolto` field to recover obligation identifiers, as described in Section 4.5. No provider-side constrained decoding is used, and the decoded values are not validated against a schema before being consumed.

This is a deliberate but consequential simplification. It keeps the two interaction modes of Section 3.7.1 interchangeable, since neither the prompts nor the parsing depend on features that only one API exposes. This interchangeability is precisely what makes the backend comparison of Chapter 7 possible with a single codebase. In exchange, the system inherits two failure modes that a schema-constrained decoder would exclude by construction. A response that deviates from the requested format fails at parsing time, where it can at least be observed; but a response that is *well-formed and wrong*, such as a verdict citing an obligation identifier that was never supplied or omitting one that was satisfied, passes through silently, and degrades the missing-obligation list rather than raising an error. Schema conformance would not prevent the second case either, since citation accuracy is a semantic property; but the absence of any validation layer means that the pipeline currently has no point at which such a claim is checked against the set of identifiers actually issued. Adding that check is inexpensive and is discussed among the improvements proposed in Chapter 9; the empirical robustness of the assumption, as it stands, is measured in the ablation studies of Chapter 7.

3.7.3 Cost and Token Accounting

Prompt and completion tokens are read from every response and accumulated in the coordinator, where they are converted into a monetary estimate using per-model pricing constants. Accounting distinguishes the two directions because they are priced differently and behave differently: input tokens grow with the amount of retrieved context, output tokens with the verbosity of the requested answer, and the two are influenced by different design decisions.

This instrumentation is not an operational nicety but part of the experimental apparatus. The comparison between pipeline variants in Chapter 7 is explicitly a quality-versus-cost comparison, so cost must be measured with the same rigour as accuracy; recording it inside the invocation layer guarantees that no call escapes the count, whichever stage or variant issued it. The persistence and reporting of these metrics is described in Chapter 6.

3.8 Prompt Design

If the output contract of Section 3.7.2 defines the *shape* of what the model returns, the prompt defines the *policy* it applies in producing it; since the shape itself is stated in the prompt, the two are in fact enforced by the same artefact. Prompt engineering therefore plays a critical role in shaping the behaviour of the LLM across tasks [63].

3.8.1 Prompts as External Artefacts

Prompts are not embedded in the code but stored as plain-text templates organised into named sections, parsed once and cached for reuse. Externalising them has three architectural effects. It makes a prompt a versioned artefact whose changes are visible in the repository history, which matters because a prompt revision can shift results as much as a code change and must therefore be attributable. It allows a variant to select a different section, such as the identifier-aware comparison prompt used by OBLIGATION_SIM_MIS, without duplicating the surrounding logic, which is how the "prompt" dimension of the design space is varied independently of the pipeline structure. And caching keeps the parsing cost out of the inner loop, since the same template is instantiated once per comparison across thousands of calls.

3.8.2 Prompt Contracts

The system defines one contract per task, each pairing a set of instructions with the output format that the corresponding stage expects:

- **Obligation Extraction**: distils a regulatory article into minimal normative elements in structured JSON.
- **Policy Comparison**: evaluates the consistency between external requirements and an internal passage, producing a structured verdict.
- **Verdict Unification**: consolidates several comparisons concerning the same target into a single outcome.
- **Policy Proposal**: generates harmonised text resolving an identified conflict.

- **Rewrite Decision and Application**: parses and applies amendment instructions in structured form.

The concrete templates are stored in five files: `obligations.txt` for obligation extraction, `comparison.txt` for comparison and verdict-related instructions, `proposal.txt` for harmonised policy generation, and `rewrite_decision.txt` and `rewrite_apply.txt` for the two phases of the agentic rewrite pipeline. Keeping these contracts separate allows each task to evolve without changing the orchestration code or the prompts used by the other stages.

The list mirrors the layered decomposition of Table 3.1: each contract belongs to exactly one stage and its output type is the input type of the next. Helper functions assemble the messages by combining the system instructions with the stage-specific input in a consistent order, so that the only variable part of a request is the data it carries.

Listing 3.5 reports the obligation-extraction prompt in its execution language. Besides defining the JSON contract, it limits the output to one organisational obligation and one general rule per input unit. This keeps downstream work bounded and discourages inferred or redundant requirements, but it can compress multiple distinct obligations and is therefore also a potential source of recall loss.

Code 3.5: Italian-language prompt used for obligation extraction.

```
1  Dato un estratto normativo o di policy esterna, estrai SOLO gli obblighi
       giuridicamente vincolanti
2  che una policy interna deve rispettare.
3  Produci una rappresentazione MINIMALE e NORMALIZZATA:
4
5  - A = Obbligo attuativo/organizzativo (SOLO se il testo impone esplicitamente
       un elemento organizzativo che deve essere presente nella policy)
6  - R = Regola generale o limite inderogabile
7
8  REGOLE:
9  1. Ogni obbligo = UNA frase breve e minimale
10 →solo nucleo normativo
```

```
2. Derivazione DIRETTA dal testo
→no inferenze, no deduzioni
3. Non trasformare facoltà in obblighi
4. Non restringere alternative consentite
5. Se più obblighi →sintetizza in uno per categoria
6. A va valorizzata SOLO se il testo richiede esplicitamente
che la policy includa un elemento organizzativo specifico
→altrimenti A = []
7. Se una categoria non èchiaramente presente →[]

OUTPUT (solo JSON):

{
"A": ["..."],
"R": ["..."]
}

VINCOLI:
- max 1 A
- max 1 R
- nessun testo fuori dal JSON
```

3.8.3 Constraining the Model

All prompts are written with deliberately restrictive instructions in order to limit hallucination and preserve traceability [68]. Two families of constraints are worth isolating, because both leave a visible trace in the architecture.

Budget constraints cap how much the model may produce, for example at most one organisational obligation and one general rule per article, which forces a decision about what the core requirement is instead of allowing an unbounded enumeration of paraphrases, and keeps the size of the downstream

stages predictable. *Citation* constraints require the model to echo the identifiers of the elements it acted upon, which is what makes gap detection possible at all, and which also shifts a known failure mode into the open: an identifier that is hallucinated or omitted can be checked against the set of known ones, whereas a wrong judgement expressed in prose could not.

Taken together, these constraints enforce a disciplined interaction paradigm between the system and the model. They do not make the LLM deterministic, but they confine its freedom to the dimension where its capability is actually needed, namely legal interpretation, and reduce it elsewhere.

3.9 Extension Points and Summary

The architecture described in this chapter is deliberately underdetermined at two points, and it is the enumeration of the choices available at those points that generates the pipeline variants studied in the rest of the thesis.

The first seam is the **unit of representation**: whether the external document enters retrieval and comparison as article-level text or as atomic obligations distilled by an LLM. This choice affects the embedding that the retriever sees, the amount of noise carried into the comparison prompt and the granularity at which a verdict can be attributed. The second seam is the **filter** applied to the retrieval relation: none, a cross-encoder gate before generation, or an identifier-based reconciliation after it. This choice determines where on the precision–recall curve the pipeline sits and how much of the candidate set reaches the most expensive stage.

Every other component, including ingestion, the embedding space, the index, cross-collection top-k retrieval, transposition, the invocation layer and the aggregation logic, is held fixed. Chapter 4 instantiates the two seams into four concrete pipelines and analyses the resulting trade-offs conceptually; Chapter 7 measures them.

4. Pipeline Architectures: A Comparative Design

This chapter describes the four pipeline variants implemented on top of the common architecture introduced in Chapter 3. The first variant, BASE, is the simplest realisation of the comparison workflow and serves as the reference design. The investigation then shifts to an obligation-oriented intermediate variant, OBLIGATION_SIM, which decouples the normative content from its textual container. The remaining two variants, OBLIGATION_SIM_MIS and SMALL_STRONG, are optimisations and extensions built on top of that intermediate design, targeting respectively traceability and cost control. The chapter closes with a conceptual trade-off analysis; the corresponding empirical validation is deferred to Chapter 7.

4.1 Architecture Overview: Common Stages

All variants share the high-level workflow introduced in Chapter 3 and differ only in two well-defined points: the representation used at retrieval time and the filtering strategy applied before LLM comparison. The shared stages are:

1. **Document ingestion and preprocessing.** External PDFs are loaded, normalised and segmented into article-level units.

2. **Chunking and length normalisation.** Articles are merged or split based on size thresholds to ensure consistent retrieval inputs.

3. **Retrieval.** Internal policy chunks are obtained through a hybrid retriever combining dense embeddings and BM25. Because external elements act exclusively as queries and candidates are always drawn from the internal collection, the stage performs directional cross-collection top-K retrieval between the two disjoint sets.

4. **Index map transposition.** The retrieval result is a relation between external elements and internal chunks; transposing it regroups the external elements per internal chunk so that the comparison stage receives compact batches.

5. **LLM-based comparison.** Each (internal chunk, external elements) pair is evaluated to produce a structured verdict.

4.2 Tunable Parameters and Their Motivation

The implementation exposes a small set of global constants and per-variant arguments. The values currently in use were not chosen arbitrarily: each encodes a trade-off between recall, precision, token cost and runtime. The following list documents every parameter worth explaining.

MIN_ARTICLE_CHAR = 300**.** Lower bound below which an article fragment is merged into the previous chunk. Regulatory PDFs frequently contain short articles, such as definitions, cross-references or incorrectly split headers, that carry limited semantic content and produce noisy embeddings [20]. Three hundred characters (roughly two to three sentences) was empirically found to stabilise retrieval behaviour [33].

MAX_ARTICLE_CHAR = 1 000**.** Upper bound above which an article is subdivided by a `RecursiveCharacterTextSplitter`. This value is aligned with typical chunk sizes recommended in the sentence-transformer literature for models such as `all-MiniLM-L6-v2`, whose retrieval quality tends to degrade for significantly longer inputs [46].

`chunk_overlap` = 50 **characters.** Overlap used by the recursive splitter. The value preserves cross-sentence references while limiting redundancy across chunks, which would otherwise inflate the index and dilute similarity scores [29].

Embedding model `sentence-transformers/all-MiniLM-L6-v2`. Selected because it offers a favourable trade-off between computational cost and semantic quality, runs efficiently on CPU and produces compact 384-dimensional embeddings well suited to vector indexing [46, 62].

Hybrid retriever weights (0.7, 0.3)**.** The ensemble combines a dense retriever and a sparse BM25 retriever. The dense component receives the higher weight because policy text is usually paraphrased rather than copied verbatim, which makes semantic similarity dominant [20].

BM25 still contributes by recovering rare terms and structured identifiers such as article numbers [47].

K **(retriever `search_kwargs`).** Number of internal chunks returned per query. Its value is tuned per variant to balance recall against the downstream cost of either NLI filtering or LLM comparison.

`max_length` = 2 000 characters. Cap on the length of grouped obligations passed to the LLM. The threshold is conservative enough to fit typical LLM context windows while leaving room for the prompt and the structured output.

`asyncio.Semaphore(5)`. Maximum number of concurrent LLM calls. The value was selected empirically to balance throughput against the API rate limits available in the development environment.

CrossEncoder model `cross-encoder/nli-MiniLM2-L6-H768`, label 0**.** Used as the filtering step of Small_Strong. The model performs natural language inference, where label 0 corresponds to *contradiction*, enabling the rejection of irrelevant pairs before LLM comparison [51].

LLM pricing constants in `settings.py`. Cost parameters for GPT-4o-mini and GPT-5.3 are kept in a single file for telemetry purposes; costs are logged after each run to support the quantitative comparison reported in Chapter 7.

4.3 Base: Full Chunk-to-Chunk Comparison

The Base architecture performs a direct comparison between every external chunk and its top-K retrieved internal counterparts.

Procedure. After article-level chunking and short-chunk merging, each external chunk c_i is used as a vector query against the hybrid retriever with K=5. Collectively these queries produce a directed bipartite retrieval relation between the external chunks and the internal collection, represented by the index map $M : c_{\text{ext}} \rightarrow [c_{\text{int}}]$. The map is then *transposed* (Algorithm 1) into $M' : c_{\text{int}} \rightarrow [c_{\text{ext}}]$

and passed to the asynchronous comparison stage. Multiple verdicts targeting the same internal passage are unified by a dedicated *unification prompt* that applies worst-case escalation: a single "In contrasto" is enough to force the final verdict to *conflict*.

Complexity. The number of LLM comparison calls is bounded by $O(C_{\text{ext}} \times K)$, the size of the top-$K$ retrieval relation, in the worst case before transposition, and by the number of distinct internal chunks appearing in it afterwards. In practice, the dominant cost factor is the comparison stage, since retrieval is purely local (embedding dot-products plus BM25 scoring).

Trade-offs. The pipeline is conceptually simple and deterministic, with no intermediate LLM call before comparison. It nonetheless sends entire article texts, which often contain non-normative preambles or definitions, into the comparison prompt, thereby inflating token consumption and exposes the model to irrelevant context that can obscure the operative legal requirements.

4.4 Obligation_Sim: Obligation-First Retrieval

Obligation_Sim marks the shift to an obligation-oriented design. Instead of comparing entire articles, the pipeline first distils each external article into a small number of atomic *obligations* and uses those statements as retrieval queries. Figure 4.1 shows where this additional representation stage enters the shared pipeline and how the missing-obligation extension builds on its output.

Procedure.

1. External articles are chunked and merged as in Base.
2. An LLM obligation-extraction call is issued per chunk, subject to the concurrency limit. The extraction prompt mandates a minimal JSON output containing at most one `A` (organisational/implementation obligation) and one `R` (general rule or inderogable limit) per article.
3. The extracted `Obligation` objects, each carrying an auto-incremented numeric ID, a type, the textual content and a back-reference to the source `Article`, are flattened into a single list.

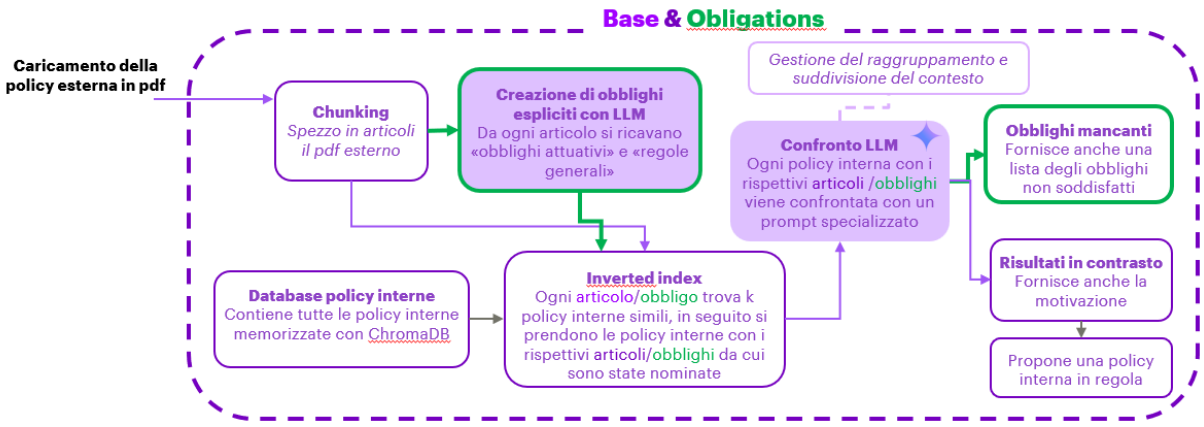


Figure 4.1: Obligation-based pipeline structure. The obligation variants add an LLM extraction stage before retrieval; the extracted obligations replace article chunks as retrieval and comparison units, while OBLIGATION_SIM_MIS additionally derives unresolved obligations from the comparison output.

4. Each obligation's `content` is used as a retrieval query against the hybrid retriever with K=20; together, the queries produce a directed bipartite relation between obligations and internal chunks.

5. The resulting index map {obligation → [int_chunks]} is *transposed* into {int_chunk → [obligations]} and dispatched to the asynchronous comparison stage.

4.4.1 Why Obligations Are the Retrieval Unit

The transformation from articles to obligations is not merely a textual normalisation. A regulatory article commonly combines the operative rule with definitions, legal references, exceptions, procedural details and background clauses. When the full article is embedded, its vector representation captures all of these elements. The normative requirement that must be checked against an internal policy can therefore be diluted by content that is relevant to legal interpretation but not to retrieval.

An extracted obligation instead makes the operational meaning explicit. It expresses the relevant subject, required or permitted action, and applicable condition in a concise form. For example, an article concerning a parent who may interrupt leave when a child is hospitalised can contain several qualifications about requests, time periods and cross-references. The corresponding obligation

retains the central requirement: a parent may interrupt leave in the event of the child's hospitalisation. This narrower representation reduces normative noise and gives the retriever a more stable semantic target.

This distinction matters because vector similarity reflects the overall semantic context rather than the legal effect alone. Two articles may use similar terminology while imposing different duties, whereas equivalent requirements may be expressed through substantially different vocabulary. Querying with an obligation focuses the dense and sparse retrievers on the normative proposition to be matched, reducing false positives caused by descriptive or structurally similar text.

The transformation does not discard the legal source. Each `Obligation` preserves a back-reference to its originating `Article`; after retrieval, the transposed map batches obligations under a common internal chunk, and every verdict can still be traced back through its obligation to the article that produced it. The architecture consequently separates two concerns: the original article remains the traceable legal source, while the obligation is the compact operational representation used for retrieval and comparison. This preserves normative traceability in the final result without sacrificing the precision gained from obligation-level matching.

Complexity. The total number of LLM calls is C_{ext} (extraction) plus the number of comparison calls after transposition. Retrieval scales as $O(\text{Obl} \times K)$ embedding lookups. The high K=20 is necessary because obligations are short, focused sentences whose embedding may not align perfectly with verbose policy passages; a wider pool compensates for the sparser signal.

Trade-offs. By distilling obligations first, the comparison prompt receives precisely scoped normative statements rather than noisy full-article text, which improves the specificity and traceability of verdicts. The trade-off is an additional LLM round-trip for extraction and a dependency on the quality of that extraction: any obligation the prompt misses will never reach the comparison stage. The pipeline does not scale with the internal database size because retrieval remains sublinear, but it scales linearly with the number of external obligations.

4.5 Obligation_Sim_Mis: Missing-Obligation Detection

Obligation_Sim_Mis extends Obligation_Sim with an obligation-traceability mechanism that identifies which extracted obligations remain *unresolved* by the internal policy.

Procedure. The pipeline reuses the extraction and retrieval stages of Obligation_Sim. The differences are concentrated around the comparison call:

1. Obligations are formatted as `"<id>: <content>"` before being placed in the index map, so that the prompt can cite specific obligation IDs in its `Risolto` field.

2. The comparison uses a dedicated *MIS* prompt variant (`[comparison_system_mis]`) that instructs the LLM to list, by numeric ID, the organisational obligations explicitly satisfied by the internal passage.

3. Once all comparisons complete, the system collects the set of resolved IDs across all verdicts (via regex extraction of integers from the `Risolto` lines) and computes the complement: obligations of type `A` whose ID never appears in any verdict are flagged as **missing**.

Output. The function returns the standard enriched comparison results together with the list of missing obligation texts, so that downstream consumers, namely the Streamlit frontend and the agentic rewrite pipeline of Chapter 5, can surface the gaps that require policy amendments.

Complexity. Asymptotically identical to Obligation_Sim: the only addition is the final ID-set difference, which is $O(|\text{Obl}|)$ and therefore does not change the overall complexity class.

Trade-offs. Carrying obligation IDs end-to-end is what unlocks the gap-detection signal, but it does not come for free. The comparison prompt grows because every obligation must be serialised with its identifier and the model is asked to echo IDs back in a structured field, which inflates both input and output tokens. More importantly, the mechanism trusts the LLM to cite the correct IDs: when the model returns incorrect IDs or omits them, the pipeline keeps producing verdicts but the list of missing obligations silently becomes unreliable. This ID-citation accuracy is not

independently measured in the present evaluation and therefore remains a limitation of the reported missing-obligation signal.

4.6 SMALL_STRONG: CrossEncoder NLI Pre-Filtering

SMALL_STRONG explores an orthogonal optimisation. Rather than acting on the unit of comparison, it inserts a lightweight neural filter between retrieval and the LLM call, aiming to eliminate semantically unrelated pairs before they incur LLM cost. Figure 4.2 shows the resulting branch: the CrossEncoder receives retrieved pairs and controls which of them proceed to LLM comparison.

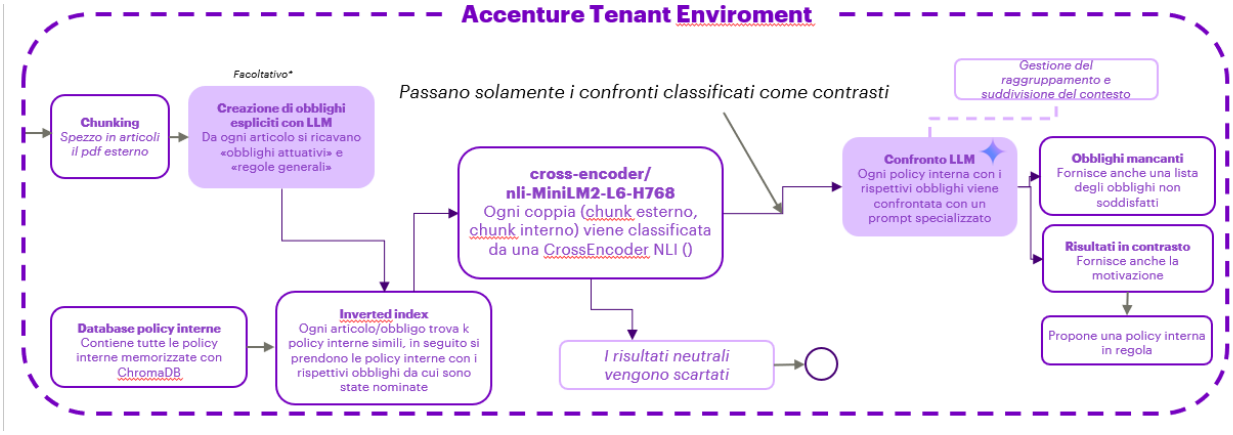


Figure 4.2: Structure of the SMALL_STRONG variant. Retrieved external–internal pairs pass through a local NLI CrossEncoder, and only pairs classified as contradictions are forwarded to the LLM comparison stage. The diagram shows the intended filtering path; its quantitative cost–recall effect is not evaluated in this thesis.

Procedure. Internal top-K chunks are retrieved as usual, with a K=10 that anticipates the downstream filter. Instead of forwarding all $(c_{\text{ext}}, c_{\text{int}})$ pairs to the LLM, each pair is first scored by a CrossEncoder Natural Language Inference model (`cross-encoder/nli-MiniLM2-L6-H768`). The model produces logits over three classes, *Entailment*, *Neutral* and *Contradiction*, and only pairs classified as **Contradiction** (label 0, i.e. the internal policy contradicts the external norm) are retained for the LLM comparison.

Rationale. The NLI CrossEncoder acts as a cheap binary gate: pairs that are merely neutral or entailing are treated as unlikely to represent genuine regulatory conflicts and are discarded. This restricts the expensive LLM calls to pairs where a semantic tension has already been flagged by a smaller model, but it also creates a direct false-negative risk.

Complexity. CrossEncoder inference requires $O(C_{\text{ext}} \times K)$ forward passes through a six-layer transformer. The effective number of LLM comparisons becomes $O(|\{(i,j) : \text{NLI}(i,j) = \text{Contradiction}\}|)$ and therefore depends directly on the selectivity of the gate; its practical reduction is not quantified here.

Trade-offs. The approach is expected to reduce cost when most pairs are non-conflicting. Its weakness is recall: false negatives, meaning true conflicts classified as neutral by the CrossEncoder, are silently dropped. Because complete run-level measurements were not retained, neither the cost savings nor the recall loss is quantified in the present evaluation.

4.7 Trade-off Analysis

The four pipelines share the skeleton described in Section 4.1 but diverge along a small number of design axes. This section organises those axes and discusses the implications conceptually; the empirical validation, together with numbers and tables, is reported in Chapter 7.

4.7.1 Axes of Divergence

Three independent design decisions generate the architectural space covered by the four variants:

- **Unit of comparison**: the granularity at which the external document is presented to the LLM. The pipelines either feed raw article-level chunks (Base, Small_Strong) or first decompose each chunk into atomic obligations (Obligation_Sim, Obligation_Sim_Mis).
- **Filtering strategy**: what happens between retrieval and comparison. The options are: no filter (Base, Obligation_Sim), an NLI CrossEncoder that keeps only contradictory pairs (Small_Strong), or an ID-based *post*-filter that tracks which obligations were resolved (Obligation_Sim_Mis).

- **Retrieval depth**: the parameter K that controls how many internal chunks are returned per query. It is chosen coherently with the previous two axes rather than as an independent knob: filtered or ID-tracked variants can afford a smaller K, while unfiltered obligation-level retrieval needs a wider pool to compensate for the noisier signal carried by short obligation strings.

4.7.2 Article-level vs. Obligation-level Comparison

The choice of unit is the most consequential design decision. Article-level pipelines are simpler to implement and avoid an entire LLM stage, but each chunk typically packs several regulatory concepts into the same block of text. The resulting embedding averages heterogeneous content and the comparison prompt forces the model to reason simultaneously about multiple, possibly unrelated, obligations.

Obligation-level pipelines pay an upfront cost, namely an extra LLM call per chunk to extract the atomic requirements, in exchange for a structured representation of the external document. The expected benefits are *conceptual* rather than purely quantitative: each comparison addresses one regulatory intent at a time, embeddings become more discriminative because they cover a narrower semantic region, and each verdict can be tied back to a specific obligation. The hypothesis is that this granularity pays off most on non-trivial conflicts, where the model needs to reason about roles, conditions or discretionary clauses; on simple textual mismatches the two families should behave comparably.

4.7.3 Filtering: CrossEncoder vs. No Pre-filter vs. ID Tracking

The way irrelevant material is discarded shapes the cost profile of each variant.

- BASE and OBLIGATION_SIM apply *no pre-filter*: every retrieved pair reaches the comparison prompt. The design is the simplest, but the LLM has to do the heavy lifting of recognising and discarding off-topic matches, which inflates token usage.

- SMALL_STRONG interposes an *NLI CrossEncoder* that classifies each (external chunk, internal chunk) pair and keeps only those labelled as contradictory. The CrossEncoder is a small,

local model, so its marginal cost is negligible compared with an LLM call; the expected effect is a sharp reduction in the number of comparison calls at the price of recall on pairs whose contradiction is too subtle for the NLI head to capture.

- OBLIGATION_SIM_MIS does not filter retrieval at all but introduces an ID-based *post*-filter: by propagating obligation IDs through the prompt and parsing them back from the verdicts, it derives the complement set of unresolved obligations. This adds traceability without removing pairs from the comparison stage.

The three strategies sit at different points of the same precision/recall trade-off curve: the CrossEncoder maximises precision per LLM call, the unfiltered variants maximise recall, and the ID-tracking variant moves the trade-off into a different dimension: coverage of obligations rather than selection of pairs.

4.7.4 Retrieval Depth and Coverage

A natural temptation is to treat K as a free knob and to increase it until recall is satisfactory. The relationship is not monotonic: beyond a certain depth, additional retrieved chunks are mostly off-topic and act as distractors inside the comparison prompt, so a larger pool can dilute rather than improve the verdict. Each variant therefore chooses a K consistent with its filtering strategy: aggressive when a downstream filter exists and conservative when the LLM has to read every retrieved chunk. It relies on the unit of comparison, not on retrieval depth alone, to drive coverage.

4.7.5 Cost and Scalability

The four architectures scale differently with the size of the external document and of the internal database.

BASE is the most exposed to scaling pressure: comparison cost is proportional to the number of external chunks times K, and both multipliers grow with the input. SMALL_STRONG attenuates this growth by pruning pairs with the CrossEncoder before they reach the LLM, trading a fixed local cost for a variable LLM cost.

Obligation-level pipelines shift the cost structure. The extraction step introduces a fixed overhead per external chunk, but the comparison stage then operates on short obligation strings rather than

full articles, which makes each LLM call cheaper and more focused. As the external document grows, the dominant term becomes the obligation extraction; as the *internal* database grows, none of the obligation-based variants is directly affected, because retrieval cost remains logarithmic in the collection size while comparison cost depends on K, not on the database size. This makes them, in principle, better suited to large regulatory corpora.

4.7.6 Traceability and Downstream Use

Article-level pipelines emit a verdict per (external chunk, internal chunk) pair: useful for surfacing conflicts, but coarse for downstream automation, because a single chunk may carry several obligations that are not individually addressable. Obligation-level pipelines emit a verdict that can be associated with a specific atomic requirement, which enables two capabilities that are out of reach for the chunk-based variants: explicitly flagging which regulatory requirements are satisfied or violated, and identifying the complement set of obligations that the internal policy never addresses. These outputs are the inputs that the agentic rewrite pipeline and the Streamlit frontend consume to propose amendments and to support compliance reporting.

4.7.7 Summary of Expected Trade-offs

The four variants populate the design space along clearly identifiable trade-offs:

- BASE prioritises simplicity over precision and scalability.
- OBLIGATION_SIM prioritises semantic granularity and stable scaling, accepting a heavier preprocessing stage.
- OBLIGATION_SIM_MIS prioritises traceability and gap detection, accepting a longer prompt and a stronger dependence on the LLM's ability to cite obligation IDs correctly.
- SMALL_STRONG prioritises precision and cost control via NLI filtering, at the expected price of recall on subtle conflicts.

A second, orthogonal dimension is the choice of LLM backend, which sits outside the architectural design but interacts with every variant: a more capable model is expected to amplify the

benefits of obligation-level reasoning, where most of the cognitive effort is concentrated. As shown in Chapter 7, the combination of an obligation-based pipeline with a stronger model produces reliable detection on the controlled test set and richer, obligation-level explanations.

5. Agentic Rewrite Pipeline

5.1 Motivation

The comparison pipelines described in Chapter 4 identify conflicts and missing obligations between an external regulation and the internal policy, but they stop at *diagnosis*: they report *what* is wrong without producing a corrected text. In regulatory practice, however, EU-style amending regulations do not restate an entire directive; instead, they issue surgical amendment instructions of the form "Article 6, paragraph 4 is replaced by the following. . . ". Applying these amendments manually is tedious, error-prone [3], and requires the operator to locate the base article, parse the amendment scope, and produce the updated version verbatim.

The **Agentic Rewrite Pipeline** closes this gap by automating the full cycle: *parse* the amendment instruction, *retrieve* the target article, and *apply* the modification to produce the updated legal text, with each stage orchestrated by an LLM-driven agent loop.

Two implementations coexist in the codebase:

- `simplerewrite.py`: a deterministic, single-LLM-call pipeline that relies on regex-based article ID extraction and direct lookup (no decision LLM).
- `agenticrewrite.py`: the full agentic architecture with a two-step LLM loop (decision + apply), flexible article matching, and structured Pydantic output models.

The agentic variant is the primary focus of this chapter; the simple variant serves as a deterministic reference implementation. The agentic loop exists solely to widen the range of amendment-document formats the pipeline can ingest. It is not introduced as a cost or latency improvement over the simple variant and, in fact, incurs one additional LLM call per chunk. This chapter documents the implemented design; amendment quality is not evaluated in Chapter 7.

5.2 EU-Style Amendment Processing

EU amending regulations express modifications via a constrained natural-language grammar [17]. A typical amendment chunk contains:

- A **target reference** (e.g. "Article 6, paragraph 4, point (b)").
- An **operation verb** ("is amended", "is replaced by the following", "the following point is added", "is deleted").
- The **new text** to insert, replace, or append.

The system models each amendment chunk as an `AmendmentChunk` Pydantic object with three fields:

- `id`: unique identifier (e.g. `AMEND-01`).
- `content`: the raw amendment text.
- `source_document`: optional reference to the amending regulation.

Operation taxonomy. The pipeline classifies every chunk into one of five operations:

Operation	Semantics
`add`	New text is inserted at the indicated position
`replace`	Existing text is substituted verbatim
`delete`	Indicated paragraph/point is removed
`amend`	Partial modification (broader scope)
`skip`	Not actionable (e.g. recitals, footnotes)

5.3 Agent Decision Loop

The `RewriteAgent` class orchestrates a sequential loop over the amendment chunks. For each chunk, a two-phase LLM interaction is executed:

5.3.1 Phase 1: Decision / Parse

The *decision prompt* (`rewrite_decision.txt`, section `[system]`) instructs the LLM to:

1. Identify the **target article** and paragraph/point being amended.
2. Determine the **operation** type from the five-valued taxonomy.
3. Decide whether **retrieval of the base article** is needed (always `true` unless `operation = skip` or the amendment introduces a standalone new provision).
4. Formulate a **retrieval query** (typically the article number and paragraph reference).
5. Provide a one-sentence **justification**.

The LLM is constrained to reply exclusively in JSON matching the `AgentDecision` schema:

```
{
  "target_article": "<article id or null>",
  "target_paragraph": "<paragraph reference or null>",
  "operation": "add|replace|delete|amend|skip",
  "needs_retrieval": true|false,
  "retrieval_query": "<query string or null>",
  "reason": "<one-sentence justification>"
}
```

A post-processing step strips Markdown code fences if the LLM returns a fenced JSON block before parsing it into the `AgentDecision` Pydantic model. If the JSON is malformed, the parser raises a `ValueError`.

5.3.2 Phase 2: Retrieve and Apply

If `needs_retrieval = true` and `operation ≠ skip`:

1. The `lookup_article` function searches the base article list (see Section 5.4).

2. The *apply prompt* (`rewrite_apply.txt`) receives both the original article text and the amendment instruction, and must produce the **full updated article text** with the amendment applied verbatim.

If retrieval is not needed or the operation is `skip`, the chunk is marked as skipped with the decision's `reason` field preserved in the output.

5.3.3 Output model

Each processed chunk produces a `RewriteResult`:

- `chunk_id`: traces back to the input `AmendmentChunk`.
- `target_article`, `operation`: from the decision phase.
- `retrieved_text`: the original article text (if retrieval occurred).
- `rewritten_text`: the LLM-produced updated article.
- `skipped`, `skip_reason`: flag and explanation for non-actionable chunks.

5.3.4 Contrast with the simple pipeline

The `simplerewrite.py` variant collapses the two-phase loop into a single step: it extracts the article number via regex (`Article\s+(\d+)`), performs a direct dictionary lookup in the parsed base document, and immediately calls the apply LLM. It works only when amendment instructions follow a single, regex-friendly reference format and cannot handle ambiguous references, multi-article amendments, or non-actionable chunks. The agentic variant exists precisely to lift this restriction: by delegating target identification and operation classification to an LLM, it accepts heterogeneous amendment-document formats that the regex-based variant cannot parse. The agentic loop is therefore a *coverage* mechanism, not a cost or latency optimisation; if anything, it adds one extra LLM call per chunk.

5.4 Article Lookup and Output Parsing

5.4.1 Base document preparation

The base regulation PDF is loaded via `PyPDFLoader` and split into `Article` objects using `manual_split_docs_ext` (the same regex-based article extractor used in the comparison pipelines). Each article is indexed by its numeric ID.

5.4.2 Multi-strategy matching

The `lookup_article` function implements a three-tier matching strategy to resolve the `target_article` string produced by the decision LLM:

1. **Exact match**: case-insensitive string equality between the LLM's output and the stored `Article.id`.
2. **Normalised match**: the pattern "Article N" is normalised to "ART-N" format (e.g. `"article 6"` → `"art-6"`), accommodating variations in how the LLM or the document express article identifiers.
3. **Substring fallback**: either the query appears within the stored ID or vice-versa, catching cases like `"6"` matching `"ART-6"` or `"Article 6 paragraph 4"` matching `"6"`.

This cascading strategy ensures robustness against LLM formatting variability [40] without requiring embedding-based retrieval for article lookup.

5.4.3 Apply prompt constraints

The apply prompt enforces strict fidelity to the amendment:

- For "is added": insert the new text at the indicated position, preserving existing content.
- For "is replaced by the following": substitute the indicated portion with the new text verbatim.
- For "is deleted": remove the indicated paragraph/point.

- Numbering, structure, and formatting of the unaffected portions must be preserved.
- The LLM must *not* paraphrase; it reproduces the amendment text exactly as given.
- Output consists solely of the full updated article text (no preamble or explanation).

5.4.4 Error handling

If `lookup_article` returns `None` (no match), the chunk is marked as skipped with a diagnostic message recording the failed query. This allows the operator to review unresolved amendments without the pipeline halting.

6. Implementation Details

6.1 Technology Stack and Project Structure

6.1.1 Core dependencies

The application is implemented in Python 3.10+ and depends on the following key libraries:

6.1.2 Project structure

The project follows a mostly flat layout, with a dedicated `architecture/` package for pipeline orchestration. In addition to core pipeline files, operational modules such as `telemetry.py` and utility modules for prompts/retrieval are first-class components of the runtime.

The codebase structure is:

```
project/
 |-- architecture/
 |-- chunkingOutput/
 |-- docs/
 |-- policy_embeddings/
 |-- PolicyEsterne/
 |-- PolicyInterne/
 |-- prompts/
 |-- test/
 |-- thesis/
 |-- .env
 |-- .gitignore
 |-- agenticrewrite.py
 |-- db_manager.py
 |-- frontend.py
 |-- models.py
 |-- prompt_loader.py
 |-- requirements.txt
 |-- retriever.py
```

```
|-- settings.py
|-- simplerewrite.py
|-- telemetry.csv
|-- telemetry.log
`-- telemetry.py
```

The `Architecture` class delegates most behaviours to submodules only when their methods are invoked. However, object initialisation remains eager for core resources (database wrapper, embeddings, Azure client), so this delegation does not defer runtime dependency initialisation.

6.2 Configuration and Environment Management

6.2.1 Centralised settings

All tuneable constants are collected in `settings.py`:

6.2.2 Environment variables

Sensitive credentials and deployment-specific parameters are loaded from a `.env` file via `python-dotenv` (with `override=True`):

- `endpoint`: Azure OpenAI resource URL.
- `subscription_key`: API authentication key.
- `api_version`: Azure API version string.
- `deployment`: Model deployment name.
- `max_tokens`: Maximum completion tokens (Chat Completions mode).
- `temperature`: Sampling temperature (Chat Completions mode).

The `.env` file is excluded from version control. The `USE_RESPONSE_API` flag in `settings.py` acts as a global switch: when `True`, all LLM calls use the Azure Responses API targeting GPT-5.3; when `False`, they fall back to the Chat Completions API with the `max_tokens` and `temperature` environment variables.

6.3 Streamlit Frontend

The user interface is implemented as a single-page Streamlit application (`frontend.py`) with a wide layout.

6.3.1 Sidebar: model and document selection

The sidebar exposes three control sections:

1. **Model selector**: a dropdown offering the four pipeline variants: `base`, `small_strong`, `obligations`, `obligationMISS`. At runtime these map to `base`, `SMALL_STRONG`, `OBLIGATION_SIM`, and `OBLIGATION_SIM_MIS`, respectively.

2. **External PDF picker**: lists all `.pdf` files found in `PolicyEsterne/` and lets the user select the external regulation to analyse.

3. **Internal document manager**: lists files in `PolicyInterne/` and provides an "Add document to DB" button that invokes `arc.load_doc(filepath)`, triggering PDF loading, character-based chunking, and insertion into ChromaDB.

6.3.2 Main panel: analysis and results

Clicking the "Analizza Conflitti" button triggers the selected pipeline. Results are presented in a structured layout:

- **Summary metrics**: two columns showing the number of detected conflicts and an overall status badge.

- **Verdict breakdown**: counts of *In contrasto*, total comparisons.

- **Missing obligations** (when using `obligationMISS`): an expandable list of obligations not satisfied by any internal passage.

- **Conflict detail cards**: for each conflict, a styled card (red left border) showing the extracted *Motivo*, with expandable sections for:

- The external policy text in conflict.
- The internal policy text in conflict.
- The LLM-proposed harmonised replacement text.

Figure 6.1 presents the complete results view, while Figure 6.2 focuses on the information exposed for an individual conflict.

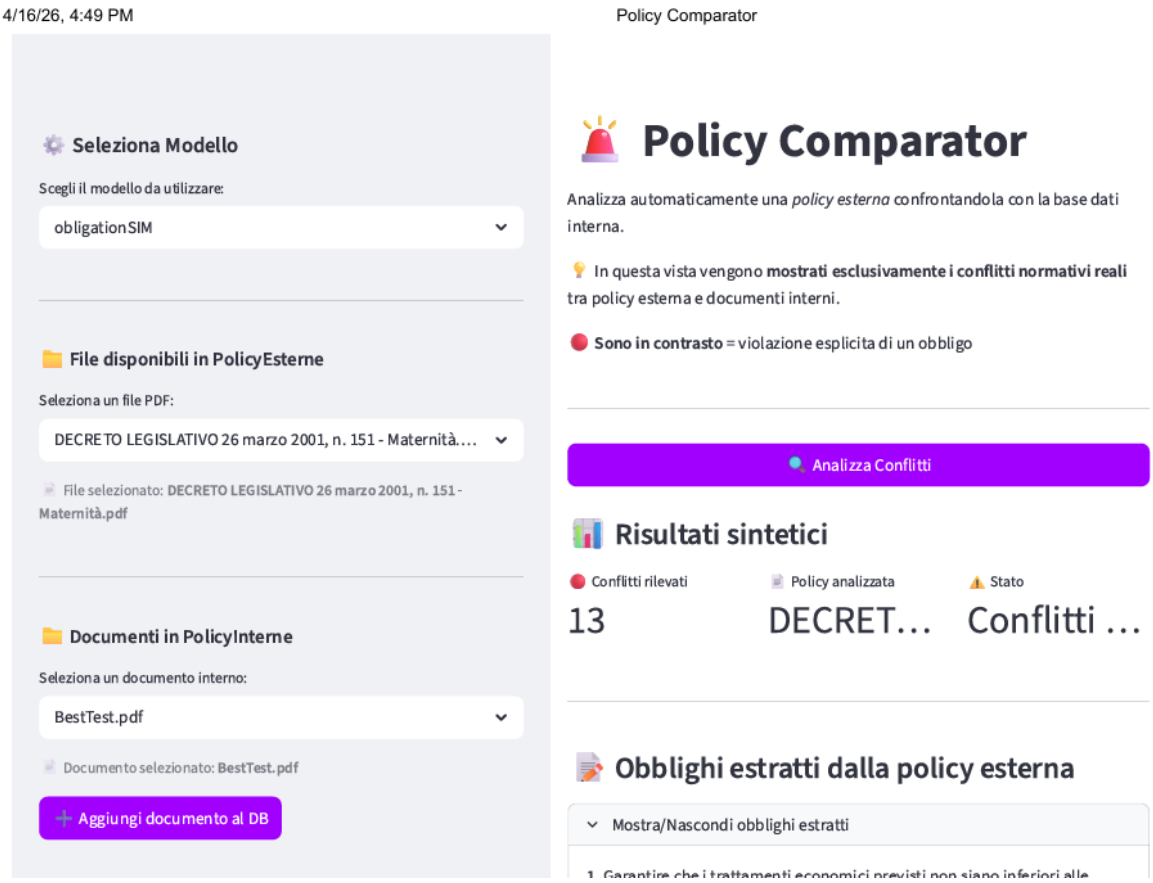


Figure 6.1: Main Streamlit interface for selecting the pipeline and policy documents, starting the conflict analysis, and inspecting the resulting summary and extracted obligations.

6.3.3 Cost and performance dashboard

A footer section displays live-accumulated metrics from the `Architecture` instance:

- Total elapsed time (seconds).

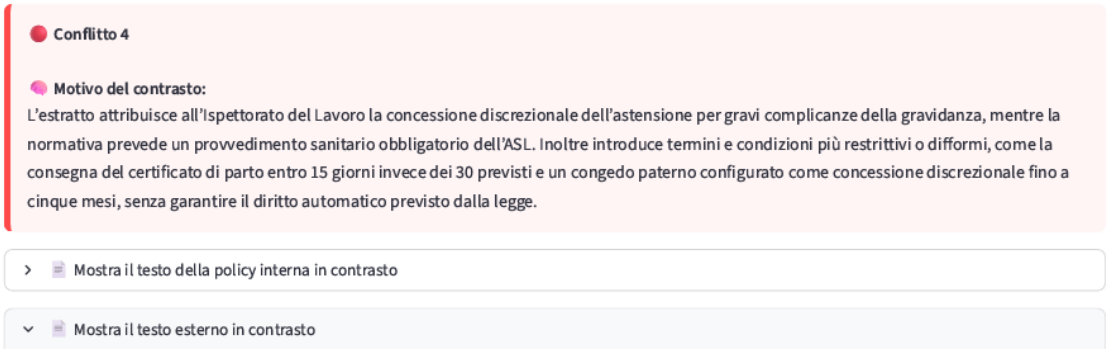


Figure 6.2: Detail view of a detected conflict, showing the model-generated rationale and the expandable source passages used to inspect the external and internal evidence.

- Total tokens consumed (formatted with "k" suffix for readability).
- Total cumulative cost in USD (displayed to six decimal places).

Figure 6.3 shows how these three run-level indicators are presented together in the interface.

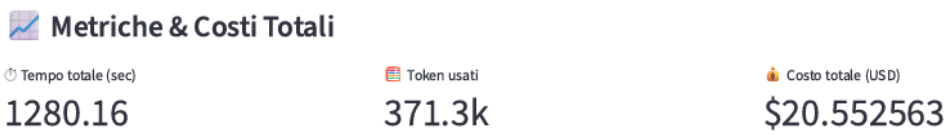


Figure 6.3: Cost and performance dashboard reporting total execution time, token consumption, and cumulative estimated cost for the completed analysis.

6.3.4 Styling

Custom CSS is injected via `st.markdown(unsafe_allow_html=True)` to style the primary action button with Accenture brand colours (`#A100FF` purple with white text).

6.4 Telemetry and Observability

6.4.1 Architecture

The telemetry subsystem (`telemetry.py`) uses Python's standard `logging` module with two handlers attached to a dedicated `"telemetry"` logger:

1. **Console handler**: streams `INFO`-level log records to `stdout` for real-time monitoring during development.
2. **CSV handler** (`CSVHandler`): a custom `logging.Handler` subclass that appends structured rows to `telemetry.csv`.

6.4.2 CSV schema

Each telemetry row contains seven columns:

6.4.3 Public API

- `log_metrics(metrics, operation)`: iterates over a dictionary of metric key/value pairs and writes one CSV row per entry, all sharing the same timestamp and operation label.
- `log_error(error, extra)`: logs a single exception row with the error message, class name, and optional context dictionary.
- `load_last_total_cost(csv_path)`: reads the CSV and returns the most recent `total_cost_usd` value, used at startup to initialise the `Architecture` cost accumulator so that cumulative cost is preserved across sessions.

6.4.4 Metrics collected per pipeline run

Each pipeline variant logs the following metrics at completion:

- `time_db_load`: seconds to load/reload ChromaDB.
- `time_chunking`: seconds for article extraction and merging.

- `number_of_chunks`: total external chunks produced.
- `time_vector_retrieval_total`: seconds for all retrieval queries.
- `time_comparison`: seconds for the async LLM comparison stage.
- `time_total`: wall-clock time for the entire pipeline.
- Cumulative cost metric (currently logged as `total cost` in pipeline metrics; `total_cost_usd` used in frontend event logs and startup restoration).
- `total_tokens`: cumulative token count (obligation-based variants).

Additionally, the frontend logs an `openai_invoke` event after each user-triggered analysis with the selected architecture name, total elapsed seconds, and total cost.

6.4.5 File management

The CSV file is created with a header row on first access (`open(..., "x")` with `FileExistsError` handling) and appended to on all subsequent writes. This ensures no data loss across application restarts while keeping the implementation dependency-free (no external time-series database required).

Table 6.1: Primary technology stack.

Layer	**Library**	**Role**
LLM client	`openai` (Azure SDK)	Azure OpenAI calls via Responses API or Chat Completions [36]
Orchestration	LangChain (`langchain_classic`, `langchain_community`)	Retriever abstractions and ensemble retrieval [27]
Vector store	`langchain-chroma` / ChromaDB	Persistent embedding store and similarity retrieval [11]
Embeddings	`langchain-huggingface` / `sentence-transformers`	`all-MiniLM-L6-v2` embeddings [50]
NLI filtering	`sentence-transformers` CrossEncoder	Contradiction filtering (`nli-MiniLM2-L6-H768`, small_strong variant) [46]
Keyword retrieval	`langchain-community` BM25Retriever	In-memory lexical ranking for hybrid retrieval [47, 27]
Chunking utilities	`langchain_text_splitters`, `langchain-experimental`	Manual article extraction; semantic chunker available but not primary in active variants
Data models	Pydantic v2	Typed schema for Article and Obligation objects [45]
PDF parsing	`PyPDFLoader`	Page-level PDF ingestion
Frontend	Streamlit	Interactive single-page UI [55]
Environment	`python-dotenv`	Runtime configuration loading from `.env`

Table 6.2: Configuration constants in `settings.py`.

Constant	Value	Purpose
EMBEDDING_PATH	"./policy_embeddings"	ChromaDB persistence directory
EXT_POLICY_PATH	"./PolicyEsterne"	Default folder for external PDFs
PRICE_PROMPT	$1.75/M tokens	GPT-5.3 input token pricing
PRICE_COMPLETION	$14.00/M tokens	GPT-5.3 output token pricing
PRICE_PROMPT1	$0.15/M tokens	GPT-4o-mini input pricing
PRICE_COMPLETION1	$0.60/M tokens	GPT-4o-mini output pricing
MAX_ARTICLE_WORDS	200	Word limit triggering chunk splitting
MAX_ARTICLE_CHAR	1 000	Character limit for article chunks
MIN_ARTICLE_CHAR	300	Minimum size before merging
USE_RESPONSE_API	True	Toggle: Responses API (GPT-5.3) vs. Chat Completions (GPT-4o-mini)

Table 6.3: Telemetry CSV schema.

Column	Type	Description
timestamp	float	Unix epoch timestamp of the event
operation	str	Pipeline stage identifier (e.g. "ObligationSIM stats")
metric_key	str	Metric name (e.g. "total_cost_usd", "time_comparison")
metric_value	Any	Numeric or string value
error	str	Exception message (error rows only)
type	str	Exception class name (error rows only)
extra	str	Arbitrary additional context

7. Experimental Evaluation

This chapter provides the empirical counterpart to the conceptual trade-off analysis of Chapter 4. It describes the dataset, the evaluation protocol and the metrics used; reports quantitative results on the three unfiltered pipeline variants and on the two LLM backends; and concludes with a qualitative analysis followed by a component-level ablation study. The agentic rewrite extension of Chapter 5 is outside the scope of this evaluation; the experiments assess retrieval and conflict detection rather than the quality of generated amendments. Given the controlled document pair, the small ground truth and the single-run efficiency measurements described below, the results should be read as a proof-of-concept evaluation rather than as a statistically generalisable benchmark.

7.1 Dataset Description

The evaluation is built around a document-comparison task in which an external regulatory policy is checked against an internal corporate policy. The external dataset is a regulatory document of approximately fifty pages focused on maternity and family-protection law. The internal dataset is a corporate policy of approximately ten pages, preprocessed and indexed in the vector database described in Chapter 3.

To make the evaluation controllable, four regulatory conflicts were intentionally injected into the internal policy. The conflicts span a range of complexity, from explicit numerical discrepancies to structural inconsistencies that affect multiple provisions at once. The four categories used throughout this chapter are:

- **Simple**: explicit numerical discrepancies (e.g. different time limits for the same right).
- **Intermediate**: conflicts that require distinguishing mandatory provisions from discretionary ones.
- **Medium-hard**: incorrect authority attribution or subtle legal misinterpretation.
- **Complex**: structural inconsistencies that affect multiple provisions and alter the legal intent of the policy.

Table 7.1 makes the construction of the controlled ground truth explicit. The descriptions identify the kind of intervention introduced into the internal policy and the evidence required to recognise it; the source passages themselves are assessed in their full document context during manual validation.

Table 7.1: Composition of the four-conflict controlled ground truth.

Case	**Injected intervention**	**Evidence required for detection**
Simple	A quantitative condition is changed so that the internal policy states a different limit from the regulation.	Match the same right or duty across the two documents and compare the explicit values.
Intermediate	A mandatory regulatory provision is represented as discretionary in the internal policy.	Distinguish obligation from permission and preserve the legal force of the provision.
Medium-hard	Responsibility for a requirement is attributed to the wrong authority or organisational actor.	Resolve the relevant roles and determine which actor legally owns the duty.
Complex	Coordinated changes across multiple provisions alter the overall legal intent of the internal policy.	Combine evidence from several related provisions rather than treating each sentence as an isolated discrepancy.

The dataset may also contain real-world inconsistencies beyond these injected cases. The pipelines are therefore not constrained to detect only the planted conflicts and may legitimately surface additional ones, which are counted manually during result validation.

7.2 Evaluation Protocol and Metrics

Each pipeline is evaluated on the same external/internal pair under two conditions: the default setup (around ten chunks in total) and a stressed setup that fragments the same content into roughly fifty

chunks. The stressed setup is used to probe scalability under increased document granularity, as anticipated in Section 4.7.

Detection metrics. The output of each pipeline is a set of final verdicts. Each conflict-labelled verdict counts as one reported conflict and is manually labelled as *correct*, *warning* or *error*. Multiple verdicts may refer to the same underlying inconsistency, so precision is computed at verdict level, whereas the four injected conflicts are tracked as distinct cases for recall. A warning counts as a relevant detection for precision because it identifies a genuine inconsistency, but contributes to recall only when it maps to one of the injected cases. Consequently, the warning count alone does not determine injected-conflict recall. Labelling is performed by the author on the basis of the two source passages read in their full document context; no independent second annotator was available, a constraint discussed in Section 8.2.

- **Precision**: proportion of reported conflicts labelled as correct or warning rather than error.
- **Recall**: proportion of the four injected conflicts that are correctly identified [44].
- **Errors**: number of reported conflicts that are factually wrong (false positives or misclassifications).
- **Warnings**: conflicts correctly identified but accompanied by an incomplete or partially incorrect explanation.

Efficiency metrics. Computational efficiency is captured by the wall-clock execution time of the end-to-end pipeline, the total number of tokens consumed by the LLM backend, and the corresponding monetary cost computed with the pricing constants logged in `settings.py`.

Execution times and token counts are approximate measurements from single runs. They support descriptive comparison in this evaluation but do not provide estimates of run-to-run variance or statistical significance.

7.3 Quantitative Results

The quantitative analysis is organised in three parts. First, the impact of the LLM backend is isolated on a single pipeline. Second, the three unfiltered pipeline variants of Chapter 4 are compared head-to-head on the default setup. Third, the effect of document granularity is measured by repeating the comparison on the stressed setup.

7.3.1 LLM Backend: GPT-5.3 vs. GPT-4o-mini

To isolate the effect of the LLM backend, the Obligation_Sim pipeline is run twice with identical inputs and parameters, changing only the model behind every LLM call (obligation extraction and comparison). Table 7.2 reports the result.

Table 7.2: Effect of the LLM backend on Obligation_Sim. Recall is computed over the four injected conflicts; precision is computed over all reported conflicts.

Metric	GPT-5.3	GPT-4o-mini
Execution time	5 min	30 min
Token usage	400k	340k
Total conflicts reported	4	4
Injected cases recovered	4 / 4	1 / 4
Warnings	0	1
Errors	0	2
Precision	100%	50%
Recall	100%	25%

On this four-conflict ground truth, the gap is substantial: GPT-5.3 reaches perfect precision and recall on the injected conflicts, while GPT-4o-mini recovers only one of the four and contributes both false positives and partially incorrect explanations. GPT-5.3 also completes this single run six times faster despite the slightly higher aggregate token count; the available telemetry does not isolate whether this difference comes from model latency, request scheduling, or another runtime

factor. This baseline justifies the choice of GPT-5.3 as the reference backend for the remaining experiments in this section.

7.3.2 Comparison of the Unfiltered Pipeline Variants

The three unfiltered variants of Chapter 4 are evaluated on the default setup using GPT-5.3 as backend. Complete run-level metrics were not retained for SMALL_STRONG, so that variant is excluded from the quantitative comparison rather than evaluated using incomplete data. It should consequently be interpreted as an implemented architectural proposal whose cost–recall trade-off remains unvalidated. Table 7.3 reports detection and efficiency metrics side by side; OBLIGATION_SIM_MIS additionally contributes the *missing-obligation* count, which the article-level variants cannot produce.

Table 7.3: Head-to-head comparison of the three unfiltered pipeline variants on the default setup (GPT-5.3 backend). The last column reports the missing-obligation signal, available only on OBLIGATION_SIM_MIS.

Variant	Exec. time	Tokens	Precision	Recall	Errors	Missing obl.
BASE	5 min	400k	100%	100%	0	Not applicable
OBLIGATION_SIM	5 min	400k	100%	100%	0	Not applicable
OBLIGATION_SIM_MIS	11 min	450k	100%	100%	0	yes

On the default setup, all three unfiltered variants reach perfect detection on the injected conflicts. OBLIGATION_SIM_MIS matches the recall of its sibling while also surfacing the set of obligations that the internal policy does not address, paying only a small overhead in tokens and runtime.

7.3.3 Document Granularity: the Stressed Setup

To probe scalability, the same conflict set is re-evaluated after fragmenting the external document into roughly fifty chunks instead of ten. Table 7.4 contrasts the default and stressed setups for the two main architectural families.

Table 7.4: Impact of document granularity on BASE and on the obligation-based pipelines (GPT-5.3 backend, default setup at 10 chunks vs. stressed setup at 50 chunks).

Metric	BASE (10)	BASE (50)	OBLIG. (10)	OBLIG. (50)
Token usage	400k	880k	400k	400k
Reported conflicts	4	3	4	3
Errors	0	0	0	0
Precision	100%	100%	100%	100%
Recall	100%	75%	100%	75%

Two effects emerge. First, fragmenting the document degrades recall by the same amount for both families: one of the four conflicts is split across chunk boundaries and is missed regardless of the unit of comparison. Second, only the obligation-based pipeline keeps its cost profile flat: BASE more than doubles token usage, confirming the scaling argument of Section 4.7. The obligation-extraction stage absorbs the additional fragmentation upstream and shields the comparison stage from the token-cost increase.

7.4 Qualitative Analysis

Beyond aggregate metrics, the verdicts produced by each variant were inspected to assess how well they capture different categories of legal reasoning. This section first reports one complete run of the pipeline on a single obligation, from extraction to final verdict, so that the subsequent discussion rests on observable output rather than on aggregate counts alone. The four categories of Section 7.1 are then revisited from the perspective of explanation quality.

7.4.1 Worked Example: from Obligations to Verdict

The example below is reproduced verbatim from a run of OBLIGATION_SIM_MIS with the GPT-5.3 backend on the maternity and family-protection pair described in Section 7.1. Because both documents are in Italian, the system output is reported in its original language and accompanied by an English gloss; the text is abridged only where marked with "[. . .]". The internal policy is a

real corporate document, and the employer's name has been replaced by "[Azienda]" throughout. The three stages shown correspond to the obligation-extraction, retrieval and comparison stages of Chapter 3.

Stage 1 — Obligations extracted from the external regulation. The extraction prompt distilled the relevant portion of the regulation into four self-contained normative statements, each phrased as a requirement that the internal policy must satisfy:

O1. "La policy interna non può in alcun modo comprimere o derogare alle condizioni di maggior favore previste da leggi, regolamenti, contratti collettivi o altre disposizioni vigenti in materia di congedi, riposi, permessi e tutele connesse a maternità e paternità, che restano inderogabilmente salve e pienamente applicabili."

O2. "Le policy interne [. . .] devono garantire che i trattamenti economici riconosciuti per gli istituti disciplinati dal testo unico non siano in ogni caso inferiori alle indennità previste dalla normativa vigente, costituendo queste un livello minimo inderogabile."

O3. "La policy interna deve garantire il divieto assoluto e non derogabile di qualsiasi discriminazione fondata sul sesso, includendo ogni trattamento meno favorevole connesso allo stato di gravidanza, alla maternità o paternità (anche adottive) nonché alla titolarità o all'esercizio dei relativi diritti, senza prevedere condizioni, limitazioni o misure che possano comprimerne o scoraggiarne l'esercizio."

O4. "La policy interna deve garantire l'applicazione non discrezionale dello sgravio contributivo del 50% previsto per le assunzioni in sostituzione di lavoratrici o lavoratori in congedo [. . .] nel rispetto del limite temporale inderogabile fino al compimento di un anno di età del figlio [. . .], senza introdurre condizioni che ne comprimano l'effettività."

In English, the four obligations require respectively that (O1) the internal policy never derogate from more favourable statutory or collective-agreement conditions on leave and parental protections; (O2) economic treatments never fall below the statutory minimum; (O3) no condition, limitation or measure restrict or discourage the exercise of parental rights, including adoptive parenthood; and (O4) the 50% contribution relief for replacement hires be applied non-discretionally. Each statement is short, self-contained and independently comparable, which is the property that the obligation-first design of Section 4.7 is intended to produce.

Stage 2 — Internal passage retrieved for comparison. Hybrid retrieval returned the following fragment of the corporate policy as the most relevant internal evidence for the obligation set:

> "[. . .] benefici economici e contributivi, si applicano integralmente anche in caso di adozione, sia nazionale sia internazionale, e di affidamento preadottivo. Anche il padre adottivo ha diritto a usufruire dei periodi di congedo previsti. A tal fine, è necessario presentare a [Azienda]: • una dichiarazione dell'altro genitore che attesti la rinuncia al diritto al congedo facoltativo; • una certificazione rilasciata dal datore di lavoro dell'altro genitore che confermi tale rinuncia. [. . .] **1.14 Riposi per allattamento** Durante il primo anno di vita del bambino, le madri lavoratrici hanno diritto a usufruire di due periodi di riposo retribuiti al giorno [. . .]"

The passage states that adoptive fathers are entitled to the statutory leave periods, but conditions that entitlement on two documents: a declaration by the other parent waiving the optional leave, and a certification from that parent's employer confirming the waiver.

Stage 3 — Verdict produced by the comparison stage. The comparison prompt, given the obligation set and the retrieved passage, emitted the following structured verdict:

> **Esito:** In contrasto
> **Motivo:** "La policy subordina il diritto del padre adottivo al congedo alla presentazione della rinuncia dell'altro genitore e alla certificazione del datore di lavoro, introducendo una condizione non prevista dalla normativa che limita l'esercizio autonomo del diritto. Questo requisito può comprimere l'effettività delle tutele di congedo previste dalla legge."

In English: the policy subordinates the adoptive father's right to leave to the production of the other parent's waiver and of the employer's certification, introducing a condition not foreseen by the legislation that limits the autonomous exercise of the right and may compress the effectiveness of the statutory protections.

What the example demonstrates. Four observations follow from this run, each of which generalises to the behaviour described in the rest of this section.

- **The conflict is procedural, not lexical.** The internal policy does not contradict any figure or deadline, and it explicitly *grants* the right in question; a keyword- or number-matching

approach would find nothing to flag. The violation lies in the *addition* of a precondition, which GPT-5.3 recognised as an instance of the prohibition in O1 and O3 against conditions that restrict the exercise of a protected right. This is precisely the *intermediate* category of Section 7.1, where mandatory provisions must be told apart from conditioned or discretionary ones, and precisely the category on which the weaker backend of Section 7.3.1 produced errors rather than detections.

- **The verdict is attributable and therefore actionable.** The justification names the offending clause (the waiver and certification requirements) rather than the article that contains it. A reviewer can act on the verdict by amending two bullet points, whereas the corresponding chunk-level verdict produced by BASE flagged the adoption provision as a whole and left the identification of the offending sentence to the reader. This is the interpretability gap quantified nowhere in Table 7.3 but visible in every inspected verdict.

- **The comparison stage is robust to retrieval noise.** The retrieved passage mixes two unrelated topics: adoption leave and, from "1.14 Riposi per allattamento" onwards, nursing breaks. The model confined its verdict to the relevant portion and did not fabricate a conflict from the nursing-break text. Grounding therefore held even when the retrieved context was only partly on topic.

- **Segmentation artefacts are visible in the evidence.** The same passage begins mid-sentence and is truncated mid-word ("garantendo l"), a direct consequence of the character-based chunk boundaries of Chapter 3. Here the truncation was harmless because the decisive sentences fell inside the chunk. In the stressed setup of Section 7.3.3 the same mechanism splits one of the four injected conflicts across two chunks and suppresses it entirely, which is the concrete form taken by the recall degradation reported there.

Finally, the example illustrates the limits of automation discussed in Section 8.3. The verdict is legally plausible but not self-certifying: whether the requested certification is a merely evidentiary formality or a constitutive precondition is a question of legal interpretation that a qualified reviewer must settle. The system's contribution is to surface the clause and to articulate why it warrants scrutiny, not to decide the matter.

7.4.2 Verdict Quality Across Conflict Categories

All variants identified the *simple* numerical conflicts and produced explanations of comparable quality: when the discrepancy is a single number, the unit of comparison and the filtering strategy matter little.

On *intermediate* conflicts, the obligation-based pipelines generated more legally grounded explanations than BASE: by operating on isolated normative statements, the comparison prompt could distinguish between a mandatory rule and a discretionary provision and state the distinction explicitly in the verdict. BASE reached the same verdict but described it in vaguer terms, because the surrounding article text introduced unrelated context.

On *medium-hard* cases involving incorrect authority attribution or subtle misinterpretation, the gap widened. The obligation-level verdicts cited the specific obligation that had been misrepresented, whereas the chunk-level verdicts blamed the article as a whole and were correspondingly harder to act upon.

Complex conflicts, which alter the legal intent across multiple provisions, provided the clearest distinction between architectures. The obligation-based pipelines surfaced the systemic nature of the conflict: each affected obligation appeared in its own verdict, making the pattern visible. BASE, by contrast, reported a single conflict per article and tended to attribute the systemic issue to one isolated violation, which under-represented the gravity of the inconsistency.

7.5 Ablation Studies

The ablation analysis isolates the contribution of each design decision of Chapter 4, treating the empirical results above as input rather than restating them.

LLM backend. Switching from GPT-4o-mini to GPT-5.3 lifts recall from 25% to 100% and removes both errors and warnings (Section 7.3.1). The two backends agree on the easy cases but diverge sharply on conflicts that require reasoning over roles and conditions, confirming the hypothesis of Section 4.7 that a stronger model amplifies the benefits of obligation-level reasoning.

Unit of comparison. On the default setup the article-level and obligation-level pipelines match in detection metrics, but the qualitative inspection (Section 7.4) shows that obligation-level verdicts are more precise and easier to map back to specific normative requirements. This is also the prerequisite for the downstream agentic rewrite pipeline of Chapter 5.

Filtering strategy. A quantitative ablation is not reported for the NLI pre-filter of SMALL_STRONG, because complete run-level measurements were not retained. Its expected cost–recall trade-off therefore remains to be tested in a controlled evaluation. The ID-based post-filter of OBLIGATION_SIM_MIS adds traceability at a small cost overhead and is the only evaluated mechanism that surfaces *missing* obligations rather than mere conflicts.

Document granularity. Increasing the number of chunks degrades recall from 100% to 75% across all variants (Section 7.3.3), because at least one injected conflict is split across chunk boundaries. The degradation is therefore a function of the segmentation strategy, not of the pipeline; mitigation requires acting on the chunking stage of Chapter 3 rather than on the comparison stage.

Scalability. Under the stressed setup, BASE more than doubles token consumption, while the obligation-based pipelines stay flat on this metric. The result is consistent with the cost model of Section 4.7: in BASE the comparison cost grows with the number of chunks times K, whereas in the obligation-based variants the comparison stage operates on a stable number of distilled statements.

Summary of findings.

- A capable LLM is a prerequisite: weaker backends fail on exactly the conflicts where the obligation-based design is most valuable [63].
- Obligation-level pipelines match article-level pipelines on aggregate metrics but produce more actionable explanations and unlock the missing-obligation signal.
- NLI pre-filtering remains a promising cost-control mechanism, but its quantitative effect requires a controlled evaluation.
- Recall degradation under aggressive chunking is a segmentation problem and must be addressed upstream of the comparison stage.

8. Discussion

This chapter interprets the empirical results of Chapter 7 in the light of the research objectives stated in Section 1.3. It draws together the main findings, acknowledges the limitations that bound their generality, and reflects on the ethical and legal implications of deploying an AI-assisted compliance system in practice.

8.1 Key Findings

Before turning to the qualifications below, the central result is a positive one: the conflict-detection pipeline works end to end. With a capable backend it recovers all of the injected conflicts (Section 7.3.1) and attributes each one to the specific provision it violates. The separate agentic pipeline of Chapter 5 extends this workflow to amendment generation, but the quality of its rewritten text is not evaluated in Chapter 7. The findings that follow delimit the conditions under which the evaluated detection system delivers on its promise.

The experiments support four principal conclusions.

Capability differed substantially between the tested backends. The most consistent finding across the evaluation is that, in the tested setup, detection quality was strongly influenced by the choice of the underlying LLM. Replacing GPT-4o-mini with GPT-5.3 lifted observed recall on the injected conflicts from 25% to 100% and eliminates both errors and warnings (Section 7.3.1). Crucially, the two backends agree on the *simple* numerical conflicts and diverge only on the cases that require reasoning over roles, conditions and authority attribution, which are precisely the conflicts that motivate an automated system in the first place. The practical reading is sober: AI can assist legal compliance analysis, but in this experiment the stronger performance was tied to the higher-priced backend. The cheaper backend did not prove to be a viable substitute for the difficult conflicts in the tested configuration.

The unit of comparison shapes actionability more than aggregate metrics. On the default setup the article-level (BASE) and obligation-level pipelines are indistinguishable in precision and recall

(Section 7.3.2), yet the qualitative analysis (Section 7.4) shows that the two designs are far from equivalent. Operating on isolated normative statements lets the obligation-based variants attribute a conflict to the specific obligation that was misrepresented, distinguish mandatory from discretionary provisions, and expose the systemic nature of *complex* conflicts that BASE collapses into a single, under-stated violation. The obligation-first representation is also what makes the missing-obligation signal of OBLIGATION_SIM_MIS and the downstream agentic rewrite of Chapter 5 possible at all. In other words, the value of the obligation layer is not visible in the headline numbers but in the interpretability and downstream usefulness of the verdicts.

The obligation layer stabilises token consumption under fragmentation. The stressed setup confirms the scaling argument of Section 4.7: under aggressive fragmentation BASE more than doubles token consumption, while the obligation-based pipelines retain the same reported total token count because the extraction stage absorbs fragmentation upstream of the comparison stage. The obligation-first design therefore shows a potential cost advantage as external-document granularity grows in the tested setup, although exact monetary cost also depends on the input–output token mix. Scalability with larger internal corpora and other regulatory documents is not established by this experiment. The cost–recall effect of the SMALL_STRONG NLI pre-filter still requires a controlled quantitative evaluation (Section 7.5).

Segmentation is the observed recall bottleneck under stress. The chunking experiments expose a limitation observed across all evaluated variants using GPT-5.3, rather than one specific to a comparison strategy. Increasing the number of chunks degrades recall uniformly from 100% to 75% across those variants (Section 7.3.3), because at least one conflict is split across chunk boundaries and is never surfaced for comparison. This points to a broader hypothesis: general-purpose sentence embeddings may capture the semantics of legal provisions less reliably than they capture general-domain similarity, so the retrieval stage may become a limiting factor for recall [8]. In the present experiment, the direct evidence supports improving segmentation before further tuning the comparison prompt; a retrieval benchmark would be needed to generalise the claim beyond this setup.

8.2 Limitations

Several limitations bound the generality of the results above and should be kept in mind when interpreting them.

Manually annotated ground truth. To the best of the author's knowledge, no public benchmark specifically targets Italian corporate-compliance conflict detection. The ground truth was therefore constructed by manually injecting four conflicts of graded difficulty and by manually labelling every reported verdict as correct, warning or error (Section 7.2). This introduces two constraints. First, the small number of injected conflicts makes each individual recall figure coarse-grained: a single missed conflict moves recall by twenty-five percentage points. Second, manual labelling carries an unavoidable element of annotator judgement, particularly for the *warning* category where an explanation is partially but not wholly correct.

Unvalidated obligation extraction and ID tracking. The obligation-based variants depend on two LLM-generated intermediate signals that are not evaluated independently. An obligation omitted during extraction never reaches retrieval or comparison, while a hallucinated or omitted identifier in OBLIGATION_SIM_MIS can corrupt the missing-obligation list without invalidating the surrounding verdict. The end-to-end results therefore do not establish extraction recall or ID-citation accuracy against a separately annotated obligation set.

Dependence on a costly backend. The headline detection results are achieved with GPT-5.3, whose input and output token prices (Table 6.2) make repeated, document-scale analyses operationally expensive. The SMALL_STRONG variant and the obligation-first design mitigate but do not remove this dependence: the quality that makes the system useful is concentrated in the most expensive component, and the cheaper backend did not prove to be a viable substitute for the difficult conflicts in the tested configuration.

8.3 Ethical and Legal Considerations

The intended use of the system is to *assist* compliance professionals, and this framing carries ethical and legal weight that the technical results alone do not convey.

Human oversight and the limits of automation. The system is designed as a decision-support tool, not as an autonomous arbiter of legal compliance. Its verdicts and rewritten provisions are proposals that must be reviewed and ratified by a qualified human before they carry any operational effect. This is not merely a precaution: legal text is inherently ambiguous and context-dependent, and the same provision may be conflicting or compatible depending on the broader normative framework (Section 1.2) [56]. Meaningful oversight consequently requires more than formal approval of a generated answer. Reviewers must have sufficient expertise, authority and contextual information to understand, verify and, where necessary, challenge both the conclusion and the reasoning offered in its support. The obligation-level representation developed in this work contributes to that process by linking a detected conflict to the relevant internal statement and external provision, rather than presenting an unsupported binary judgement.

Accountability for AI-suggested text. Because the harmonisation stage produces concrete replacement text, the allocation of responsibility for that text must be settled before deployment. Adopting an AI-suggested provision does not transfer accountability to the model. The organisation should instead define, under the applicable law, contractual arrangements and professional duties, who is authorised to approve the text and who is responsible for the resulting decision. The system's value is in accelerating the drafting and triage of candidate revisions, while final legal judgement remains subject to review by qualified professionals.

Legal liability for AI-assisted compliance assessments. Responsibility also arises before the harmonisation stage, when the system classifies provisions as conflicting or compatible. These assessments are decision-support outputs rather than autonomous legal determinations or a substitute for authoritative legal advice. An organisation must determine how they may be used and allocate responsibility under the applicable legal and organisational framework, whether the system produces a false positive that prompts an unnecessary intervention or a false negative that leaves a genuine

conflict unaddressed. Their decision-support role, rather than the probabilistic or deterministic nature of the underlying technology, is what limits their intended operational effect. RAG grounding, structured outputs and source references can constrain and expose the reasoning process, but they cannot guarantee that an interpretation is complete or correct.

The distinction between system output and organisational decision is important in practice. A reviewer may accept, reject or modify a recommendation after considering facts and legal context unavailable to the pipeline. Governance procedures should accordingly identify who is authorised to approve each type of decision, when a result must be escalated, and how disagreement between the reviewer and the system is resolved. Such procedures preserve professional responsibility: as with conventional legal research tools, automation can support analysis without replacing the judgement expected from lawyers, compliance officers and risk specialists.

Hallucination and the need for verification. RAG grounding reduces, but does not eliminate, the risk that the model fabricates or misstates a regulatory requirement [52, 68]. The evaluation itself observed both false positives and partially incorrect explanations on the weaker backend (Section 7.3.1), a reminder that every verdict must be traceable to, and verifiable against, the source text. The obligation-level and ID-based traceability mechanisms of the stronger variants are therefore not only quality features but governance features: they make verification by a human reviewer practical, which is a precondition for using the system responsibly in any compliance workflow.

Auditability and documentary evidence. Compliance assessments may subsequently be examined during internal audits, regulatory inspections, litigation or periodic compliance reviews. In those settings, retaining only the final verdict would provide an incomplete account of how the conclusion was reached. An auditable workflow should preserve, subject to proportionate retention rules, the relevant input documents, retrieved passages, model and configuration used, generated analysis, reviewer interventions and final human decision. These records can support error investigation and reproducibility, while also demonstrating that an appropriate review process took place. The traceability mechanisms evaluated in this thesis provide part of this evidential chain by connecting each reported conflict to identifiable provisions. They do not, by themselves, supply a

complete audit trail, which would also require controlled logging, versioning, access restrictions and documented approval procedures.

Automation bias and professional judgement. High measured accuracy may itself create a governance risk if it encourages reviewers to treat recommendations as presumptively correct. Such automation bias can reduce independent scrutiny precisely where complex provisions require contextual interpretation [53]. The perfect recall observed with the stronger backend on the four injected conflicts (Section 7.3.1) should not be read as a guarantee for unseen documents, especially given the limitations of the test set discussed above. Meaningful human control therefore depends on both interface-level traceability and organisational measures: reviewers need sufficient time and competence to test the cited evidence, record reasons for consequential decisions, and escalate uncertainty rather than merely confirm the model's output.

Regulatory currency and lifecycle governance. A compliance assessment is valid only in relation to the sources and factual context available when it is produced. Amendments to legislation, regulatory guidance or internal policy may render an earlier conclusion obsolete even when the original analysis was reasonable. Operational use would therefore require ownership of the retrieval corpus, reliable source-version tracking, controlled update procedures and periodic reassessment of conclusions that remain operationally relevant. This lifecycle perspective also changes the meaning of reproducibility: an organisation must be able to identify not only which model produced a verdict, but also which version of each legal source was retrieved at that time.

Dependence on third-party model providers. Use of an external foundation-model service introduces contractual and operational dependencies beyond the technical limitations measured in this thesis. A provider may change model behaviour, availability, safety controls, context limits or pricing, with consequences for the consistency, cost and reproducibility of the compliance workflow. The backend comparison in Section 7.3.1 illustrates why model substitution cannot be assumed to preserve legal-analysis quality. Deployment governance should therefore include change assessment, version monitoring and a defined process for revalidation when a model or service configuration changes. Contractual terms concerning service availability, data handling and retention also form part of that assessment.

Data protection and confidentiality. Corporate policies and compliance records may contain personal data, confidential business information or legally sensitive material. Sending such documents to an external service consequently requires an assessment of the applicable data-protection and confidentiality obligations rather than a purely technical integration decision. Appropriate controls may include data minimisation, purpose limitation, role-based access, encryption, defined retention periods and deletion procedures [18]. The choice of what to log for auditability must be balanced against these same principles: retaining every prompt and document indefinitely could improve retrospective inspection while creating an excessive repository of sensitive information. Audit records should therefore be limited to what is necessary, protected according to their sensitivity and retained only for a justified period.

Evolving governance expectations. Requirements applicable to AI-assisted legal and compliance systems continue to develop alongside broader AI governance frameworks. The EU AI Act illustrates this development, although the obligations applicable to a particular deployment depend on the system's classification and intended use [19]. Rules and professional guidance may impose specific expectations concerning transparency, risk management, human oversight, monitoring and documented accountability in high-impact professional settings. The framework should therefore not be treated as governed once at deployment. Its technical and organisational controls need periodic review so that changes in law, professional duties, model behaviour and operational use are reflected throughout the system lifecycle. Designing for traceability and human review from the outset makes such adaptation more feasible, but does not remove the organisation's responsibility to determine which obligations apply in its particular jurisdiction and use case.

9. Conclusions and Future Work

9.1 Summary of Contributions

This thesis set out to determine whether a large language model can assist the detection and resolution of conflicts between regulatory text and internal corporate policy, a task that, in current practice, requires a compliance professional to read the two documents side by side and reason over them line by line. The central outcome is that it can. With a capable backend the proposed pipeline recovers all of the injected conflicts and attributes each one to the specific provision it violates, turning an essentially manual cross-reading exercise into an automated detection workflow. The agentic rewrite pipeline extends this workflow to amendment generation, but the quality of its generated text remains outside the scope of the present evaluation.

Around this result the work makes four contributions.

An end-to-end conflict-detection and harmonisation pipeline. The thesis presents a complete RAG-based system that ingests regulatory and policy documents, retrieves candidate provision pairs, classifies them for conflict, and proposes harmonised replacement text. The design addresses Italian corporate-compliance conflict detection end to end rather than treating retrieval, comparison and harmonisation as isolated tasks.

An obligation-level representation that makes verdicts actionable. Replacing the article as the unit of comparison with the individual obligation lets the system attribute a conflict to the precise normative statement that was misrepresented, separate mandatory from discretionary provisions, and expose the systemic structure of complex conflicts. As Chapter 8 shows, this value is invisible in aggregate precision and recall but decisive for interpretability and for the downstream rewrite stage.

A characterisation of the cost, scalability and recall trade-offs. Through controlled variation of the backend and document fragmentation, the evaluation maps where detection quality comes from and what it costs: capability tracks the strength and price of the LLM, the obligation-first design

keeps a flat comparison-cost profile as granularity grows in the tested setup, and segmentation becomes the observed constraint on recall under stress. The NLI pre-filter defines an additional cost–recall trade-off architecturally, but is not quantified by the present evaluation.

An evaluation protocol and annotated test set. In the absence of any public benchmark, the thesis contributes a reproducible evaluation built on injected conflicts of graded difficulty and a manual correct/warning/error labelling of every verdict, together with the metric definitions needed to compare pipeline variants on equal terms.

9.2 Future Directions

The limitations identified in Chapter 8 point to four concrete avenues for further work.

Domain-adapted retrieval and segmentation. Because recall is bounded by the retrieval stage rather than the reasoning stage, the most promising improvement is upstream: legal-domain sentence embeddings and conflict-aware segmentation that avoids splitting a single conflict across chunk boundaries. Overlapping or provision-aware chunking and embeddings fine-tuned on normative text are the natural starting points.

Reducing dependence on a costly backend. The quality of the system is concentrated in its most expensive component. Future work should explore whether targeted fine-tuning or distillation of a smaller model on the specific reasoning patterns of compliance conflicts (roles, conditions and authority attribution) can recover the difficult cases at a fraction of the inference cost. A complementary direction is to replace the hard contradiction gate of SMALL_STRONG with a calibrated threshold: borderline pairs would still reach the LLM for final verification, allowing the system to tune the cost–recall balance instead of silently discarding every pair whose argmax class is not contradiction.

A larger and independently annotated benchmark. The coarse-grained recall figures and the single-annotator labelling limit the statistical strength of the conclusions. Expanding the test set with a greater number and variety of naturally occurring conflicts, and introducing multiple independent

annotators with an inter-annotator agreement measure, would place the results on firmer empirical ground.

Human-in-the-loop deployment and study. Given the ethical and legal constraints discussed in Section 8.3, a natural next step is to embed the pipeline in a reviewer-facing workflow and study it in use: how compliance professionals interact with the proposed verdicts and rewrites, whether the traceability mechanisms make verification practical at scale, and where automation genuinely accelerates expert review.

Bibliography

[1] Douglas W. Arner, Janos Barberis, and Ross P. Buckley. *FinTech and RegTech in a Nutshell, and the Future in a Sandbox*. 2017.

[2] Kevin D. Ashley. *Artificial Intelligence and Legal Analytics*. Cambridge University Press, 2017.

[3] Michael Bommarito and Daniel Katz. "GPT Takes the Bar Exam". In: *arXiv preprint arXiv:2212.14402* (2022).

[4] Samuel R. Bowman et al. "A Large Annotated Corpus for Learning Natural Language Inference". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 632–642. DOI: 10.18653/v1/D15-1075.

[5] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 1877–1901.

[6] Celery Project. *Celery: Distributed Task Queue*. Framework for concurrent and asynchronous task management. 2025. URL: https://docs.celeryproject.org/en/stable/.

[7] Ilias Chalkidis et al. "LEGAL-BERT: The Muppets straight out of Law School". In: (2020).

[8] Ilias Chalkidis et al. "LEGAL-BERT: The Muppets Straight Out of Law School". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, 2020, pp. 2898–2904.

[9] Ilias Chalkidis et al. "LexGLUE: A Benchmark Dataset for Legal Language Understanding in English". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. 2022, pp. 4310–4330. DOI: 10.18653/v1/2022.acl-long.297.

[10] Charlie Chen et al. "Accelerating Large Language Model Decoding with Speculative Sampling". In: *arXiv preprint arXiv:2302.01318* (2023).

[11] Chroma. *Chroma Documentation*. https://docs.trychroma.com. Accessed: 2026-07-08. 2025.

[12] Chroma. *ChromaDB: The Open-Source Embedding Database*. https://www.trychroma.com/. Accessed: 2025-02-01. 2023.

[13] Robert Dale. "Law and Word Order: NLP in Legal Tech". In: *Natural Language Engineering* 25.1 (2019), pp. 211–217.

[14] DeepSeek-AI. "DeepSeek-V3 Technical Report". In: *arXiv preprint arXiv:2412.19437* (2024).

[15] Django Software Foundation. *Django Web Framework*. Framework for web application development. 2024. URL: https://www.djangoproject.com/.

[16] Docker Inc. *Docker Documentation*. Platform for developing software containers. 2024. URL: https://docs.docker.com.

[17] European Commission. *Joint Practical Guide of the European Parliament, the Council and the Commission for persons involved in the drafting of European Union legislation*. https://eur-lex.europa.eu. 2023.

[18] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data*. Official Journal of the European Union, L 119. Accessed: 2026-09-14. 2016. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[19] European Parliament and Council of the European Union. *Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence*. Official Journal of the European Union, L 2024/1689. Accessed: 2026-09-14. 2024. URL: https://eur-lex.europa.eu/eli/reg/2024/1689/oj.

[20] Yunfan Gao et al. "Retrieval-Augmented Generation for Large Language Models: A Survey". In: *arXiv preprint arXiv:2312.10997* (2023).

[21] Neel Guha et al. "LegalBench: A Collaboratively Built Benchmark for Measuring Legal Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems* 36 (2023). URL: https://arxiv.org/abs/2308.11462.

[22] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale Similarity Search with GPUs". In: *IEEE Transactions on Big Data* (2019).

[23] Pratik Joshi et al. “The XTREME Benchmark”. In: *ICML* (2020).

[24] Greg Kamradt. *5 Levels of Text Splitting*. `https://github.com/FullStackRetrieval-com/RetrievalTutorials`. Accessed: 2025-01-15. 2023.

[25] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[26] Daniel Martin Katz et al. “GPT-4 Passes the Bar Exam”. In: (2023).

[27] LangChain. *LangChain Documentation*. `https://python.langchain.com`. Accessed: 2026-07-08. 2025.

[28] LangChain. *PyPDFLoader*. 2024. URL: `https://python.langchain.com/docs/integrations/document_loaders/pypdfloader/`.

[29] LangChain. *Text Splitters*. 2024. URL: `https://python.langchain.com/docs/concepts/text_splitters/`.

[30] LangChain Team. *Plan-and-Execute Agents*. Accessed: 2026-07-24. 2024. URL: `https://www.langchain.com/blog/planning-agents`.

[31] Yaniv Leviathan, Matan Kalman, and Yossi Matias. “Fast Inference from Transformers via Speculative Decoding”. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 2023.

[32] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems*. 2020.

[33] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 9459–9474.

[34] Chris Lu, Cong Lu, Robert Tjarko Lange, et al. “The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery”. In: *arXiv preprint arXiv:2408.06292* (2024).

[35] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[36] Microsoft. *Azure OpenAI Service Documentation*. https://learn.microsoft.com/en-us/azure/ai-services/openai/. Accessed: 2026-07-08. 2025.

[37] Microsoft. *Azure OpenAI Service Pricing*. https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/. Accessed: 2025-03-01. 2024.

[38] Microsoft. *Hybrid Search Using Reciprocal Rank Fusion*. https://learn.microsoft.com/en-us/azure/search/hybrid-search-ranking. Accessed: 2026-07-24. 2025.

[39] James O'Neill et al. "Classifying Sentential Modality in Legal Language: A Use Case in Financial Regulations, Acts and Directives". In: *Proceedings of the 16th International Conference on Artificial Intelligence and Law (ICAIL)*. ACM, 2017, pp. 159–168.

[40] OpenAI. *Introducing Structured Outputs in the API*. 2024. URL: https://openai.com/index/introducing-structured-outputs-in-the-api/.

[41] OpenAI. *Whisper*. Open-source AI model for automatic speech recognition. 2022. URL: https://github.com/openai/whisper.

[42] Long Ouyang et al. "Training Language Models to Follow Instructions with Human Feedback". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 27730–27744.

[43] Michal Ovádek. *Legislative Output of the European Union*. Based on EUR-Lex data. 2026. URL: https://michalovadek.github.io/eulaw/.

[44] David M. W. Powers. "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation". In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63.

[45] Pydantic. *Pydantic Documentation*. https://docs.pydantic.dev. Accessed: 2026-07-08. 2025.

[46] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2019, pp. 3982–3992.

[47] Stephen Robertson and Hugo Zaragoza. “The Probabilistic Relevance Framework: BM25 and Beyond”. In: *Foundations and Trends in Information Retrieval* 3.4 (2009), pp. 333–389.

[48] Jaromír Savelka and Kevin D. Ashley. “The Unreasonable Effectiveness of Large Language Models in Zero-Shot Semantic Annotation of Legal Texts”. In: *Frontiers in Artificial Intelligence*. Vol. 6. Frontiers Media, 2023.

[49] Timo Schick et al. “Toolformer: Language Models Can Teach Themselves to Use Tools”. In: *Advances in Neural Information Processing Systems (NeurIPS 2023)*. 2023. URL: https://arxiv.org/abs/2302.04761.

[50] Sentence Transformers. *all-MiniLM-L6-v2 Model Card*. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. 2021.

[51] Sentence Transformers. *cross-encoder/nli-MiniLM2-L6-H768 Model Card*. https://huggingface.co/cross-encoder/nli-MiniLM2-L6-H768. 2024.

[52] Kurt Shuster et al. “Retrieval Augmentation Reduces Hallucination in Conversation”. In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, 2021, pp. 3784–3803.

[53] Linda J. Skitka, Kathleen L. Mosier, and Mark Burdick. “Does Automation Bias Decision-Making?” In: *International Journal of Human-Computer Studies* 51.5 (1999), pp. 991–1006. DOI: 10.1006/ijhc.1999.0252.

[54] SQLAlchemy Project. *SQLAlchemy — The Database Toolkit for Python*. Toolkit and ORM for database management in Python. 2025. URL: https://www.sqlalchemy.org/.

[55] Streamlit. *Streamlit Documentation*. https://docs.streamlit.io. Accessed: 2026-07-08. 2025.

[56] Harry Surden. “Machine Learning and Law”. In: *Washington Law Review* 89.1 (2014), pp. 87–115.

[57] Flower Team. *Flower: A Friendly Federated Learning Framework*. Frontend web for monitoring Celery tasks. 2025. URL: https://flower.dev.

[58] Nandan Thakur et al. "BEIR: A Heterogeneous Benchmark for Zero-Shot Evaluation of Information Retrieval Models". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021. URL: https://openreview.net/forum?id=wCu6T5xFjeJ.

[59] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017.

[60] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[61] Lei Wang, Chen Ma, Xueyang Feng, et al. "A Survey on Large Language Model Based Autonomous Agents". In: *Frontiers of Computer Science* 18.6 (2024). DOI: 10.1007/s11704-024-40231-1.

[62] Wenhui Wang et al. "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[63] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2022).

[64] Jason Wei et al. "Emergent Abilities of Large Language Models". In: *Transactions on Machine Learning Research (TMLR)* (2022).

[65] Adina Williams, Nikita Nangia, and Samuel R. Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2018.

[66] John Yang, Carlos E. Jimenez, Alexander Wettig, et al. "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering". In: *Advances in Neural Information Processing Systems* 37 (2024). URL: https://arxiv.org/abs/2405.15793.

[67] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *International Conference on Learning Representations (ICLR)*. 2023. URL: https://arxiv.org/abs/2210.03629.

[68] Wan Zhang and Jing Zhang. “Hallucination Mitigation for Retrieval-Augmented Large Language Models: A Review”. In: *Mathematics* 13.5 (2025), p. 856.